

Fundamentos de Sistemas Digitais

CIRCUITOS COMBINACIONAIS

Fernando G. Moraes

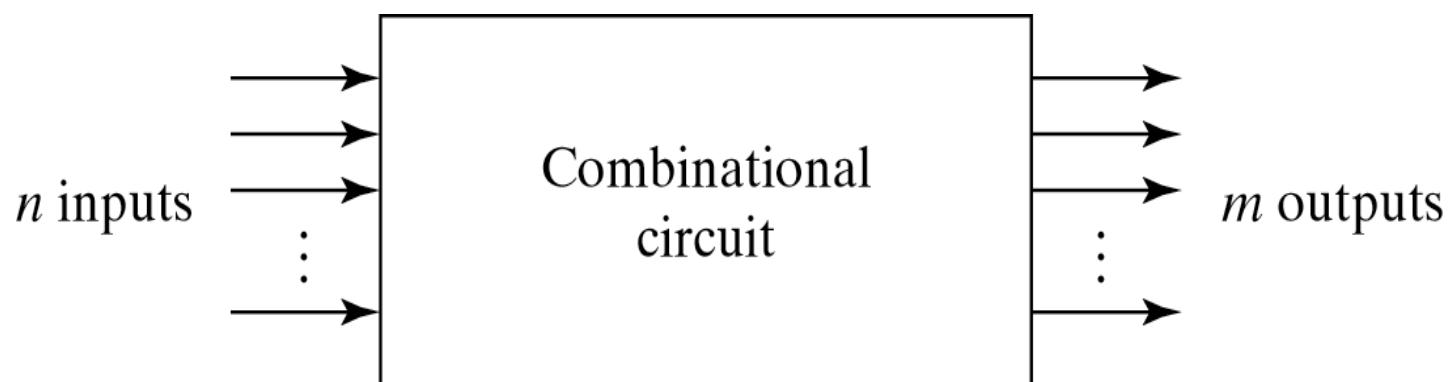
Circuitos Combinacionais

- Um circuito combinacional consiste em portas lógicas cujas saídas, em qualquer momento, são determinadas pela combinação dos valores das entradas
- Para n variáveis de entrada, existem 2^n combinações de entrada binária possíveis
- Para cada combinação binária das variáveis de entrada, existe uma saída possível

Circuitos Combinacionais

Um circuito combinacional pode ser descrito por:

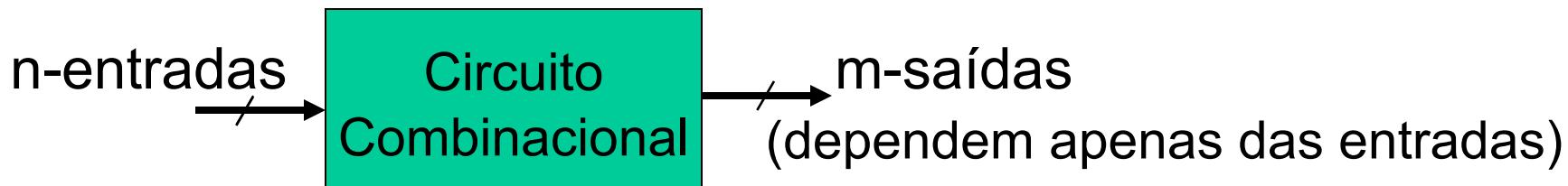
1. Uma **tabela de verdade** que lista os valores de saída para cada combinação das variáveis de entrada, ou
2. m **funções booleanas**, uma para cada variável de saída.



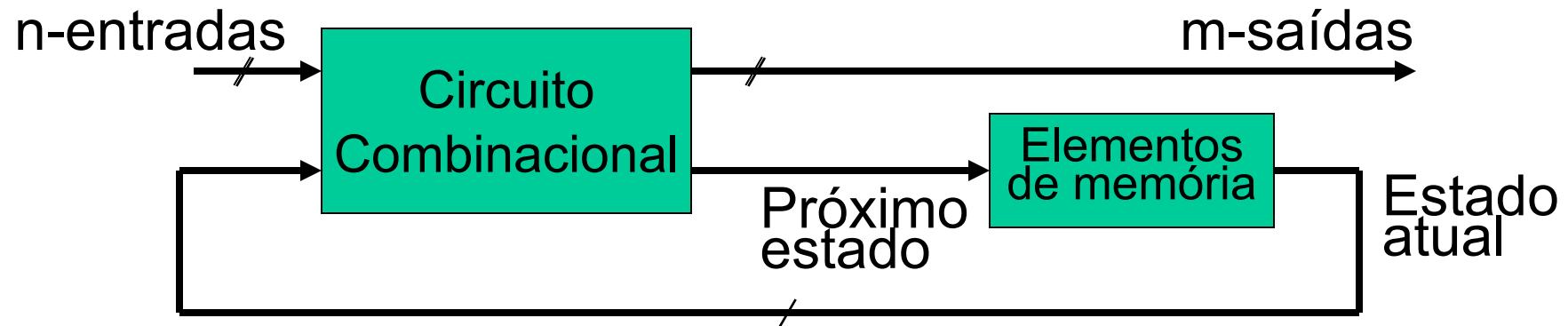
Circuitos Combinacionais versus Sequenciais

- Os circuitos combinacionais não possuem memória interna
 - O valor de saída depende apenas dos valores atuais de entrada
- Os circuitos sequenciais contêm lógica combinacional, e elementos de memória (usados para armazenar estados de circuito)
 - As saídas dependem dos valores de entrada atuais **e** dos valores de entrada anteriores (mantidos nos elementos de memória)

Circuitos Combinacionais versus Sequenciais



Circuito Combinacional



Circuito Sequencial

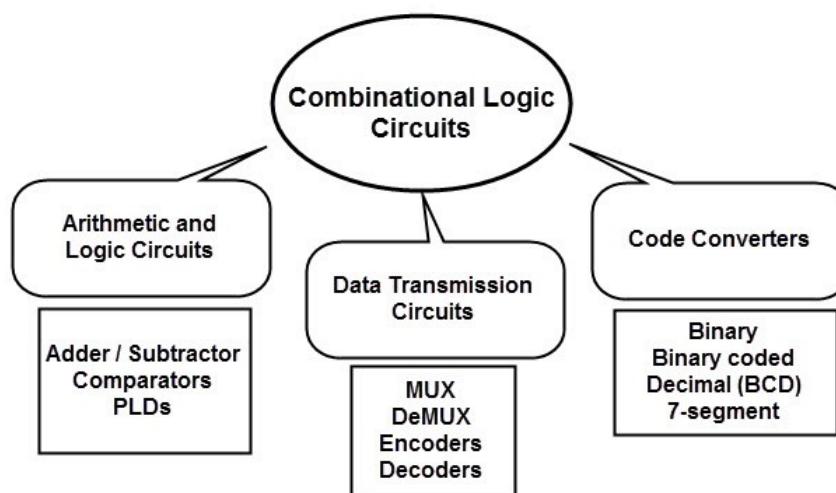
Circuitos básicos

□ Exemplos de circuitos combinacionais

1. Codificador/decodificador
2. Comparadores
3. Geradores de paridade
4. Multiplexador/demux
5. PLAs
6. Memórias ROM
7. Somador / Subtrator
8. ULA
9. Multiplicadores / Divisores

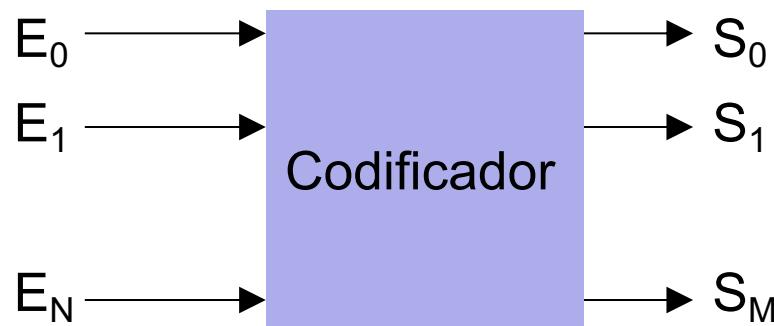
□ Exemplos de circuitos seqüenciais

1. Registradores (deslocamento, carga paralela, acumulador, serial-paralelo)
2. Contadores (binário, BCD, Johnson, Gray / up, down, up-down)
3. Máquina de Estados
4. Geradores de clock
5. Memória RAM



(1) (DE)CODIFICADOR

- Codificador é um circuito que mapeia um conjunto de entradas em um conjunto de saídas segundo uma função de codificação
 - Em outras palavras, é um circuito que transforma uma informação de um formato para outro
- Representação gráfica de codificador genérico



Codificador $2^n \rightarrow n$

Decodificador $n \rightarrow 2^n$

DECODIFICADOR - 4-Bit Decoder ($n \rightarrow 2^n$)

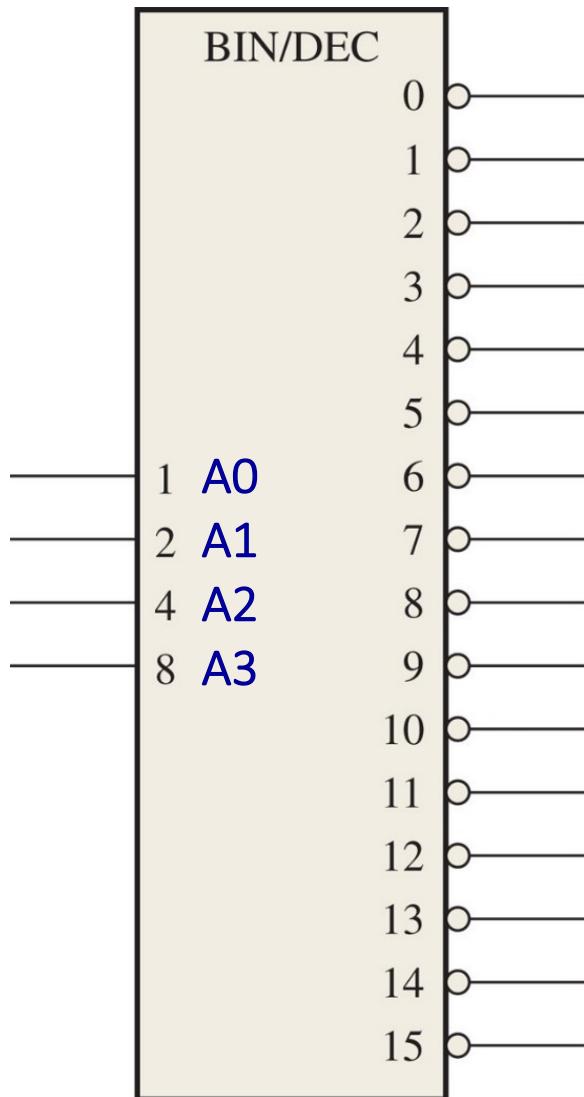


FIGURE 6-28 Logic symbol for a 4-line-to-16-line (1-of-16) decoder.

DECODIFICADOR - 4-Bit Decoder ($n \rightarrow 2^n$)

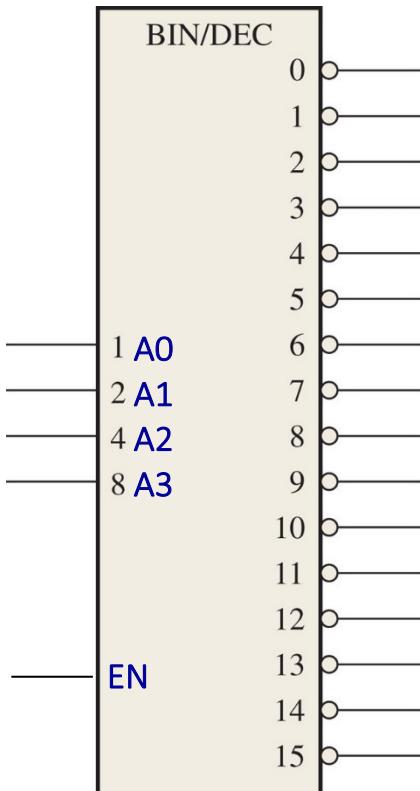
Apenas uma saída ativa para cada valor binário de entrada

TABLE 6-4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

Decimal Digit	Binary Inputs				Decoding Function	Outputs														
	A_3	A_2	A_1	A_0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
10	1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
11	1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
12	1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
13	1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

DECODIFICADOR - 4-Bit Decoder ($n \rightarrow 2^n$)



```

OUT0 <= not(not A0 and not A1 and not A2 and not A3 and EN);
OUT1 <= not(A0 and not A1 and not A2 and not A3 and EN);
OUT2 <= not(not A0 and A1 and not A2 and not A3 and EN);
OUT3 <= not(A0 and A1 and not A2 and not A3 and EN);
OUT4 <= not(not A0 and not A1 and A2 and not A3 and EN);
OUT5 <= not(A0 and not A1 and A2 and not A3 and EN);
OUT6 <= not(not A0 and A1 and A2 and not A3 and EN);
OUT7 <= not(A0 and A1 and A2 and not A3 and EN);
OUT8 <= not(not A0 and not A1 and not A2 and A3 and EN);
OUT9 <= not(A0 and not A1 and not A2 and A3 and EN);
OUT10 <= not(not A0 and A1 and not A2 and A3 and EN);
OUT11 <= not(A0 and A1 and not A2 and A3 and EN);
OUT12 <= not(not A0 and not A1 and A2 and A3 and EN);
OUT13 <= not(A0 and not A1 and A2 and A3 and EN);
OUT14 <= not(not A0 and A1 and A2 and A3 and EN);
OUT15 <= not(A0 and A1 and A2 and A3 and EN);

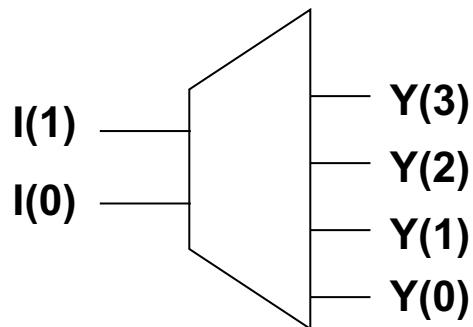
```

EN: *enable* – sinal adicional que habilita o codificador

DECODIFICADOR – portas lógicas ($n \rightarrow 2^n$)

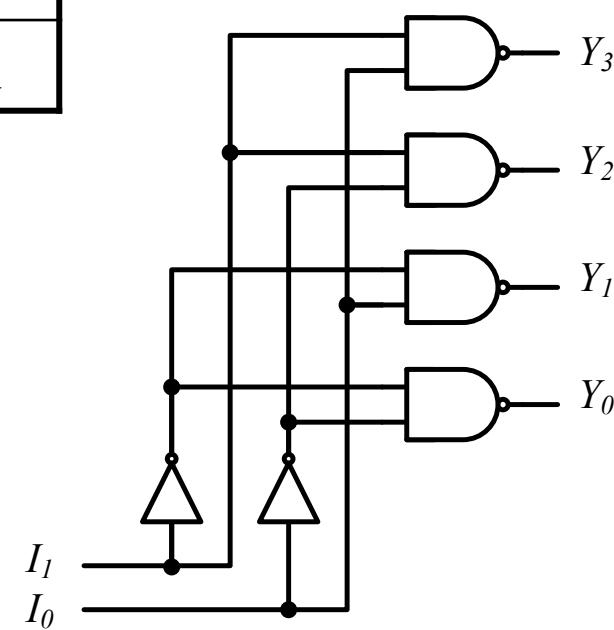
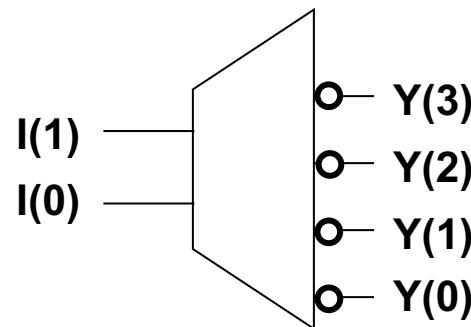
Ativo alto

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

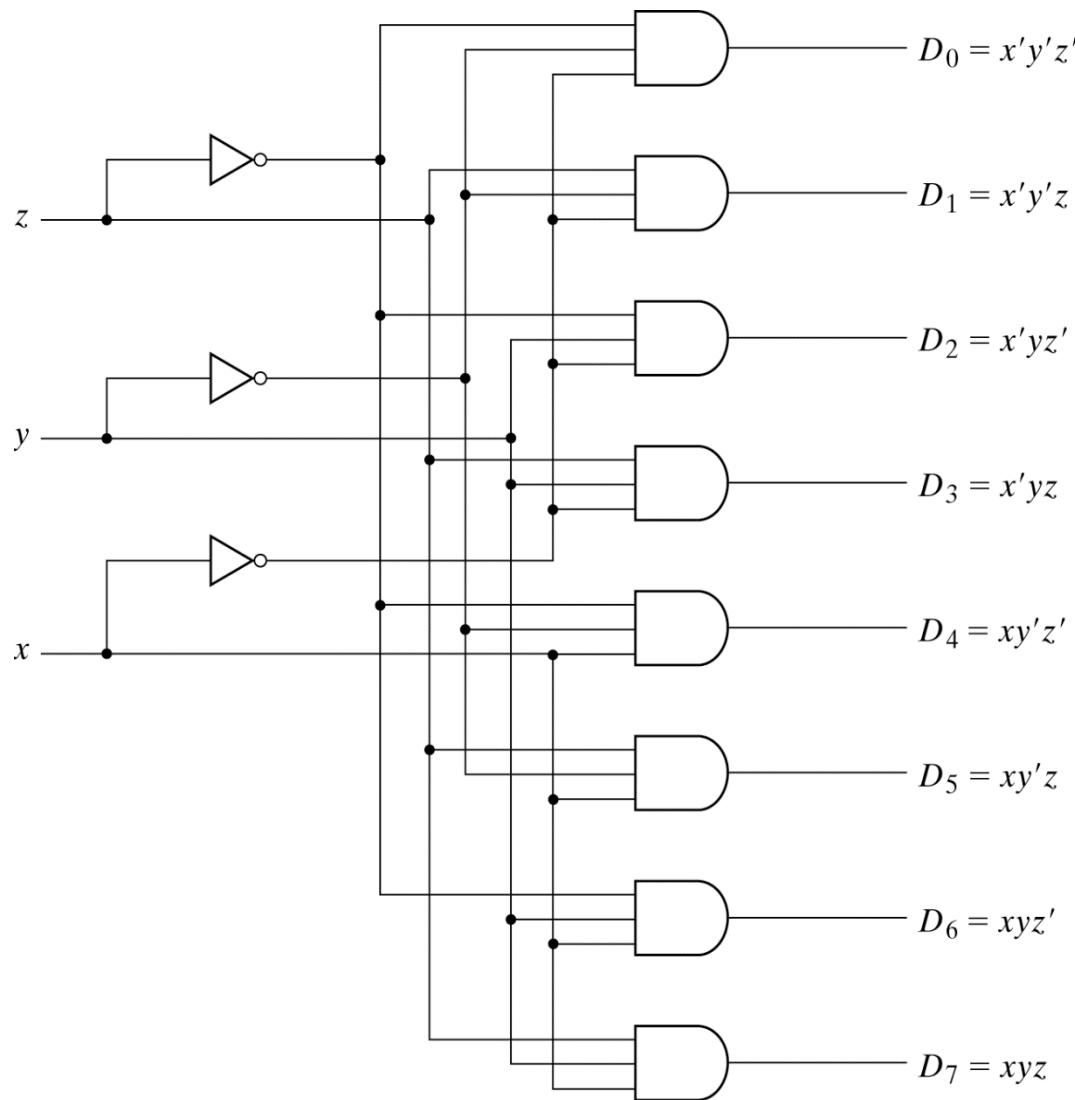


Ativo baixo

I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1



DECODIFICADOR 3→8 (ativo alto, sem enable)

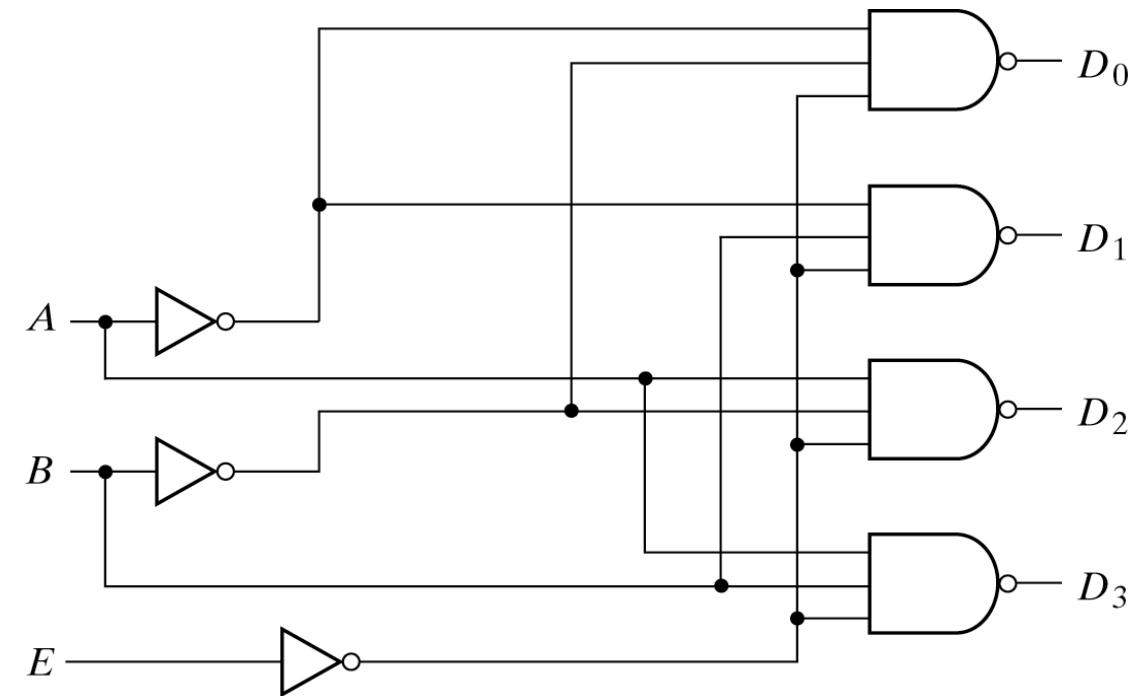


Cada porta **and** recebe
um determinado
endereço

Fig. 4-18 3-to-8-Line Decoder

DECODIFICADOR – com sinal de habilitação

Conforme indicado pela tabela de verdade, apenas uma saída pode ser igual a 0 a qualquer momento, todas as outras saídas são iguais a 1.



(a) Logic diagram

E	A	B	D ₀	D ₁	D ₂	D ₃
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

CODIFICADOR – portas lógicas ($2^n \rightarrow n$)

- Neste exemplo temos 10 entradas, correspondendo a sinais que representam um valor decimal
- 4 bits com a codificação decimal

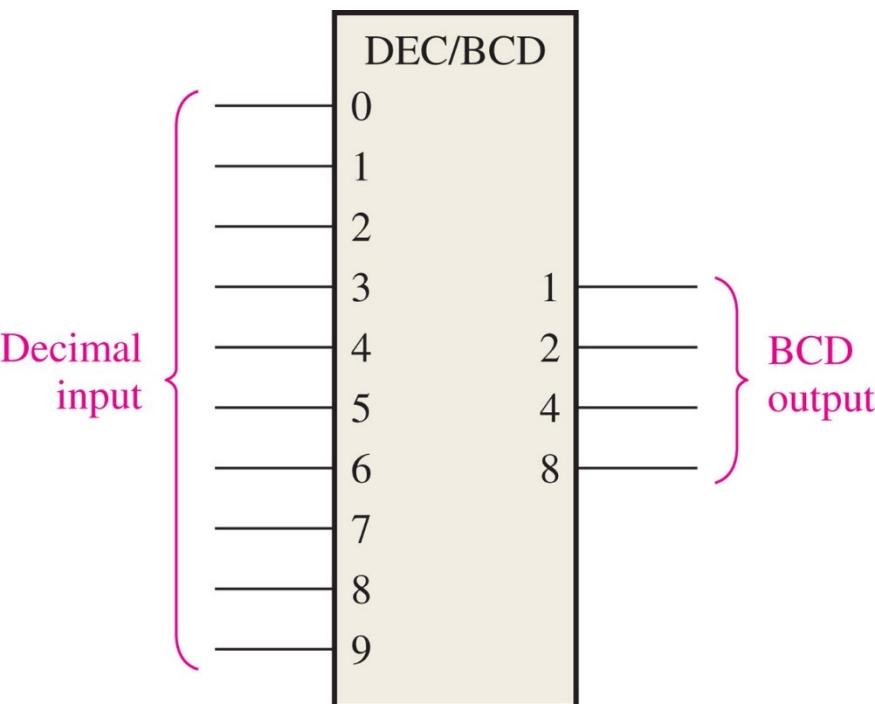


TABLE 6-6

Decimal Digit	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

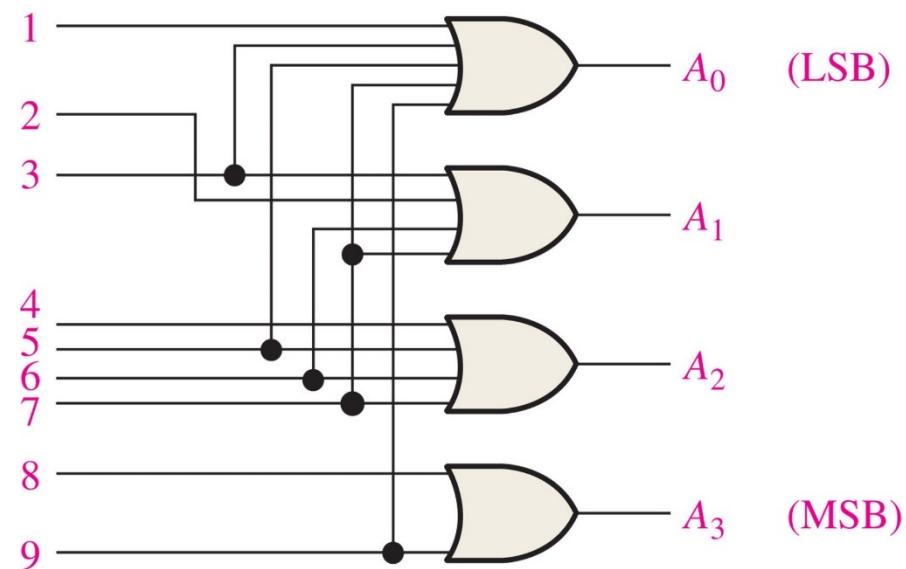
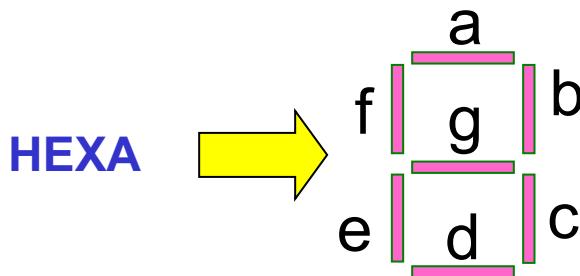


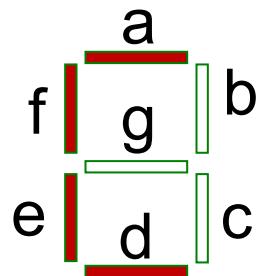
FIGURE 6-36 Logic symbol for a decimal-to-BCD encoder.

CODIFICADOR – hexa para sete-segmentos

- Este exemplo ilustra um codificador que tem por entrada um valor **hexadecimal** (0-F) para sete segmentos.
 - Entrada em 4 bits: A3 / A2 / A1 / A0
 - Saída em 7 bits: a, b, c, d, e ,f, g



Entrada C (1100) e saída:



A3	A2	A1	A0	a	b	c	...
0	0	0	0	1			
0	0	0	1				
0	0	1	0	1			
0	0	1	1	1			
0	1	0	0				
0	1	0	1	1			
0	1	1	0	1			
0	1	1	1	1			
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	1			
1	0	1	1				
1	1	0	0	1			
1	1	0	1				
1	1	1	0	1			
1	1	1	1	1			

- 7 mapas de Karnaugh, um para cada dígito

Display 7 segmentos

	CD	00	01	11	10
AB		00			
00		1		1	1
01		1	1	1	1
11		1		1	1
10		1	1		1

A

Ativo em 0, 2, 3, 5, 6, 7,.....
 $AB'C'$ $A'BD$ $B'D'$ $A'C$ AD' BC

	CD	00	01	11	10
AB		00			
00		1	1	1	1
01		1		1	
11			1		
10		1	1		1

B

	CD	00	01	11	10
AB		00			
00		1	1	1	
01		1	1	1	1
11		1			
10		1	1		1

C

	CD	00	01	11	10
AB		00			
00		1			1
01			1		1
11		1	1		1
10		1	1		

D

	CD	00	01	11	10
AB		00			
00		1			1
01			1		1
11		1	1	1	1
10		1		1	1

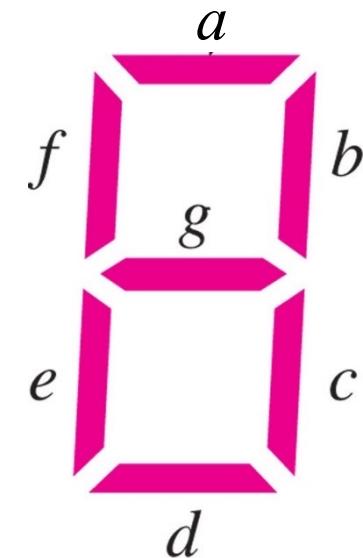
E

	CD	00	01	11	10
AB		00			
00		1			
01		1	1		1
11		1	1	1	1
10		1		1	1

F

	CD	00	01	11	10
AB		00			
00				1	1
01		1	1		1
11		1	1	1	1
10		1		1	1

G



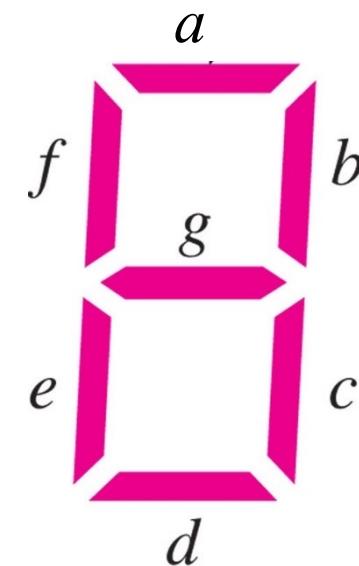
Display 7 segmentos – simplificando com VHDL

with **vetorEnt** select

Saida <=

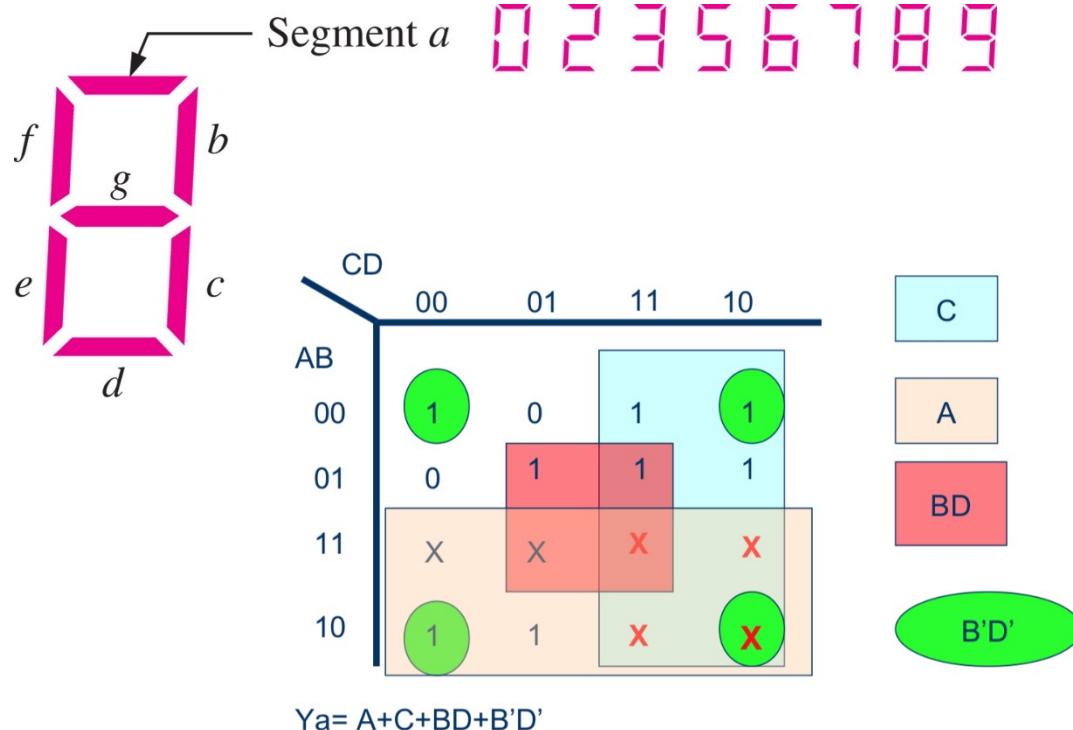
"1111110"	when "0000",
"0110000"	when "0001",
"1101101"	when "0010",
"1111001"	when "0011",
"0110011"	when "0100",
"1011011"	when "0101",
"1011111"	when "0101",
"1110000"	when "0111",
"1111111"	when "1000",
"1110011"	when "1001",
"1110111"	when "1010",
"0011111"	when "1011",
"1001110"	when "1100",
"0111101"	when "1101",
"1001111"	when "1110",
"1000111"	when "1111";

vetorEnt: valor hexa



CODIFICADOR decimal – uso de don't cares

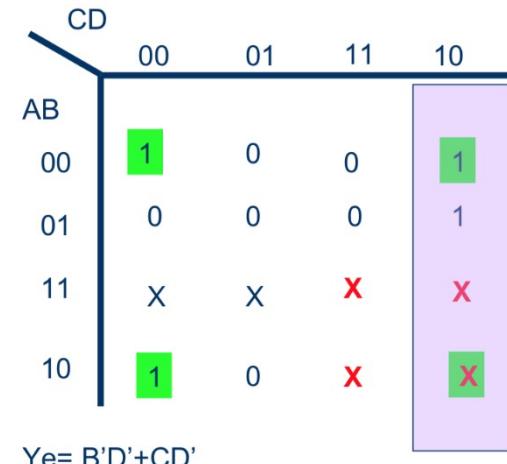
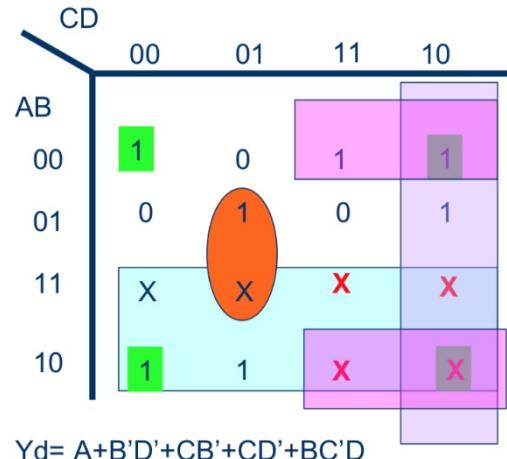
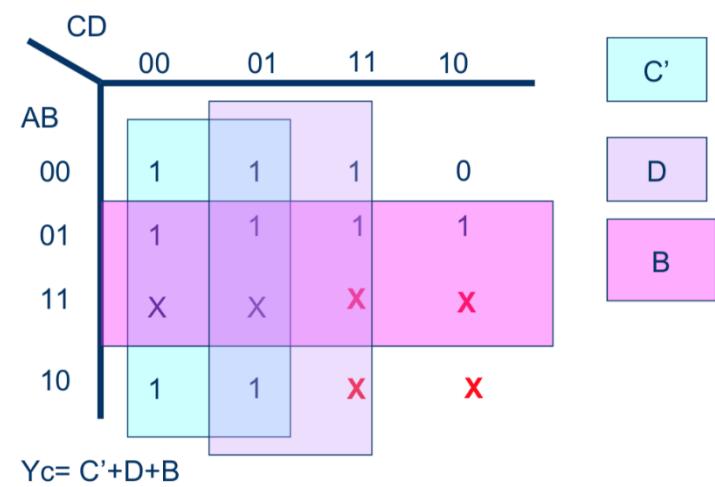
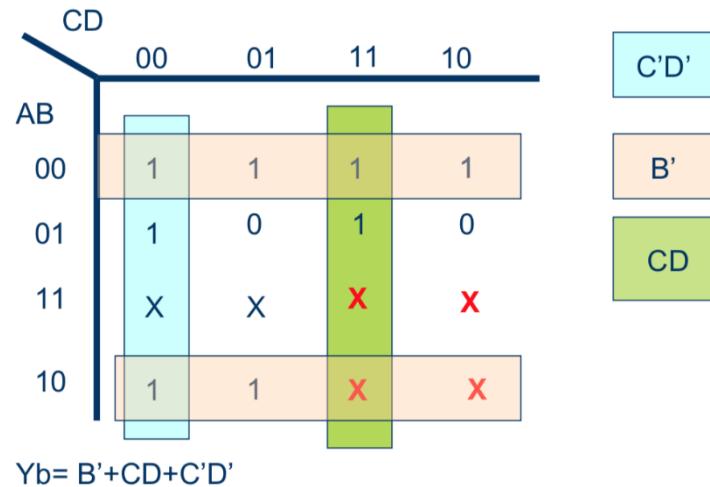
- As entradas só variam entre 0 e 9 (código BCD)
 - Entrada em 4 bits: A3 / A2 / A1 / A0
 - Saída em 7 bits: a, b, c, d, e ,f, g



A3	A2	A1	A0	a	b	c	...
0	0	0	0	1			
0	0	0	1				
0	0	1	0	1			
0	0	1	1	1			
0	1	0	0				
0	1	0	1	1			
0	1	1	0	1			
0	1	1	1	1			
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	x			
1	0	1	1	x			
1	1	0	0	x			
1	1	0	1	x			
1	1	1	0	x			
1	1	1	1	x			

- 7 mapas de Karnaugh, um para cada dígito

CODIFICADOR – decimal para sete-segmentos



CODIFICADOR COM PRIORIDADE ($2^n \rightarrow n$)

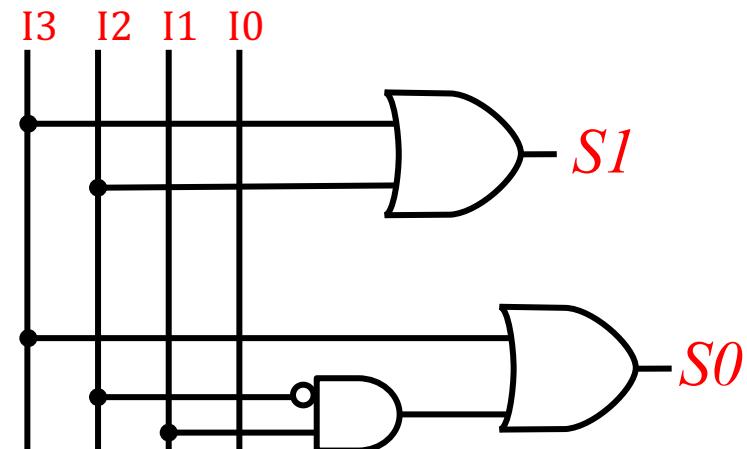
Codificador com prioridade

- Em um codificador com prioridade se o bit menos significativo for ‘1’ a saída é ‘0’, se o bit seguinte for 1, independentemente do anterior, a saída é ‘1’; e assim sucessivamente.
- Exemplo - I3 tem maior prioridade (todos bits em 0 não ocorre) :

I3	I2	I1	I0	S1	S0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

$$S1 = I3 + I2$$

$$S0 = I3 + \bar{I2} \cdot I1$$

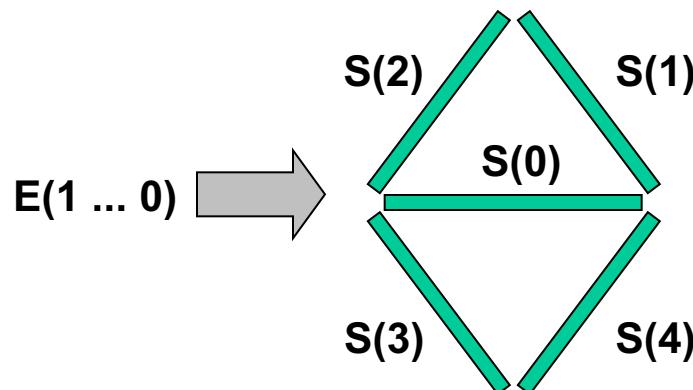


- Circuito resultante:

DECODIFICADOR - exercício

EXERCÍCIOS:

- Faça a codificação do display de elevador ilustrado abaixo



- Este tem como entrada um vetor de 2 bits que recebe a seguinte codificação:

parado	00	-
subindo	01	Λ
descendo	10	∨
defeito	11	todos segmentos acesos

E1	E0	S4	S3	S2	S1	S0
0	0	0	0	0	0	1
0	1	0	0	1	1	0
1	0	1	1	0	0	0
1	1	1	1	1	1	1

$$\begin{aligned}S4 &= S3 = E1 \\S2 &= S1 = E0 \\S0 &= E1 \text{ xnor } E0\end{aligned}$$

Decodificador utilizado como gerador de mintermos

Pode-se utilizar um decodificador para realizar funções de n variáveis

Exemplo para o somador:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Cada saída do decodificador pode ser vista como uma linha da tabela verdade (ou *minterm*)

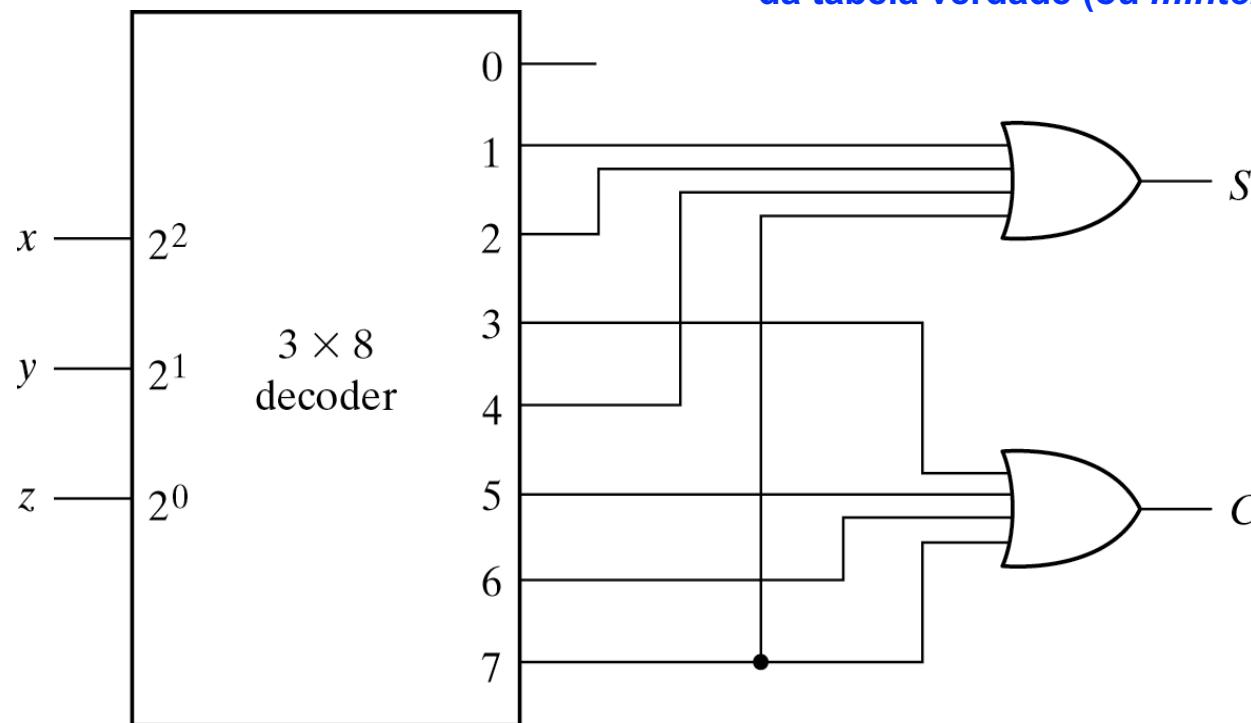
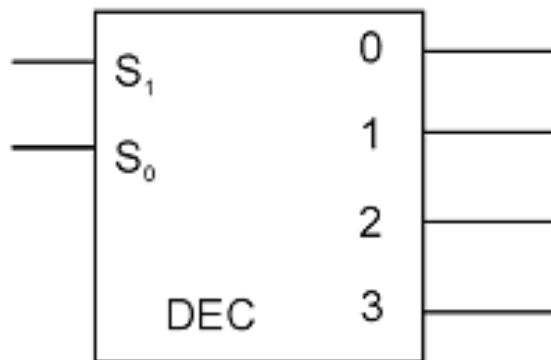


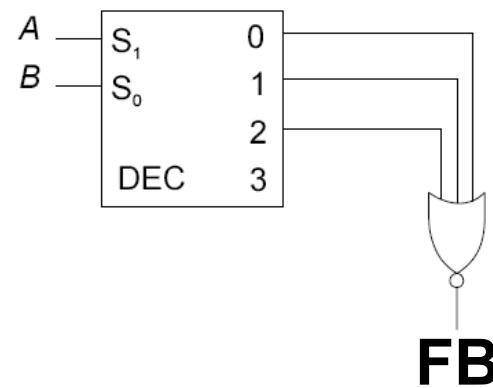
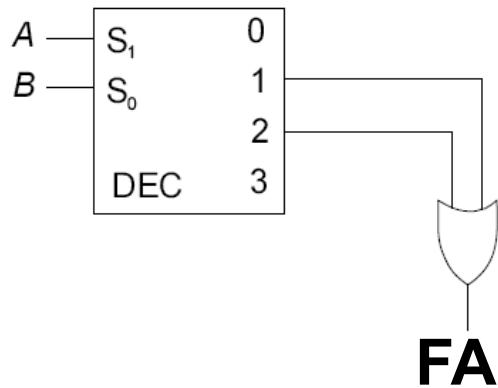
Fig. 4-21 Implementation of a Full Adder with a Decoder

Decodificador como gerado de funções

Considere o **decodificador** abaixo e sua correspondente tabela verdade. Determine as funções lógicas a seguir:

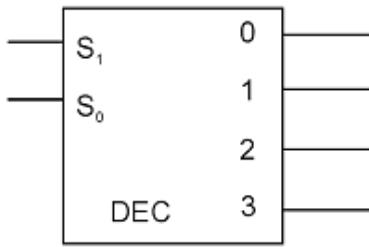


entradas		saídas			
S ₁	S ₀	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

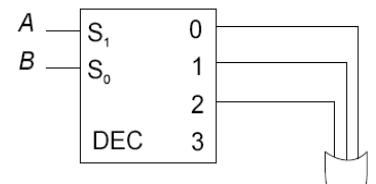
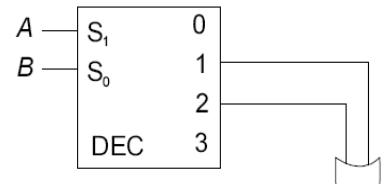


$$\begin{aligned} FB &= \text{not} (S_1'S_0' + S_1'S_0 + S_1S_0') \\ &= \text{not}(S_1'(S_0' + S_0) + S_0'(S_1' + S_1)) \\ &= \text{not}(S_1' + S_0') \\ &= S_1 \cdot S_0 \end{aligned}$$

Decodificador: SOLUÇÃO



entradas		saídas			
S_1	S_0	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



FA

FB

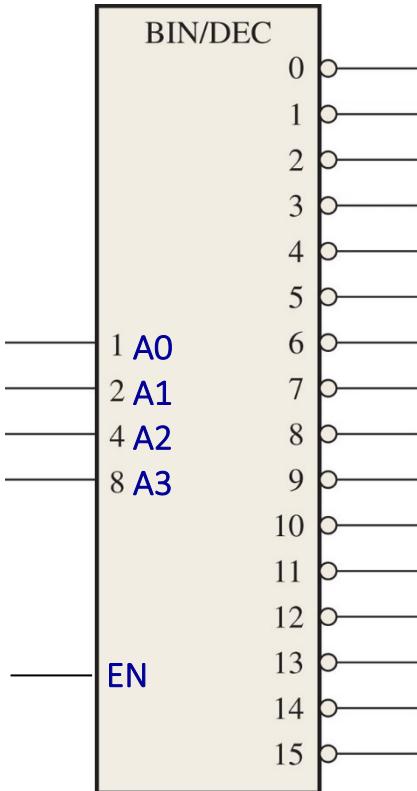
0 $A'B'$	1 $A'B$	2 AB'	3 AB	FA 1 or 2	F2 (0 or 1 or 2)'
1	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	1	0
0	0	0	1	0	1

XOR

AND

Exercício

- Supondo que desejamos realizar o acesso a uma linha de dados de uma memória de 64 palavras, mas possuímos decodificadores $4 \rightarrow 16$ e $2 \rightarrow 4$. Como podemos construir um decodificador $6 \rightarrow 64$ utilizando estes decodificadores?



Solução: dividir o endereço em parte alta e baixa

endereço:

A5 A4 A3 A2 A1 A0

Parte alta **Parte baixa**

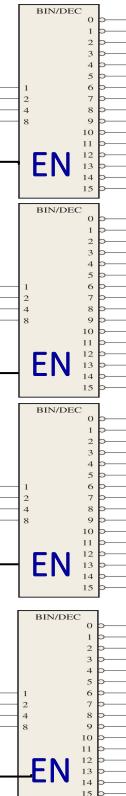
Parte baixa
A3..A0

Parte alta

A5 S₁ 0
A4 S₀ 1
DEC 2
A3..A0 3

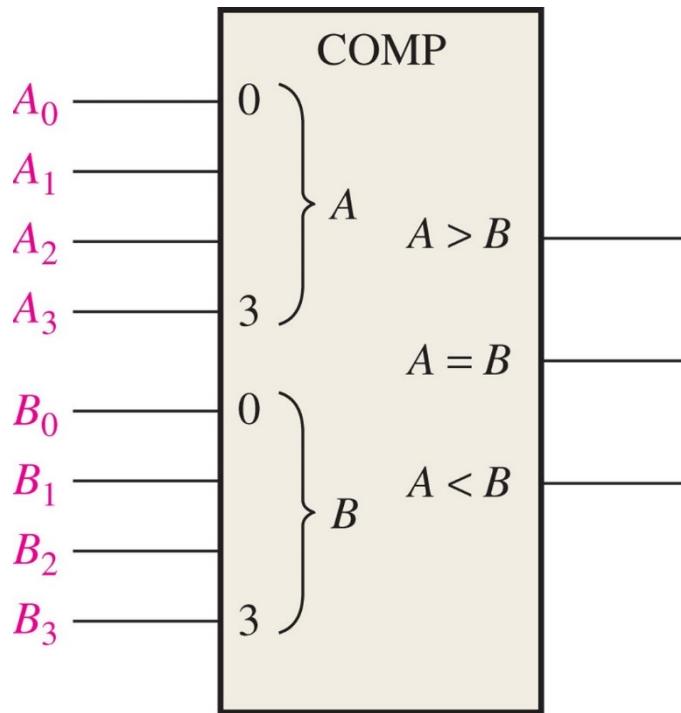
A3..A0

A3..A0



(2) COMPARADOR

- Dois números hexadecimal como entradas, 3 saídas



XNOR		
Inputs	Output	
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

- Circuito para detectar igualdade:

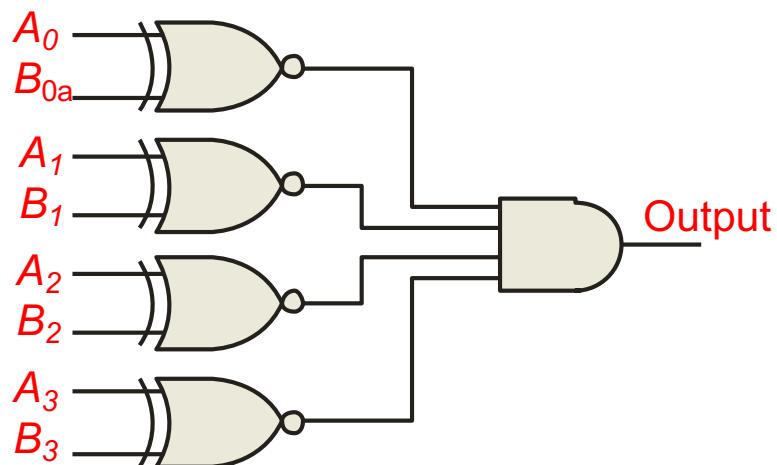
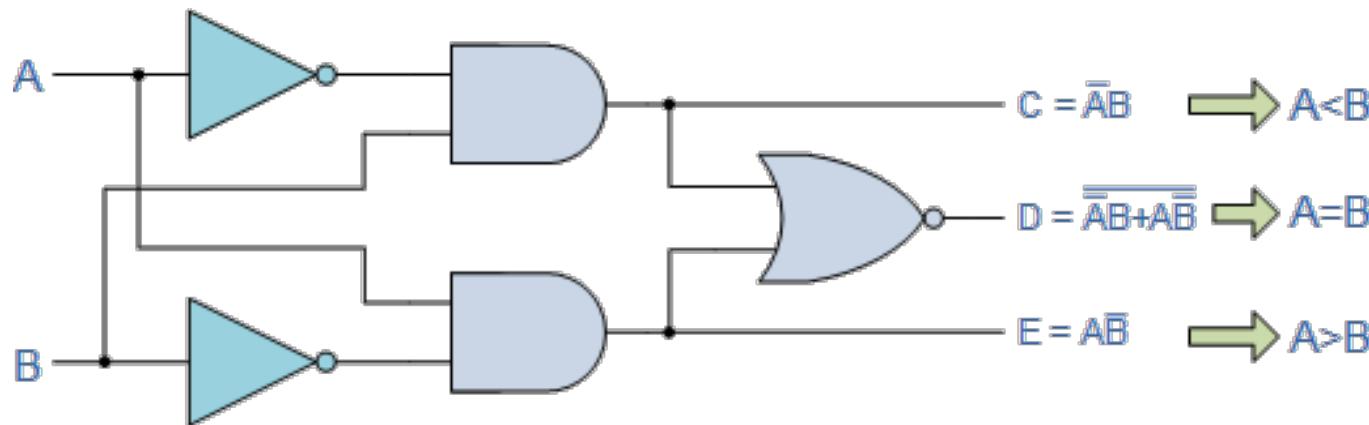


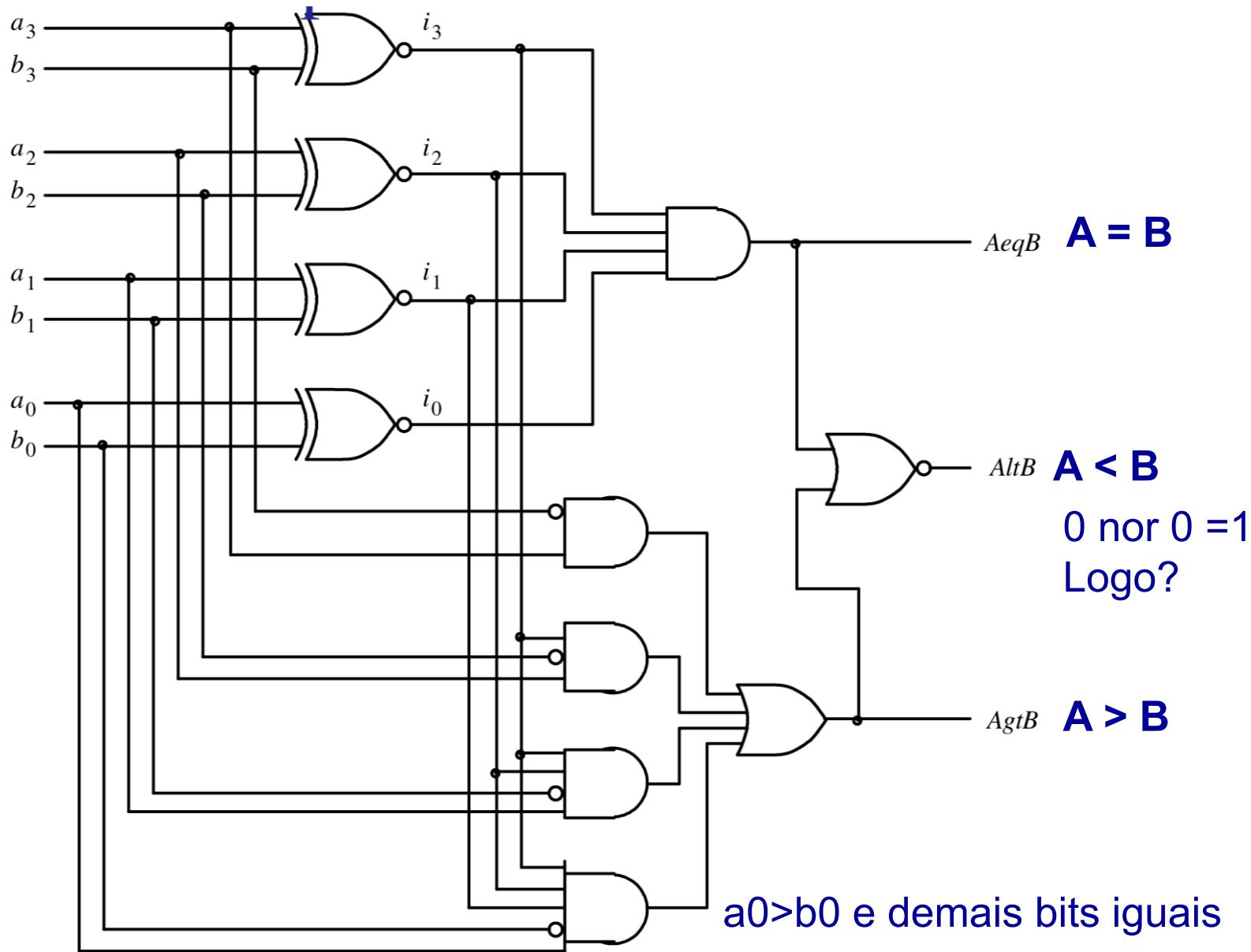
FIGURE 6-21 Logic symbol for a 4-bit comparator with inequality indication.

COMPARADOR DE 1 BIT

Fazer a tabela verdade para as saídas C / D / E :



COMPARADOR DE DUAS PALAVRAS DE 4 BITS



COMPARADOR – com VHDL (simplifica a modelagem)

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all; --****--
4
5 entity comparador is
6   generic(N : integer := 4);
7   port(A, B : in std_logic_vector(N-1 downto 0);
8         igual  : out std_logic;
9         maior  : out std_logic;
10        menor  : out std_logic
11      );
12 end entity;
13
14
15 architecture a1 of comparador is
16 begin
17
18   igual <= '1' when A = B else '0';
19   maior <= '1' when A > B else '0';
20   menor <= '1' when A < B else '0';
21
22 end architecture a1;

```

igual <= '1' when A = B else '0' ;
maior <= '1' when A > B else '0' ;
menor <= '1' when A < B else '0' ;



(3) Gerador de paridade e verificação de paridade

- Gerador de paridade - cria o bit de *paridade*, adicionado a uma palavra
- Verificador de paridade – calcula e verifica a paridade para garantir que não haja erro na palavra recebida

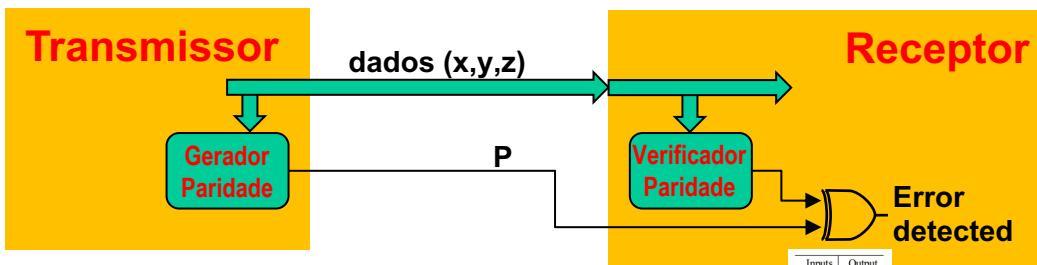
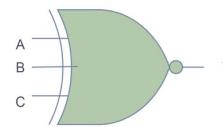


Table 3.11: Parity Generator

x	y	z	Parity Bit
0	0	0	1
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	0



XNor de n entradas: '1' quando número de bits em '1' é PAR ou todos bits são '0'

Em computadores:

ECC memory – Error-correcting code memory

Table 3.12: Parity Checker

x	y	z	P	Error Detected?
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Gerador de paridade e verificação de paridade

- Exemplo de gerador de bit de paridade com portas xor (negada)

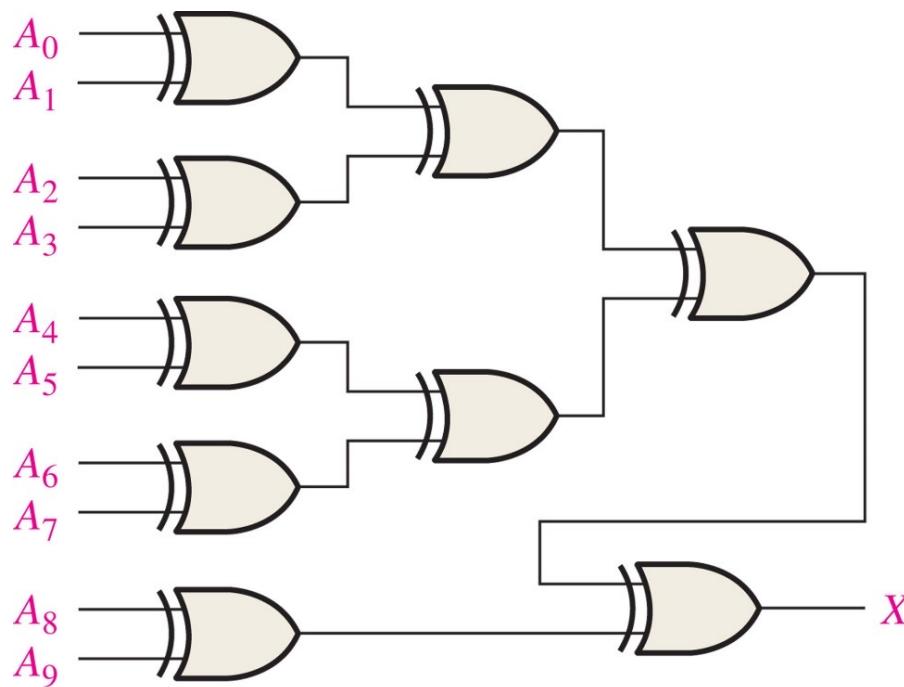


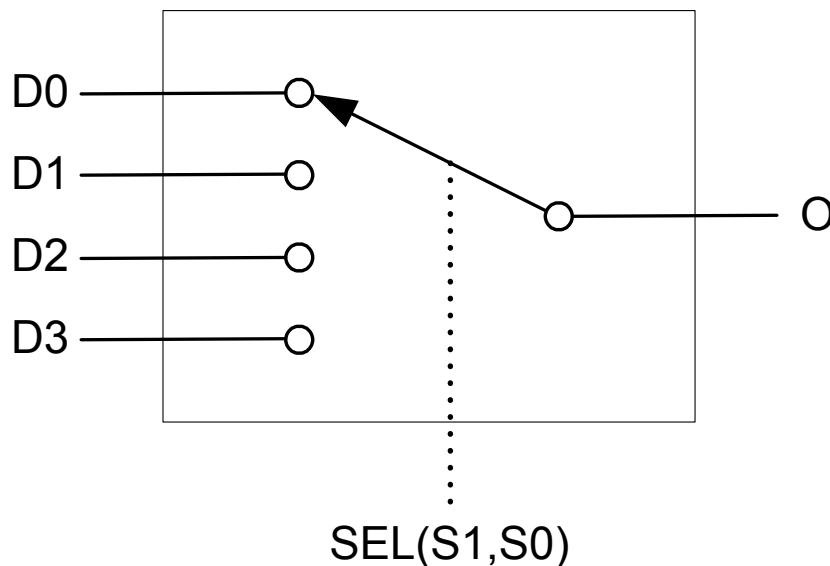
FIGURE 6-57

Uso de portas XOR

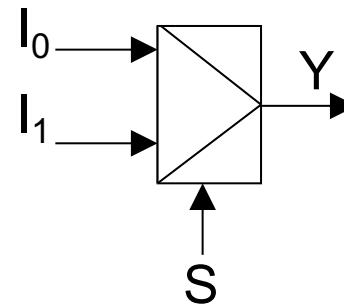
Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

(4) MULTIPLEXADOR

- É um circuito que permite selecionar uma dentre várias entradas em função de uma variável de controle



2-to-1 MUX

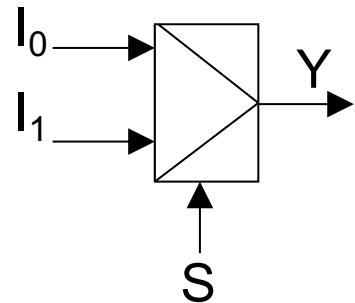


- EQUIVALENTE EM SOFTWARE: *if then else*

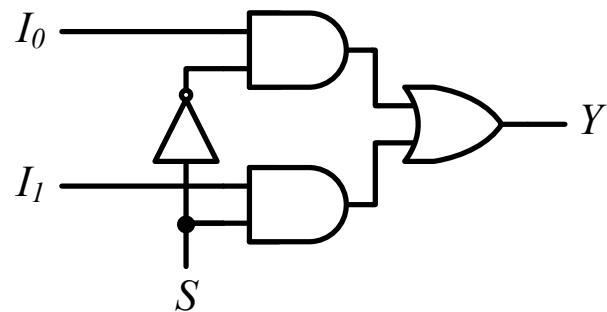
Multiplexadores

MUX 2:1

Símbolo:

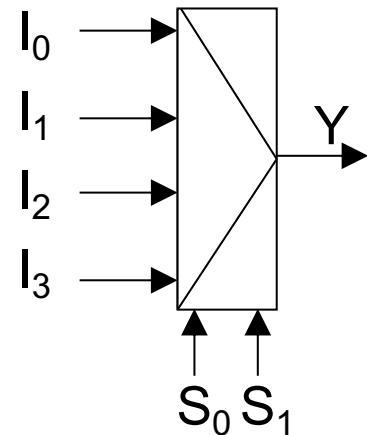


Estrutura Interna:

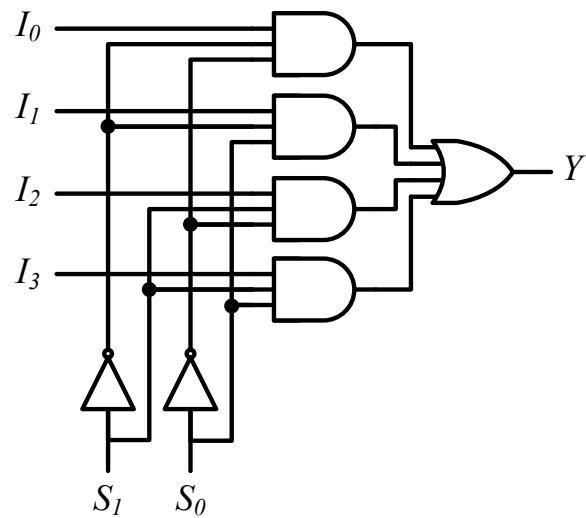


MUX 4:1

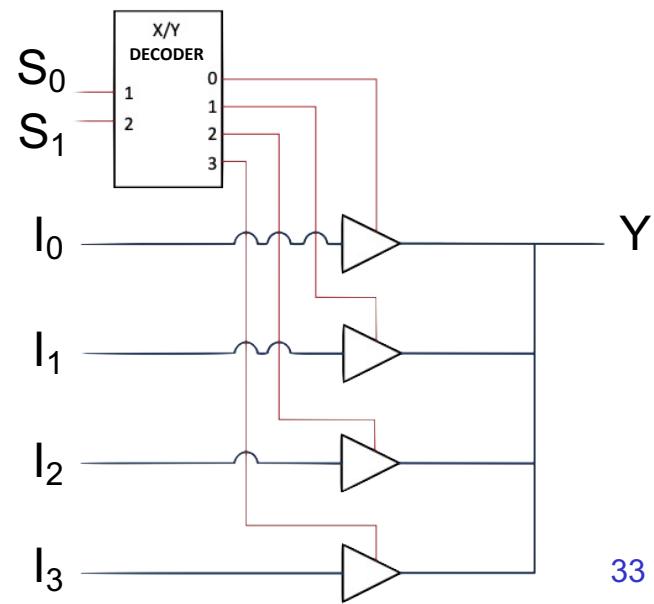
Símbolo:



Estrutura Interna:



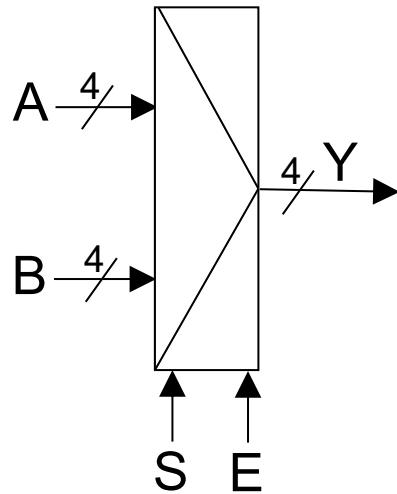
Estrutura Interna Alternativa:



Multiplexadores (2x1 - 4 bits com enable)

MUX 2:1 de 4 bits com enable:

Símbolo:



Estrutura Interna:

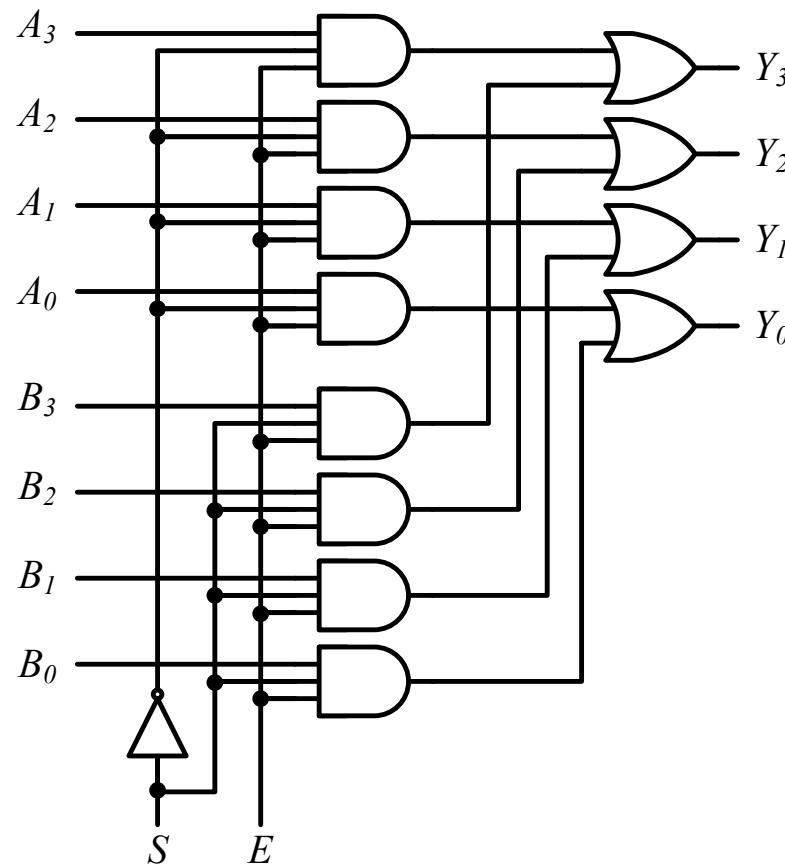


Tabela Verdade:

E	S	Y
0	x	0
1	0	A
1	1	B

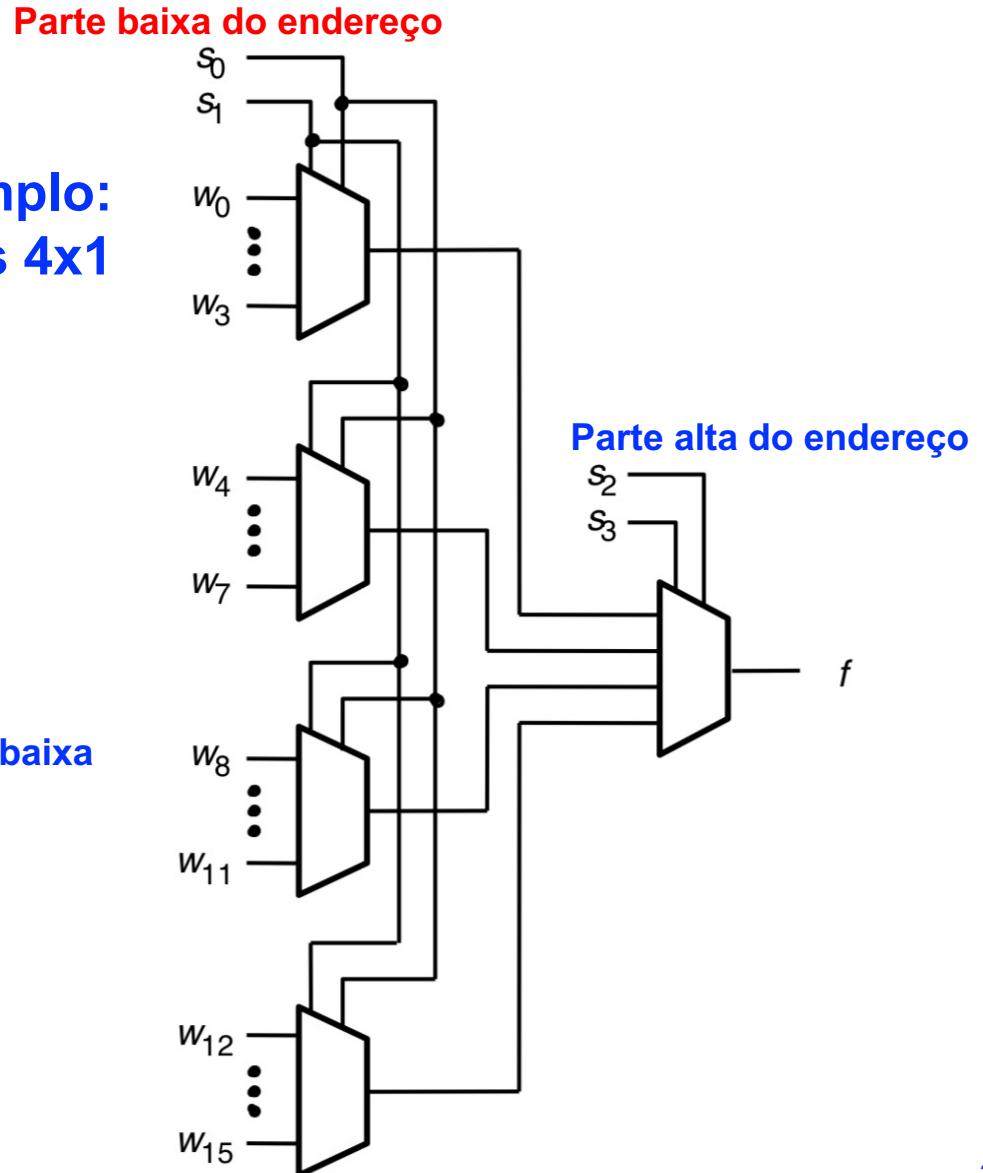
Construindo multiplexadores maiores a partir de multiplexadores menores

Exemplo:
Mux 16 x1 usando muxes 4x1

Solução: dividir o endereço em parte alta e baixa

endereço:
S3 S2 S1 S0

Parte alta Parte baixa



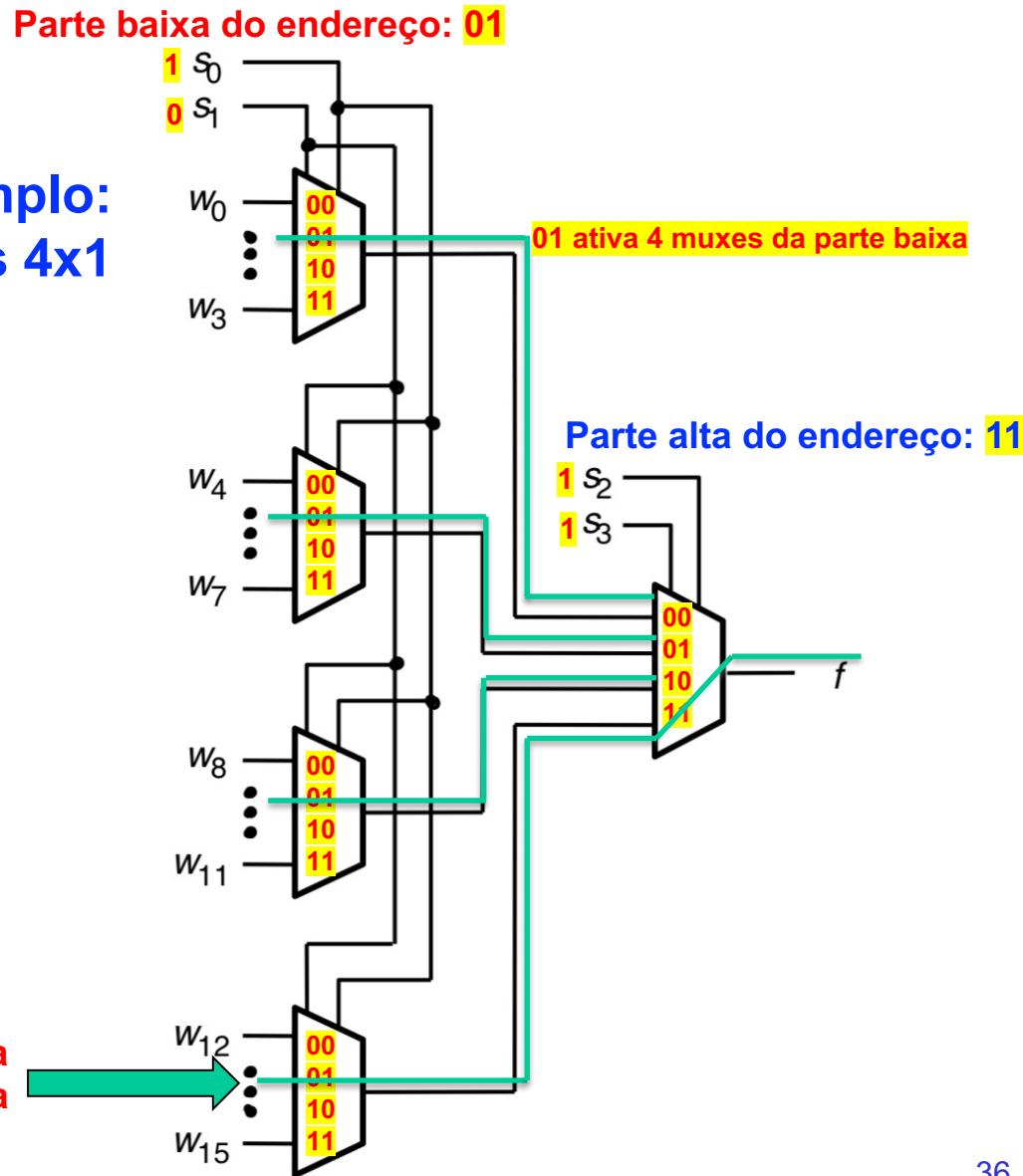
Construindo multiplexadores maiores a partir de multiplexadores menores

Exemplo:
Mux 16 x1 usando muxes 4x1

Supor endereço 1101 (13 em decimal)

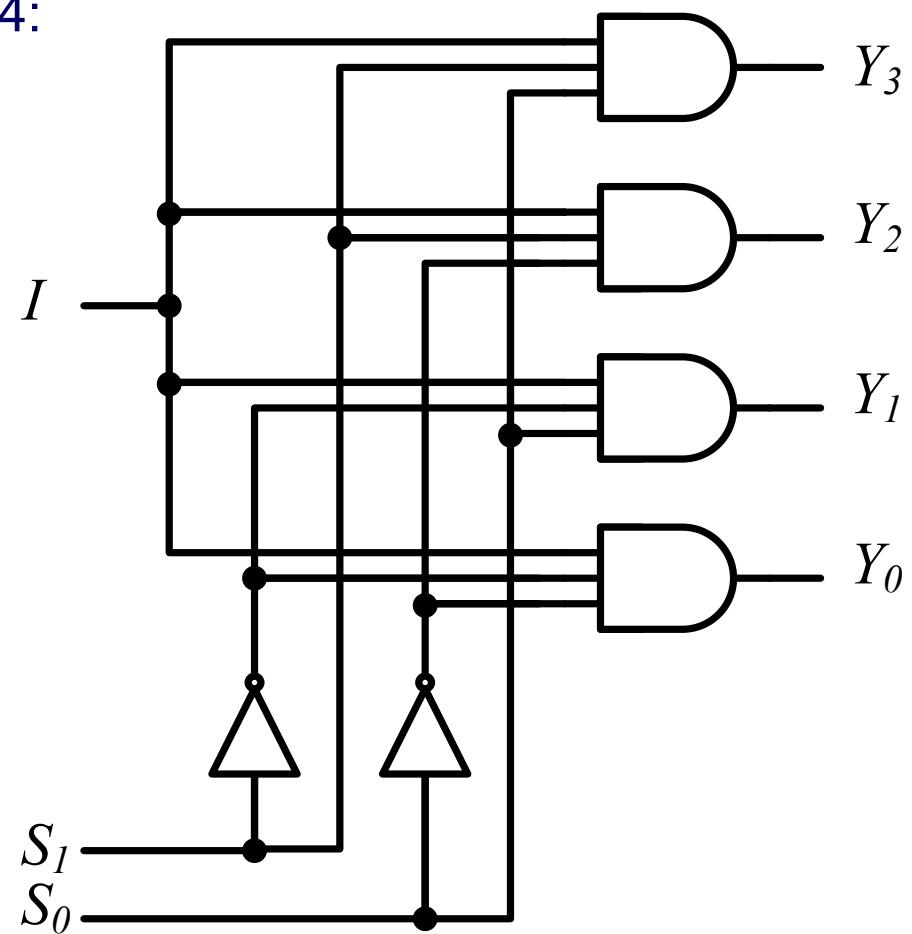
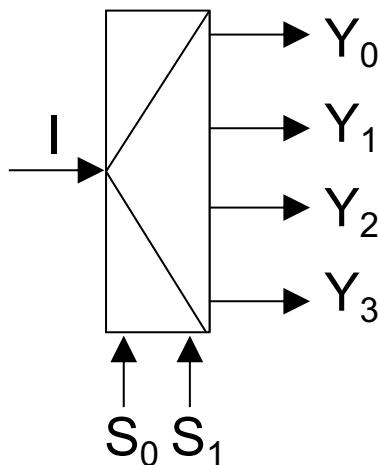
1 1 0 1
S3 S2 S1 S0

W13 é a entrada
que vai para a saída



Demultiplexador

- É um circuito que opera de forma inversa ao multiplexador. Ou seja, recebe uma entrada e distribui esta em uma de várias saídas conforme um sinal de seleção
- Exemplo de um multiplexador 1x4:



Exemplo de aplicação com Mux/Demux

- Multiplexação do meio físico para enviar diferentes sinais

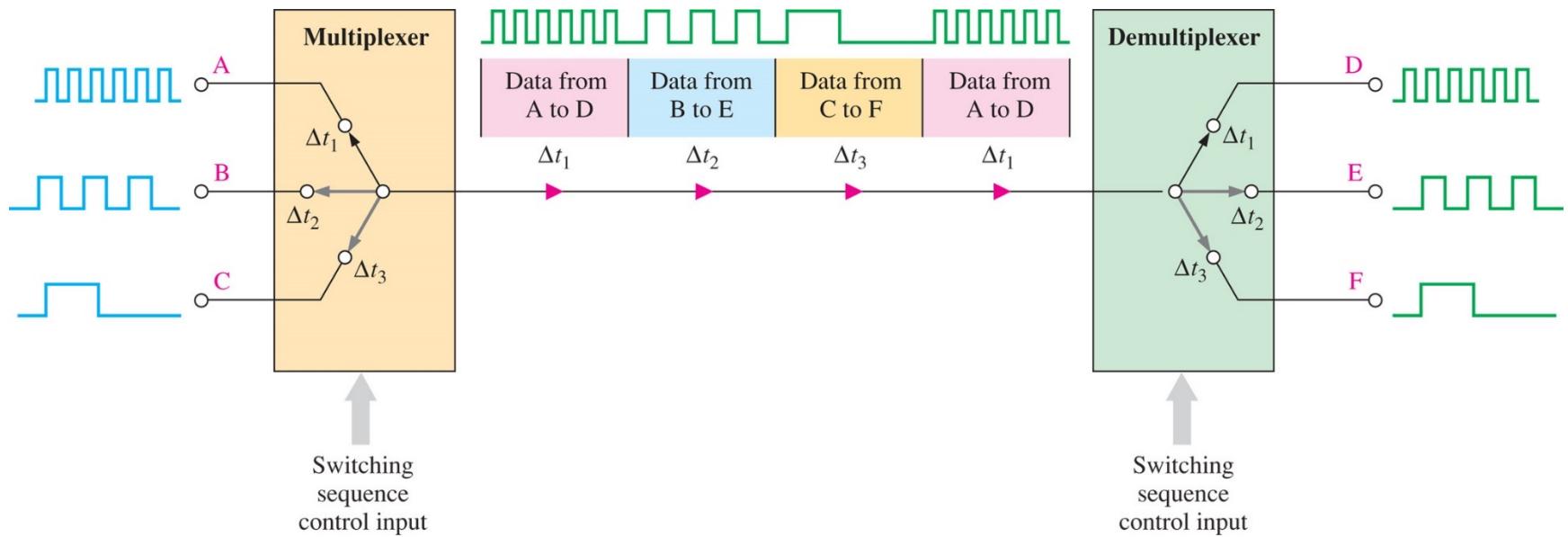


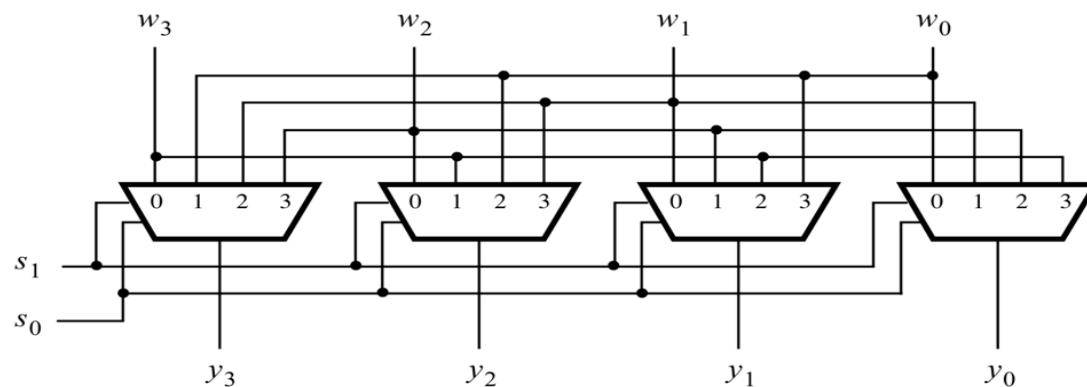
FIGURE 1-24 Illustration of a basic multiplexing/demultiplexing application.

Time-division multiple access (TDMA) is a channel access method for shared-medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots

(https://en.wikipedia.org/wiki/Time-division_multiple_access)

Multiplexadores

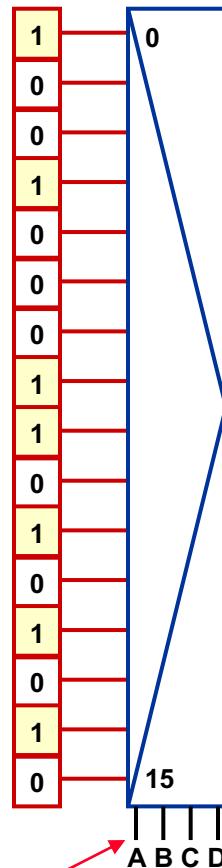
O circuito abaixo é composto por 4 multiplexadores. As entradas são: uma palavra de dados (w_3-w_0) e dois bits de controle (s_1-s_0). A saída é o vetor y (y_3-y_0). Preencha a tabela verdade correspondente a este circuito, e interprete o que este circuito realiza. No preenchimento da tabela verdade utilizar os valores $w_3/w_2/w_1/w_0$.



S1	S0	Y3	Y2	Y1	Y0
0	0				
0	1				
1	0				
1	1				

Multiplexador para gerar funções de n entradas (LUT)

A tabela verdade da função é armazenada em uma memória

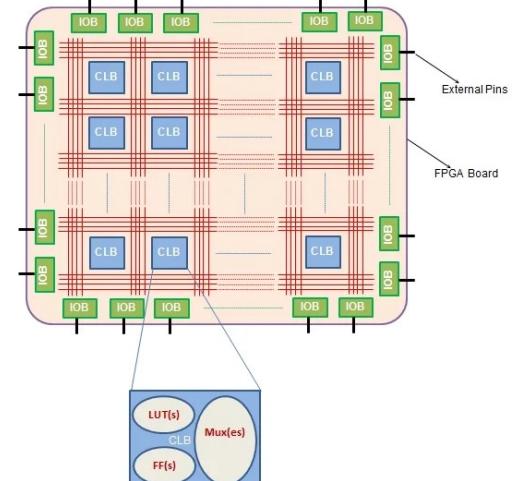


LUT: Look-Up Tables (tabelas da busca)

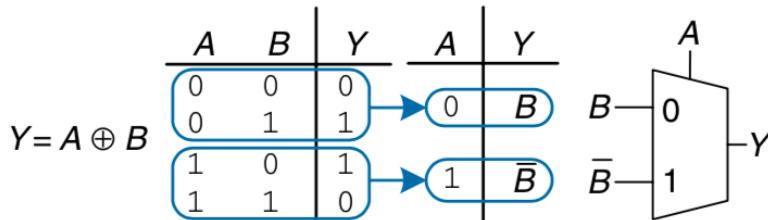
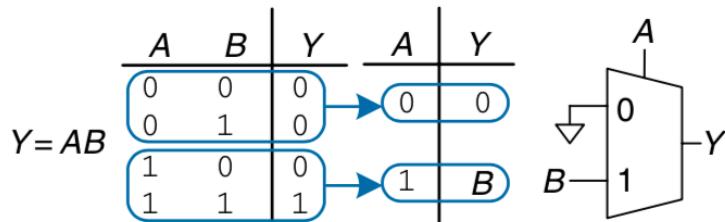
$$F(A, B, C, D) = \overline{A}\overline{B}CD + \overline{A}CD + A\overline{D}$$

$$F(A, B, C, D) = \sum(0, 3, 7, 8, 10, 12, 14)$$

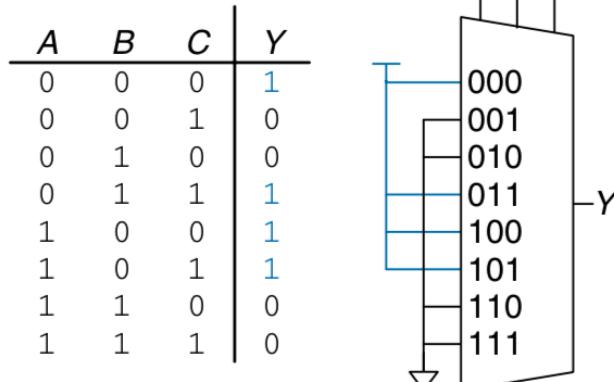
Onde LUTs são utilizadas? FPGAs



Multiplexador para gerar função booleanas



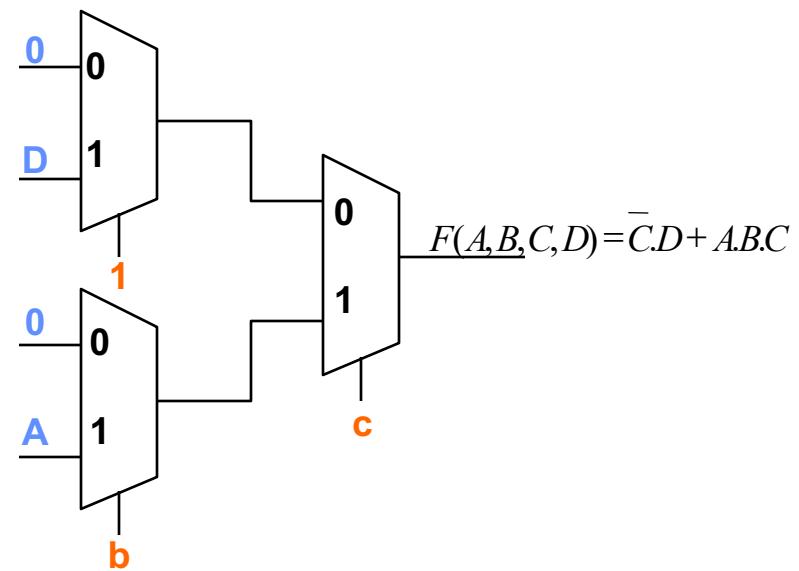
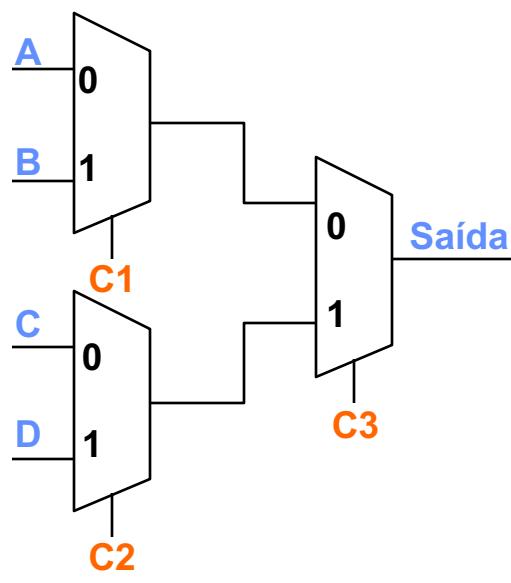
Na forma de LUT:



Digital Design and Computer Architecture
p. 86

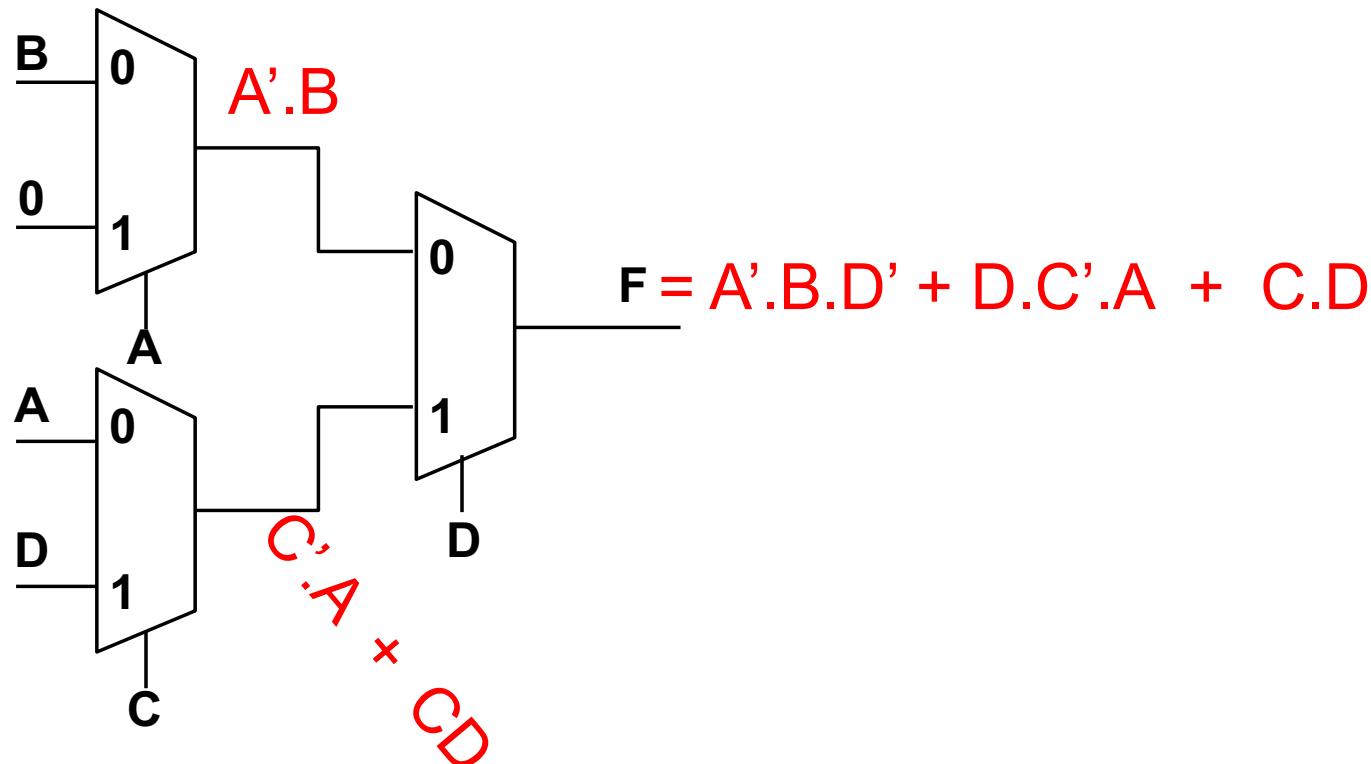
Multiplexador para gerar funções booleanas

- estrutura conhecida como “gerador universal de funções lógicas” - ULG
- não implementa todas as funções lógicas de n entradas
- funções lógicas mais complexas requerem diversos ULGs



EXERCÍCIO

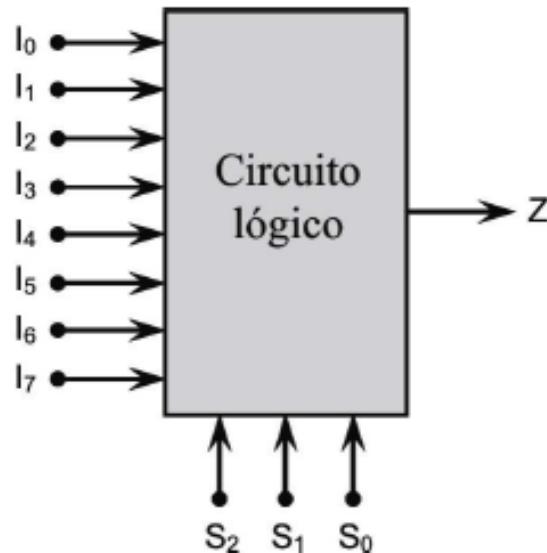
Os circuitos **multiplexadores** são também utilizados para a geração de funções booleanas. Considerando a conexão dos multiplexadores 2:1 abaixo, qual a função resultante no sinal F? Expressar a resposta na forma de soma de produtos.



EXERCÍCIO

(POSCOMP 2014, Questão 47) Analise o diagrama a seguir.

Observe o diagrama do circuito lógico e sua respectiva tabela verdade a seguir.



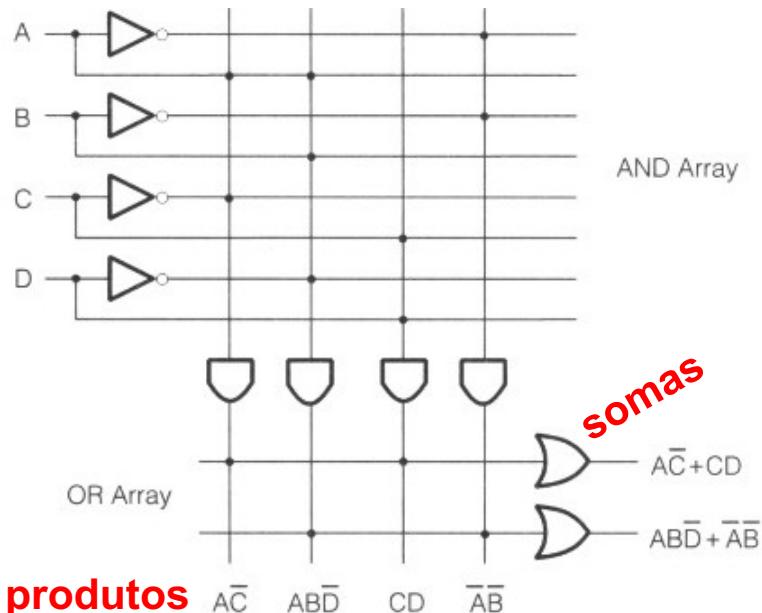
S_2	S_1	S_0	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Com base nesse diagrama e nessa tabela verdade, é correto afirmar que se trata de um circuito lógico

- a) codificador.
- b) comparador.
- c) decodificador.
- d) demultiplexador.
- e) multiplexador.

(5) PLA Matrizes Lógicas Programáveis

- Matrizes lógicas programáveis são tipos de circuitos que têm hardware pré-definido (parte estática) que implementa diversas funcionalidades conforme este for programado (parte dinâmica)
- Normalmente a programação é compreendida como uma camada de software de baixo nível programada em memórias do tipo RAM
- Muitas vezes, este tipo de circuito permite rapidamente criar novas funcionalidades de hardware, seja em tempo de projeto, seja em tempo de operação
- Exemplos de matrizes lógicas programáveis são **PLAs**, **PLDs**, e **FPGAs (usam LUT)**
- Exemplo de um PLA (contendo um plano E e outro OU)



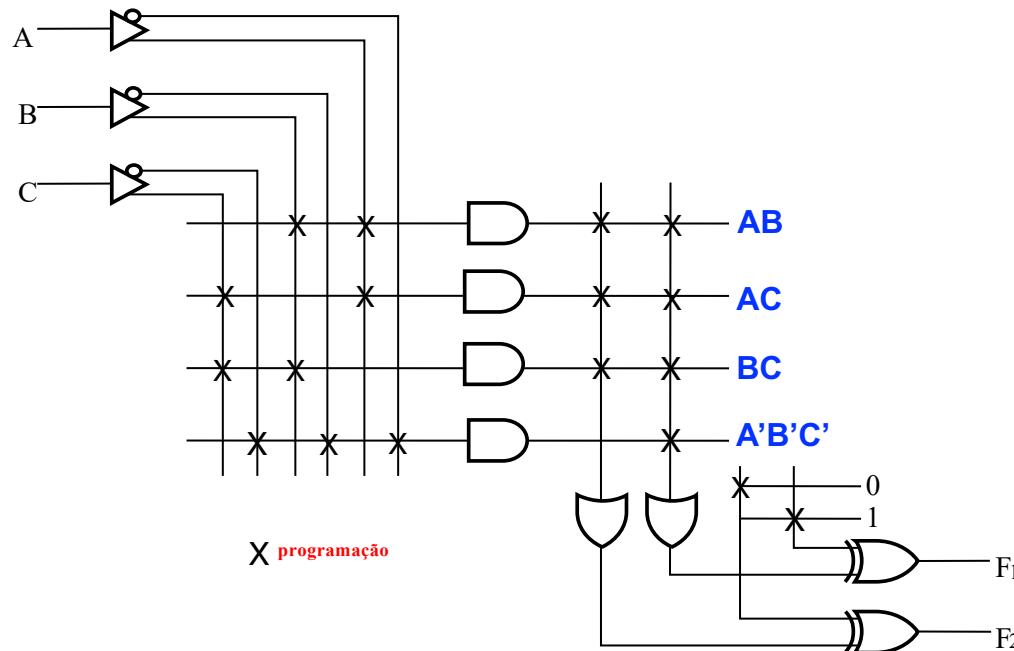
Programmable Logic Array (PLA)

- O conjunto de funções a serem implementadas é primeiro transformado em somas de produto
- Uma vez que a inversão de saída está disponível, as funções podem ser implementadas com o seu complemento

Exemplo:

$$F_1 = (AB + AC + BC + A'B'C')$$

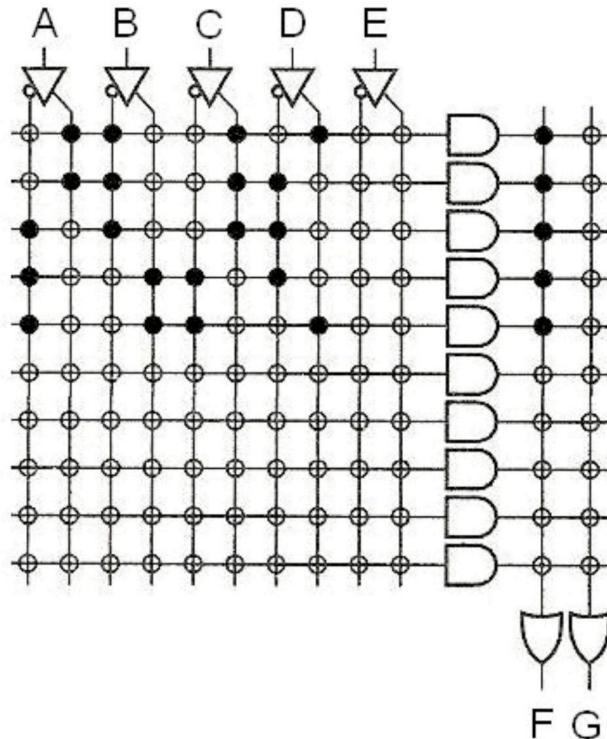
$$F_2 = AB + AC + BC$$



PLA – EXERCÍCIO (1/2)

(POSCOMP 2010 - 41) Considere o circuito digital apresentado no diagrama a seguir. Ressalte-se que, por convenção, chaves representadas por círculos escuros representam conexões fechadas e chaves representadas por círculos vazados representam conexões abertas.

Determine a função F.

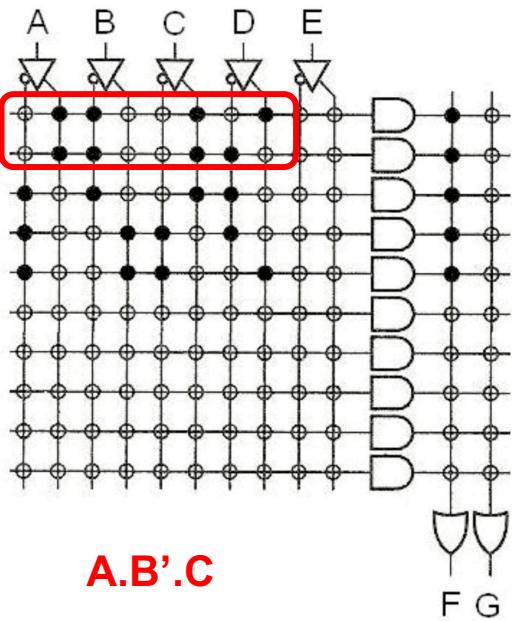


Gabarito com erro (b=c)

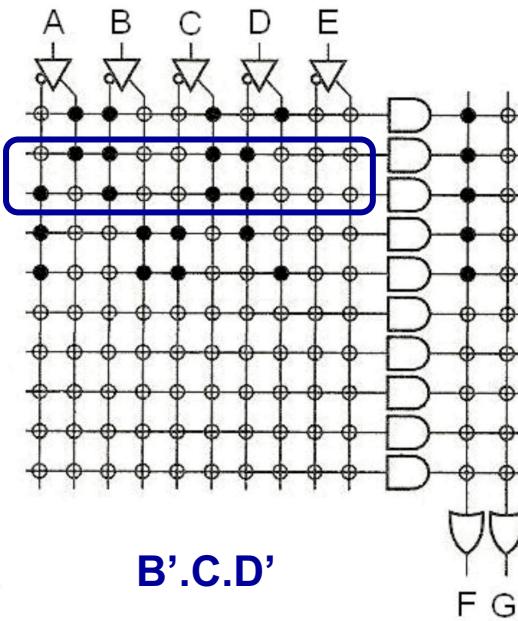
Assinale a alternativa correta.

- a) O circuito representa uma implementação em PAL da função $F = \bar{A}\bar{B}\bar{C} + B\bar{C}D + A\bar{B}C$.
- b) O circuito representa uma implementação em FPGA da função $F = \bar{A}\bar{B}\bar{C} + B\bar{C}D + A\bar{B}C$.
- c) **O circuito representa uma implementação em PLA da função $F = \bar{A}\bar{B}\bar{C} + B\bar{C}D + A\bar{B}C$.**
- d) O circuito representa uma implementação em PAL da função $G = \bar{A}\bar{B}\bar{C} + B\bar{C}D + A\bar{B}C$.
- e) O circuito representa uma implementação em PLA da função $G = \bar{A}\bar{B}\bar{C} + B\bar{C}D + A\bar{B}C$.

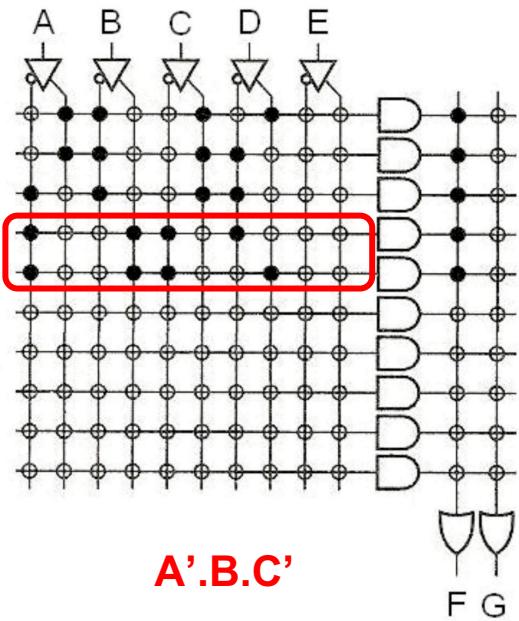
PLA – EXERCÍCIO (1 solução)



$$A \cdot B'$$



$$B' \cdot C \cdot D'$$



$$A' \cdot B \cdot C'$$

$$F = A' \cdot B \cdot C' + B' \cdot C \cdot D' + A \cdot B'$$

Assinale a alternativa correta.

- a) O circuito representa uma implementação em PAL da função $F = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}D + A\bar{B}C$.
- b) O circuito representa uma implementação em FPGA da função $F = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}D + A\bar{B}C$.
- c) O circuito representa uma implementação em PLA da função $F = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}D + A\bar{B}C$.**
- d) O circuito representa uma implementação em PAL da função $G = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}D + A\bar{B}C$.
- e) O circuito representa uma implementação em PLA da função $G = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}D + A\bar{B}C$.

Gabarito com erro (b=c)

PLA – EXERCÍCIO (2)

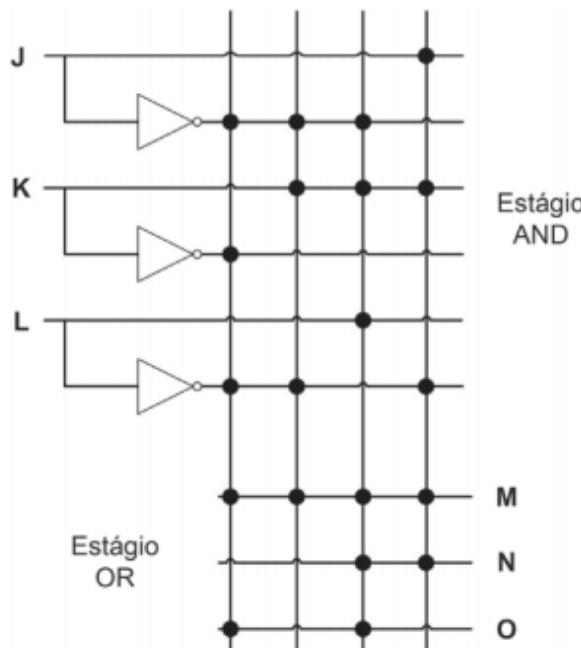
(ENADE 2014, Questão 23)

QUESTÃO 23

Um componente bastante usado em circuitos lógicos é a matriz lógica programável (ou PLA, do inglês *Programmable Logic Array*). Uma PLA usa como entrada um conjunto de sinais e os complementos desses sinais (que podem ser implementados por um conjunto de inversores). A lógica é implementada a partir de dois estágios: o primeiro é uma matriz de portas AND, que formam o conjunto de termos produto (também chamados *mintermos*); o segundo estágio é uma matriz de portas OR, cada uma efetuando uma soma lógica de qualquer quantidade dos mintermos. Cada um dos mintermos pode ser o resultado do produto lógico de qualquer dos sinais de entrada ou de seus complementos.

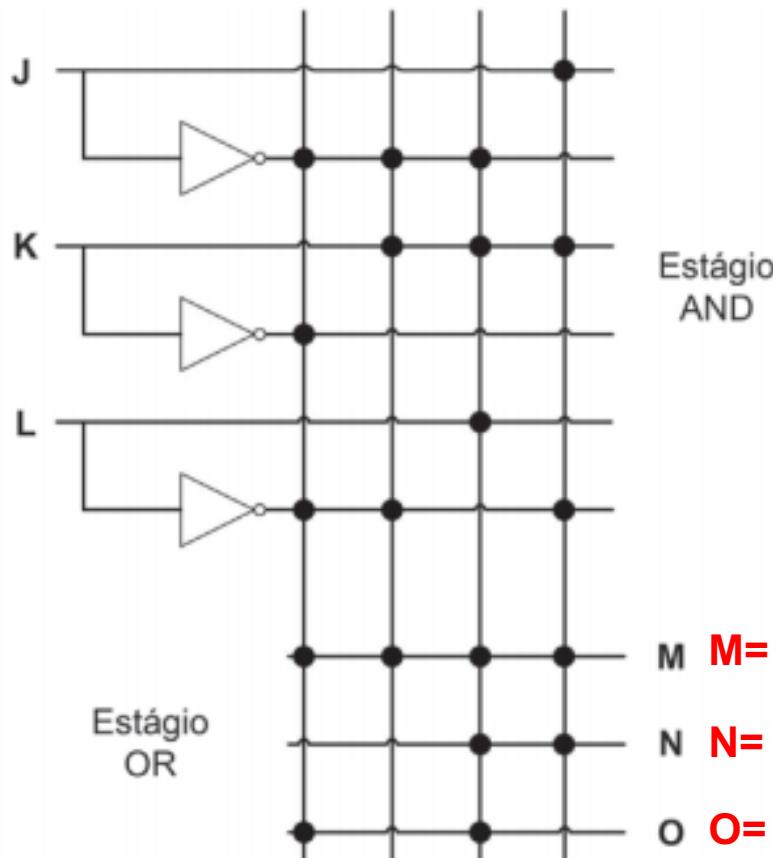
É comum, em lugar de desenhar todas as portas lógicas de cada um dos estágios, representar apenas a posição das portas lógicas em uma matriz, conforme ilustra a figura a seguir.

A partir da figura apresentada, infere-se que as entradas $JKL = 000$ e $JKL = 101$ levam a saídas MNO iguais, respectivamente, a



PLA – EXERCÍCIO (2 – solução)

(ENADE 2014, Questão 23)



$$M = J'.K'.L' + J'.K.L' + J'.K.L + J.K.L'$$

$$N = J'.K.L + J.K.L'$$

$$O = J'.K'.L' + J'.K.L$$

A partir da figura apresentada, infere-se que as entradas $JKL = 000$ e $JKL = 101$ levam a saídas MNO iguais, respectivamente, a

- A** 000 e 000. **B** 000 e 010. **C** 100 e 101. **D** 101 e 000. **E** 101 e 010.

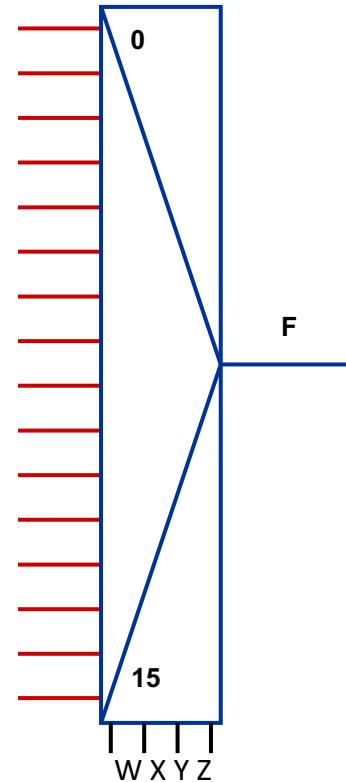
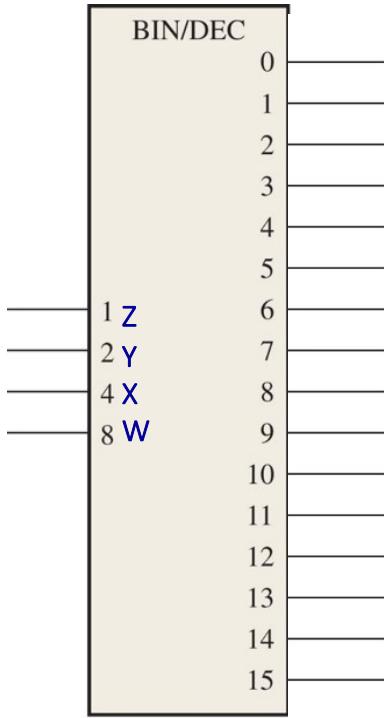
(6) Memória ROM

- Memória de apenas leitura – na prática um vetor de constantes
- Utilizada em hardware para constantes em determinado circuito (por exemplo: em código para boot em um processador)

Exercício (1/10)

Dada a seguinte expressão Booleana $F(w, x, y, z) = \sum(0, 1, 5, 7, 10, 14)$

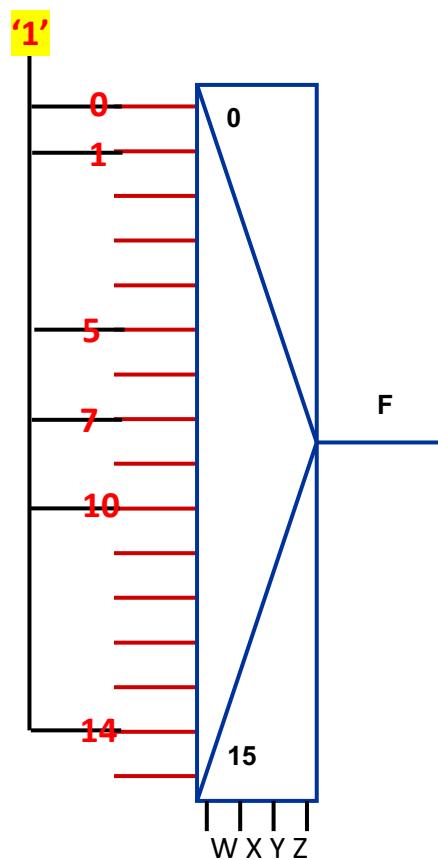
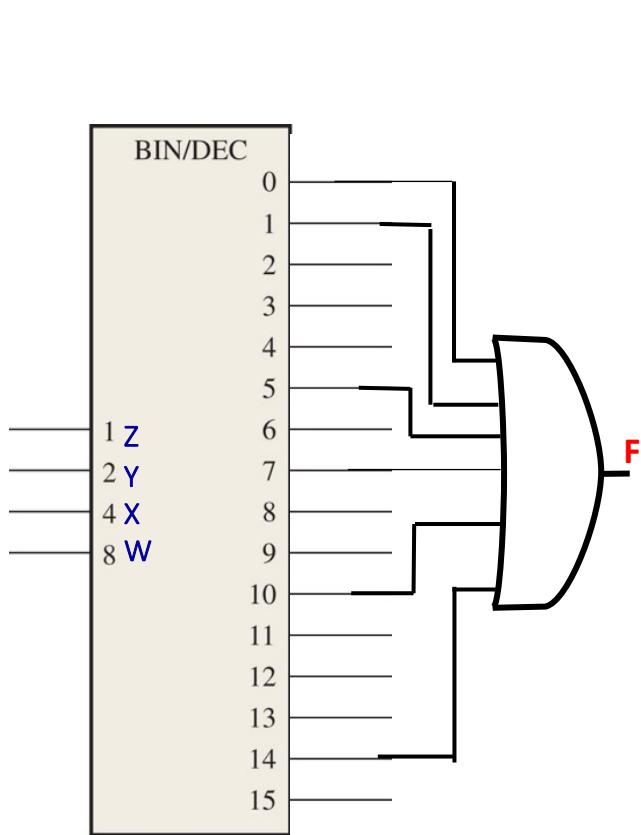
- Implemente esta expressão utilizando um decodificador 4x16;
- Implemente esta expressão utilizando um multiplexador $16 \rightarrow 1$.



Solução (1/10)

Dada a seguinte expressão Booleana $F(w, x, y, z) = \sum(0, 1, 5, 7, 10, 14)$

- Implemente esta expressão utilizando um decodificador 4x16;
- Implemente esta expressão utilizando um multiplexador 16→1.

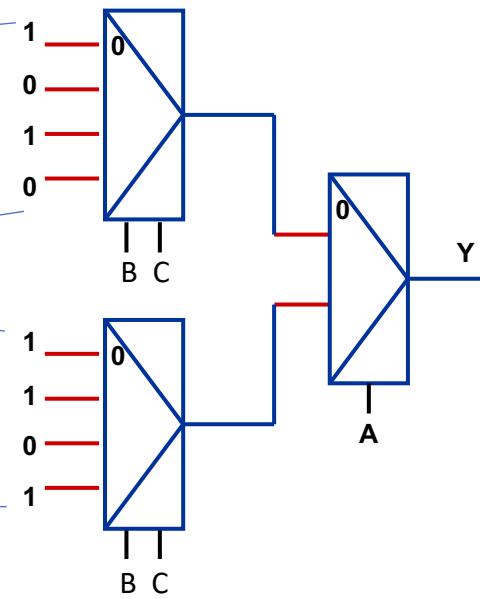


Demais entradas em zero

Exercício (2/10)

Implemente a seguinte tabela verdade
utilizando somente mux 4:1 e 2:1

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



Exercício (3/10)

Exercise 2.28 Write a minimized Boolean equation for the function performed by the circuit in Figure 2.85.

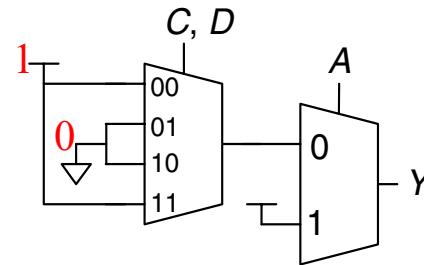


Figure 2.85 Multiplexer circuit

$$Y = A' \cdot (C'D' + C \cdot D) + A$$

Minimizando
 $Y = A + CD + C'D'$

$A \setminus CD$	00	01	11	10
0	1		1	
1	1	1	1	1

Exercício (4/10)

Realize um circuito que compare dois números inteiros positivos de 2 bits (faixa de representação de 0 a 3), e indique em sua saída se $A > B$. Apresente o circuito resultante por:

- a) Expressão booleana minimizada
- b) Diagrama de portas lógicas

Entradas do circuito: A_1, A_0, B_1, B_0

Saída do circuito: Maior

Dica: utilizar Mapa de Karnaugh.

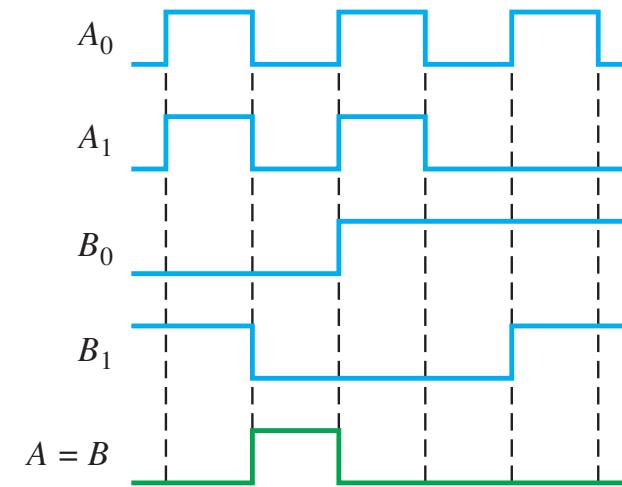
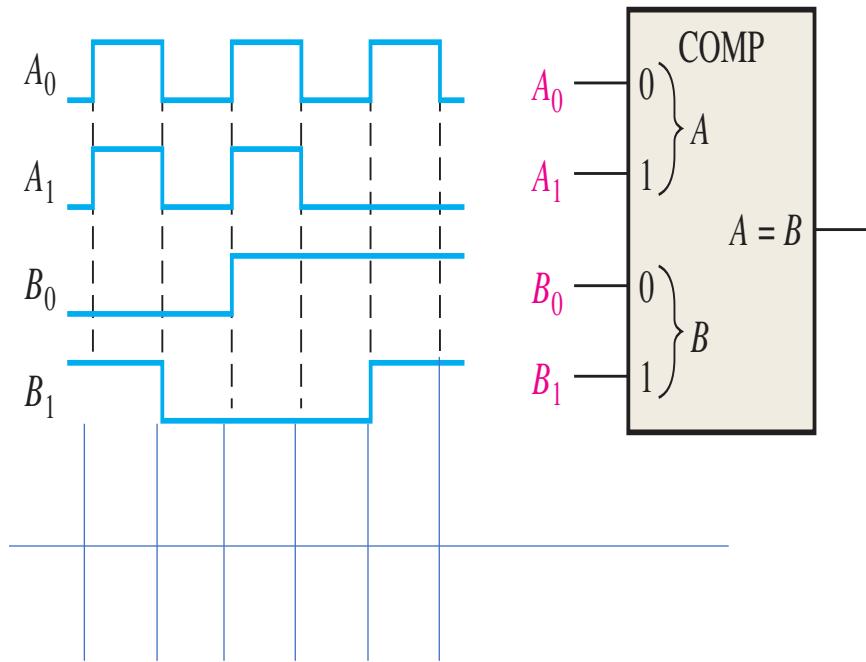
$A_1 \ A_0$ \ $B_1 \ B_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$M = A_1 \cdot B_1' + A_0 \cdot B_1' \cdot B_0' + A_1 \cdot A_0 \cdot B_0'$$

Exercício (5/10)

Section 6-4 Comparators

13. The waveforms in Figure 6-73 are applied to the comparator as shown. Determine the output ($A = B$) waveform.



Exercício (6/10)

14. For the 4-bit comparator in Figure 6–74, plot each output waveform for the inputs shown. The outputs are active-HIGH.

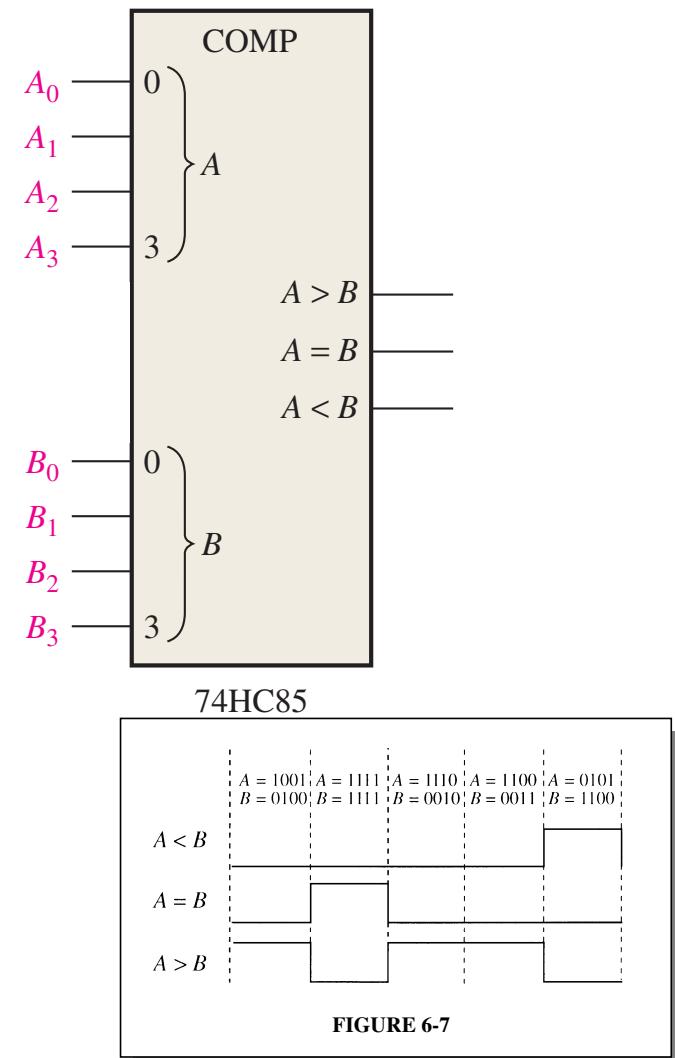
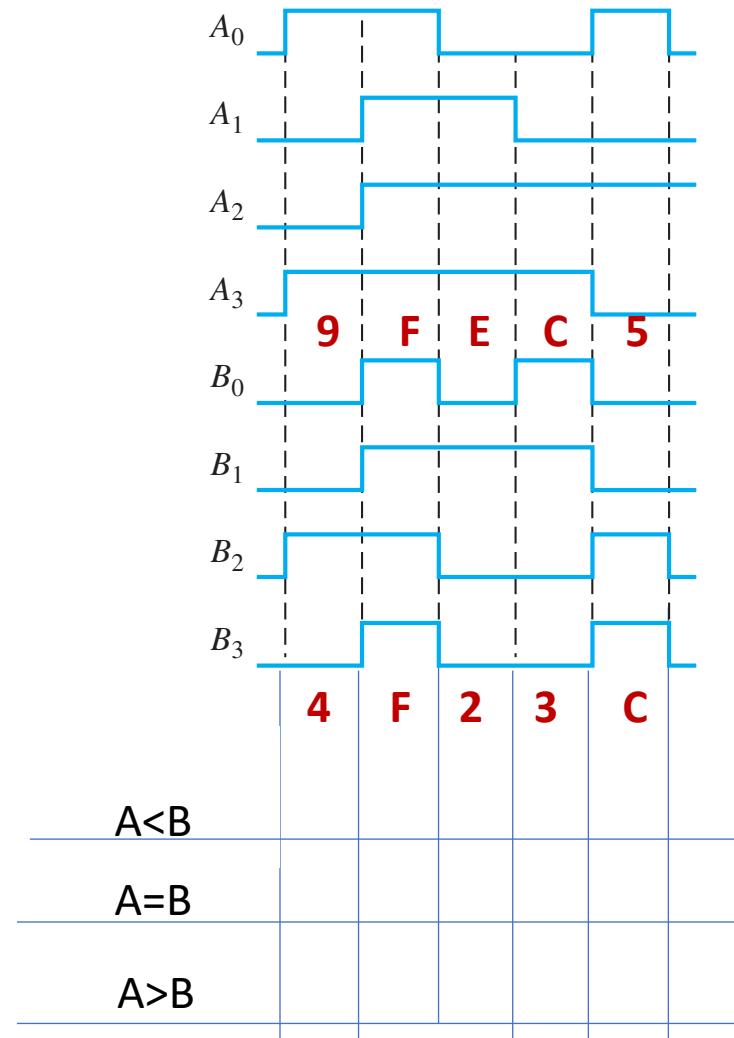
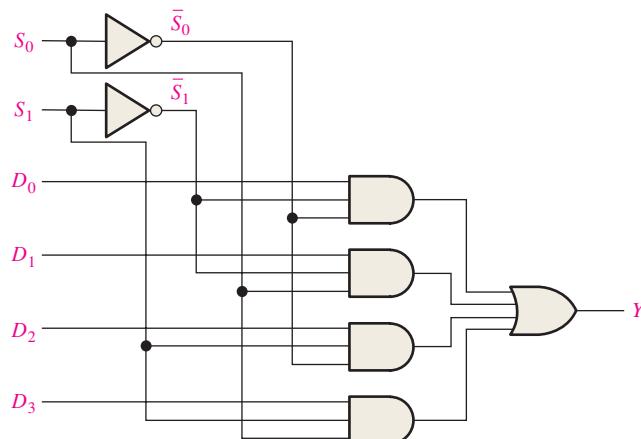


FIGURE 6-7

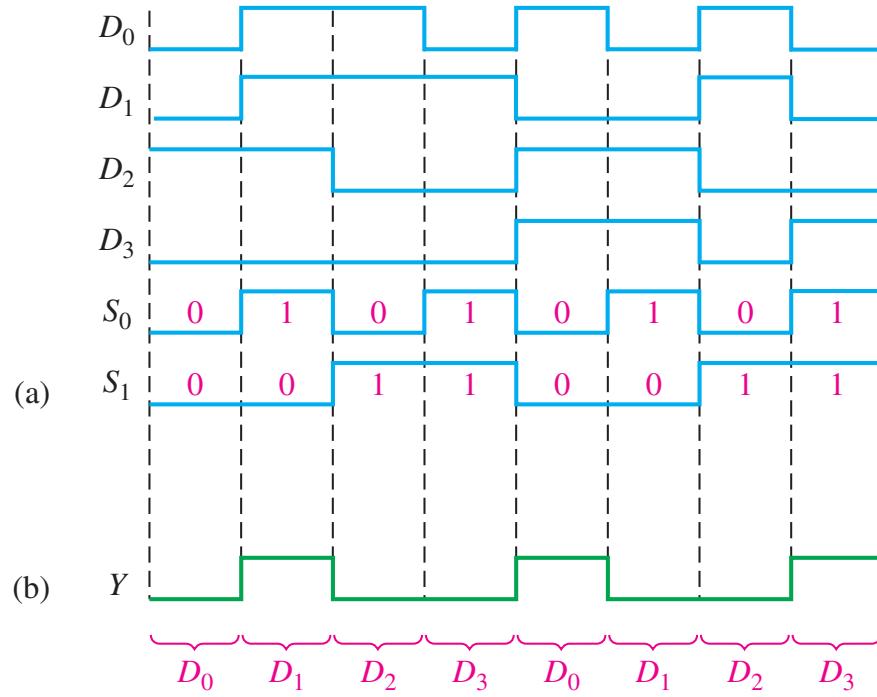
Exercício (7/10)

EXAMPLE 6-14

Mux 4-1



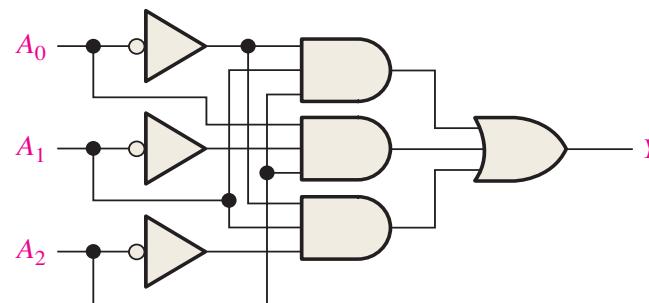
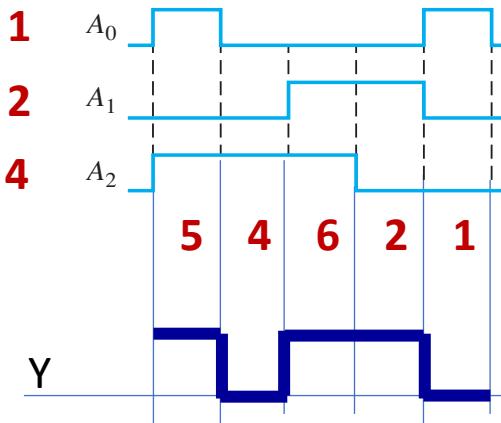
Determine a saída Y



Exercício (8/10)

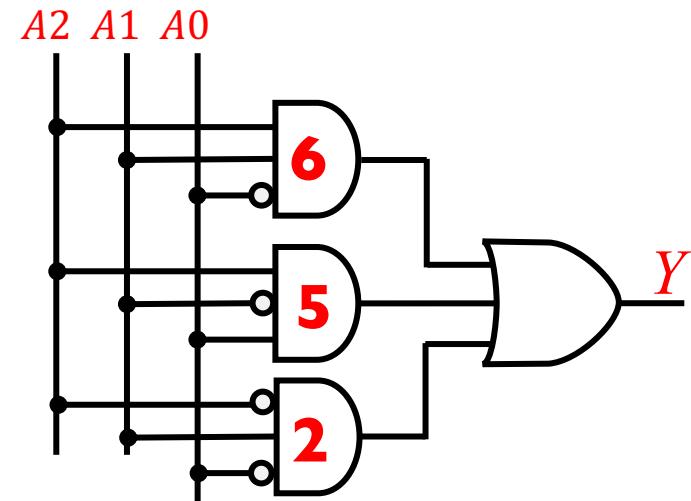
20. If the input waveforms are applied to the decoding logic as indicated in Figure 6–76, sketch the output waveform in proper relation to the inputs.

Determine a função Y e preencha a saída esperado em Y



$$Y = A_2 \cdot A_1 \cdot A_0' + A_2 \cdot A_1' \cdot A_0 + A_2' \cdot A_1 \cdot A_0'$$

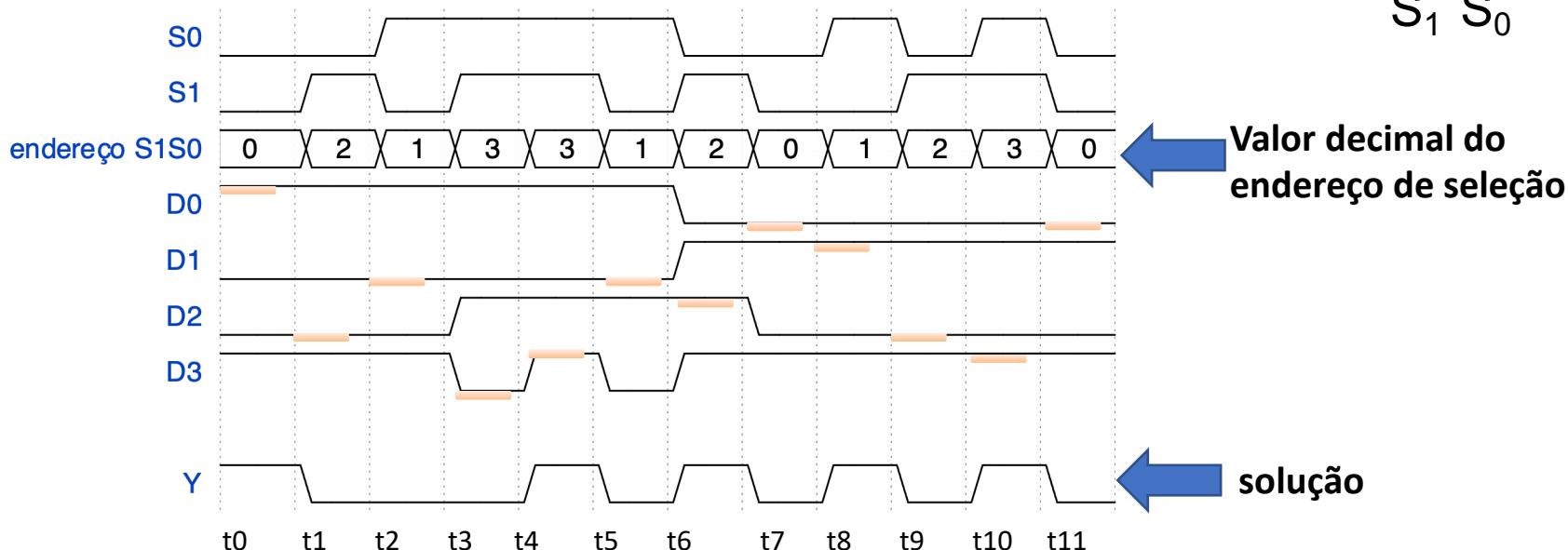
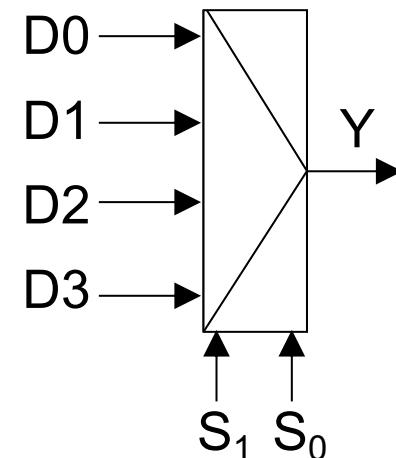
$$Y = \sum (6, 5, 2)$$



Exercício (9/10)

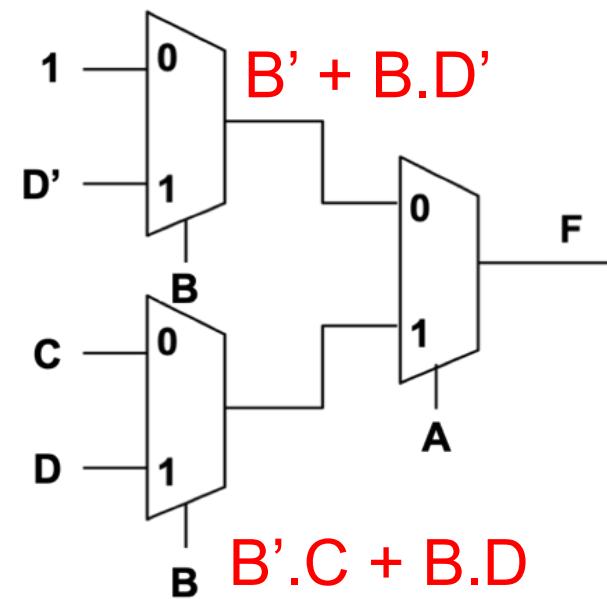
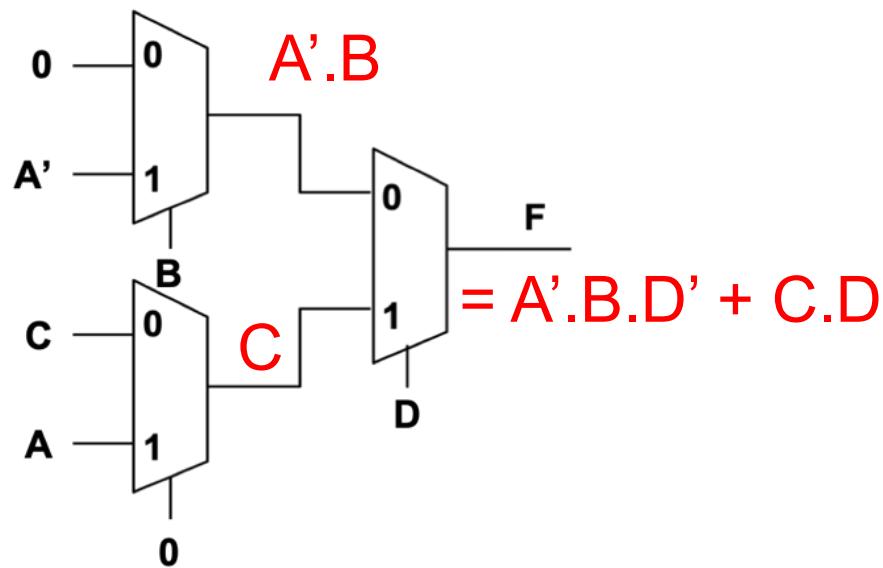
29. If the data-select inputs to the multiplexer in Figure 6-79 are sequenced as shown by the waveforms in Figure 6-80, determine the output waveform with the data inputs specified in Problem 28.

Para o multiplexador 4-1 ao lado, determine a saída Y em função do endereço de seleção (S_1S_0) e as entradas de dados D3 a D0



Exercício (10/10)

Considerando a conexão dos multiplexadores 2:1 mostrados abaixo. Quais as funções resultantes dos sinais F? Expressar a resposta na forma de soma de produtos.



$$= A'.B' + A'.B.D' + A.B'C + A.B.D$$

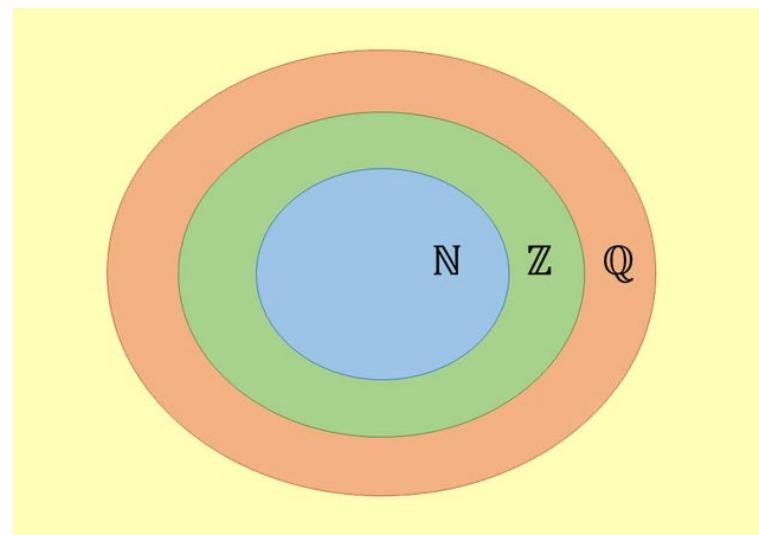
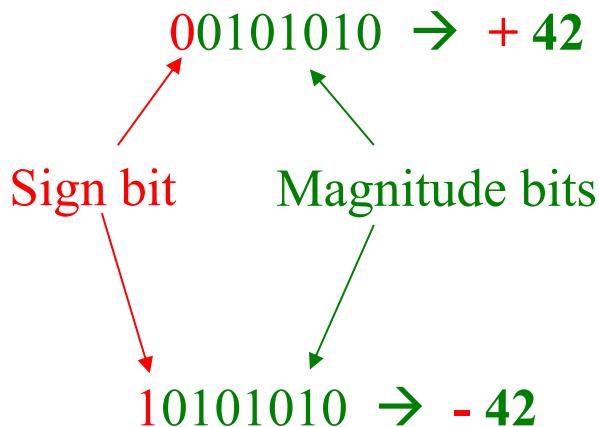
$$= A'.B' + A'.D' + B'C + A.B.D$$

AB/CD	00	01	11	10
00	1	1	1	1
01	1			1
11		1	1	
10			1	1

Sinal Magnitude (representa inteiros)

SM – sinal magnitude

→ o bit mais significativo é o sinal



NATURAIS: $N = \{0, 1, 2, 3, 4, 5..., n, ...\}$

INTEIROS: $Z = \{..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...\}$ ($N \subset Z$)

RACIONAIS: $Q \rightarrow$ todos os números que podem ser escritos na forma p/q , sendo p e q números inteiros e $q \neq 0$.

Complemento de 2 - 2's (representa inteiros)

Os computadores usam complemento de 2 para números inteiros

→ Bit mais significativo corresponde à $-(2^{n-1})$

Exemplo para 5 bits

$-(2^4)$ 2^3 2^2 2^1 2^0

-16 8 4 2 1

0 1 1 0 1 = 13

1 0 1 0 1 = $-16 + 5 = -11$

**Não
confundir
SM com 2's**

Complemento de 2 - 2's

Outra forma de obter o complemento de 2:

Utilizando o complemento de 1 + '1'

Complemento de 1 é o número invertido

Exemplo:

Determinar o valor em complemento de 2 de -11

Valor binário de 11: 0 1 0 1 1

Complemento de 1: 1 0 1 0 0

Soma 1:

1

1 0 1 0 1 (2's complement)

Este método é importante no momento de implementar a subtração

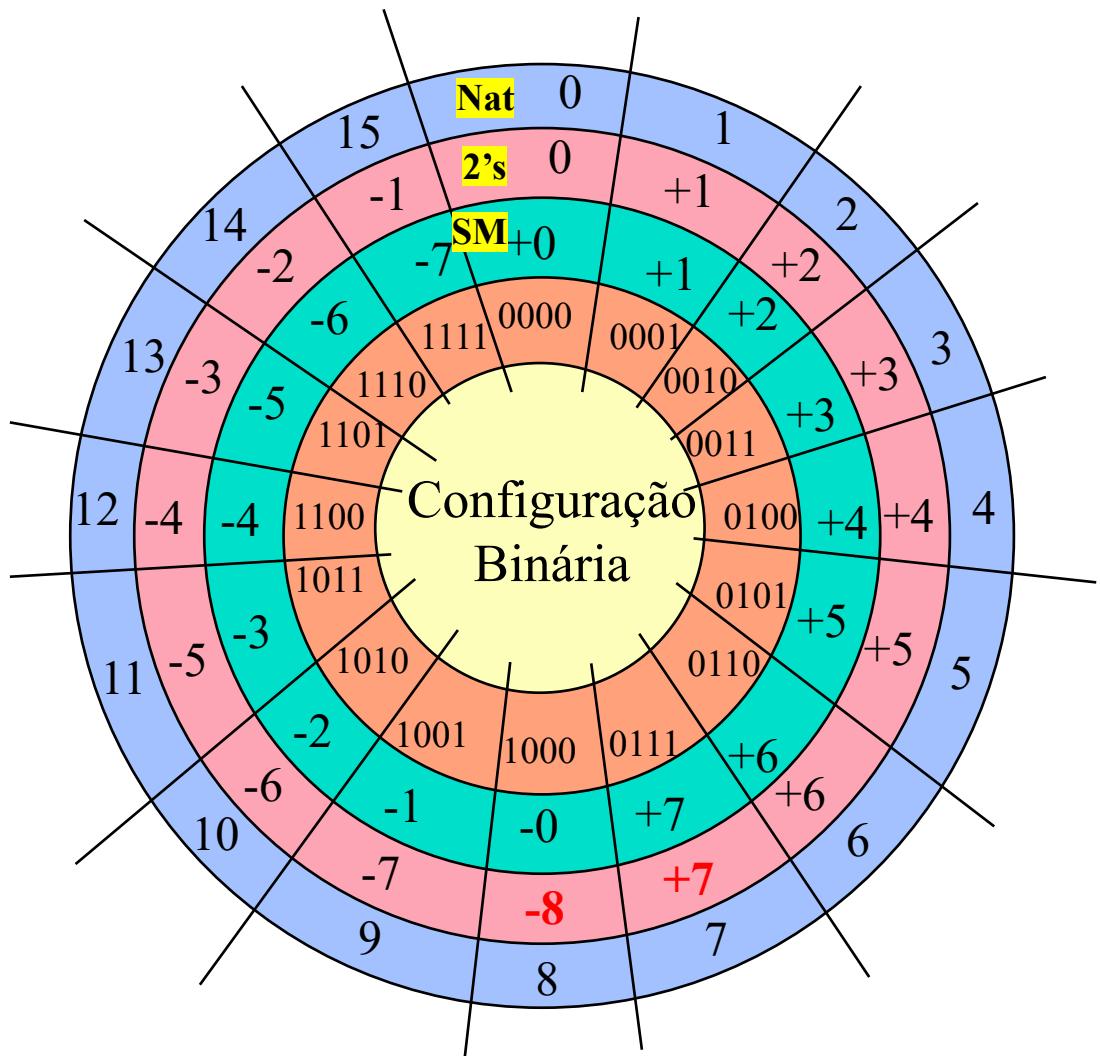
RESUMINDO

Um número binário de n bits poder ser interpretado de várias formas:

$$2^{n-1} \ 2^{n-2} + 2^{n-3} + \dots + 2^0$$

- Unsigned. (naturais - N)
- Sinal magnitude (SM): 2^{n-1} é o sinal (+/-) (inteiros - Z)
- Complemento de 2 (2's): $-(2^{n-1})$. (inteiros - Z)

	Unsigned (N)	SM (Z)	2's (Z)
0 1 0 1 0 0	20	20	20
1 1 0 1 0 0	52	-20	-12
1 1 1 1 1 1	63	-31	-1

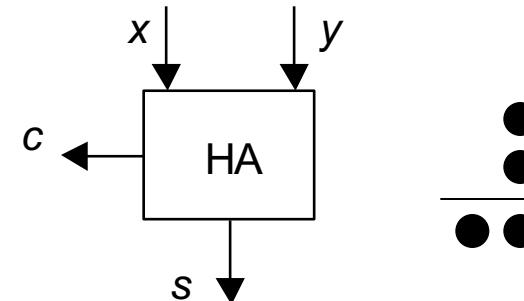
Representações de
Inteiros - 3 formas


- SM é fácil de entender e separa sinal de valor
 $[-(2^{n-1}-1) \dots 2^{n-1}-1]$
notar que o zero possui duas representações (+0 e -0)
- 2's facilita soma
 $[-(2^{n-1}) \dots 2^{n-1}-1]$
- Naturais (*unsigned*)
 $[0 \dots 2^n-1]$

(7) Soma e subtração

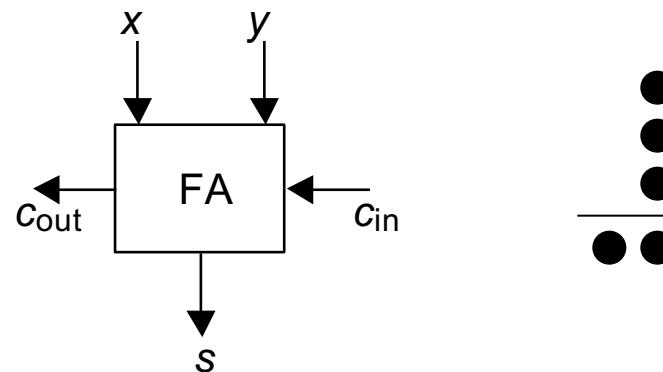
Somadores
de 1 bit

Inputs		Outputs	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Half-adder (HA): Truth table and block diagram

Inputs			Outputs	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full-adder (FA): Truth table and block diagram

Half-adder

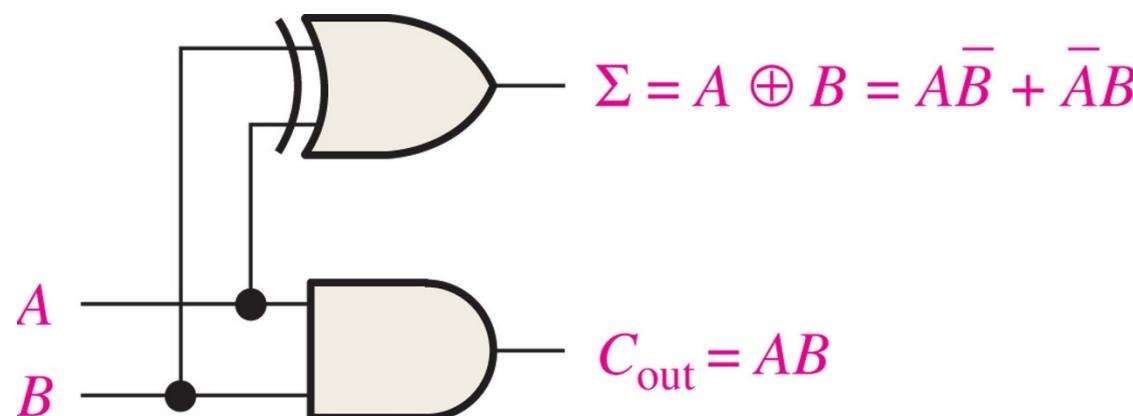
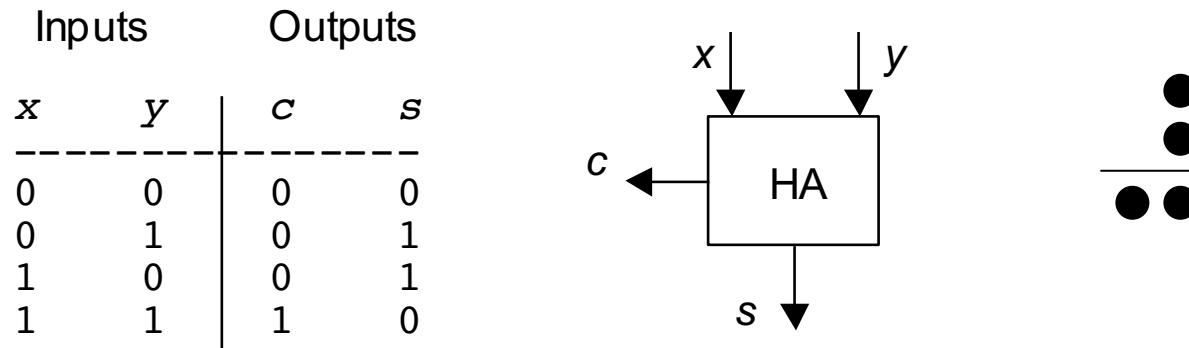
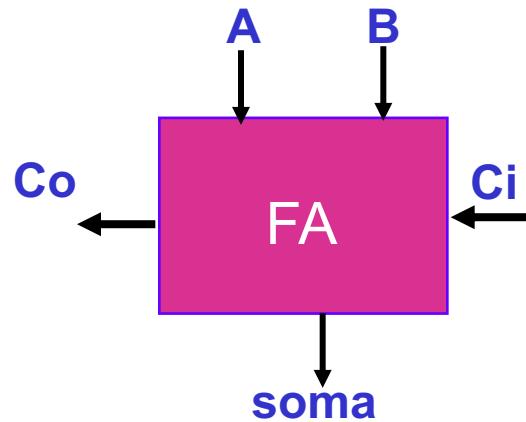


FIGURE 6-2 Half-adder logic diagram.

Somador Completo - FA



A	B	C _i	S	C _o
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\c	00	01	11	10
0		1		1
1	1		1	

Co:

a\b\c	00	01	11	10
0			1	
1		1	1	1

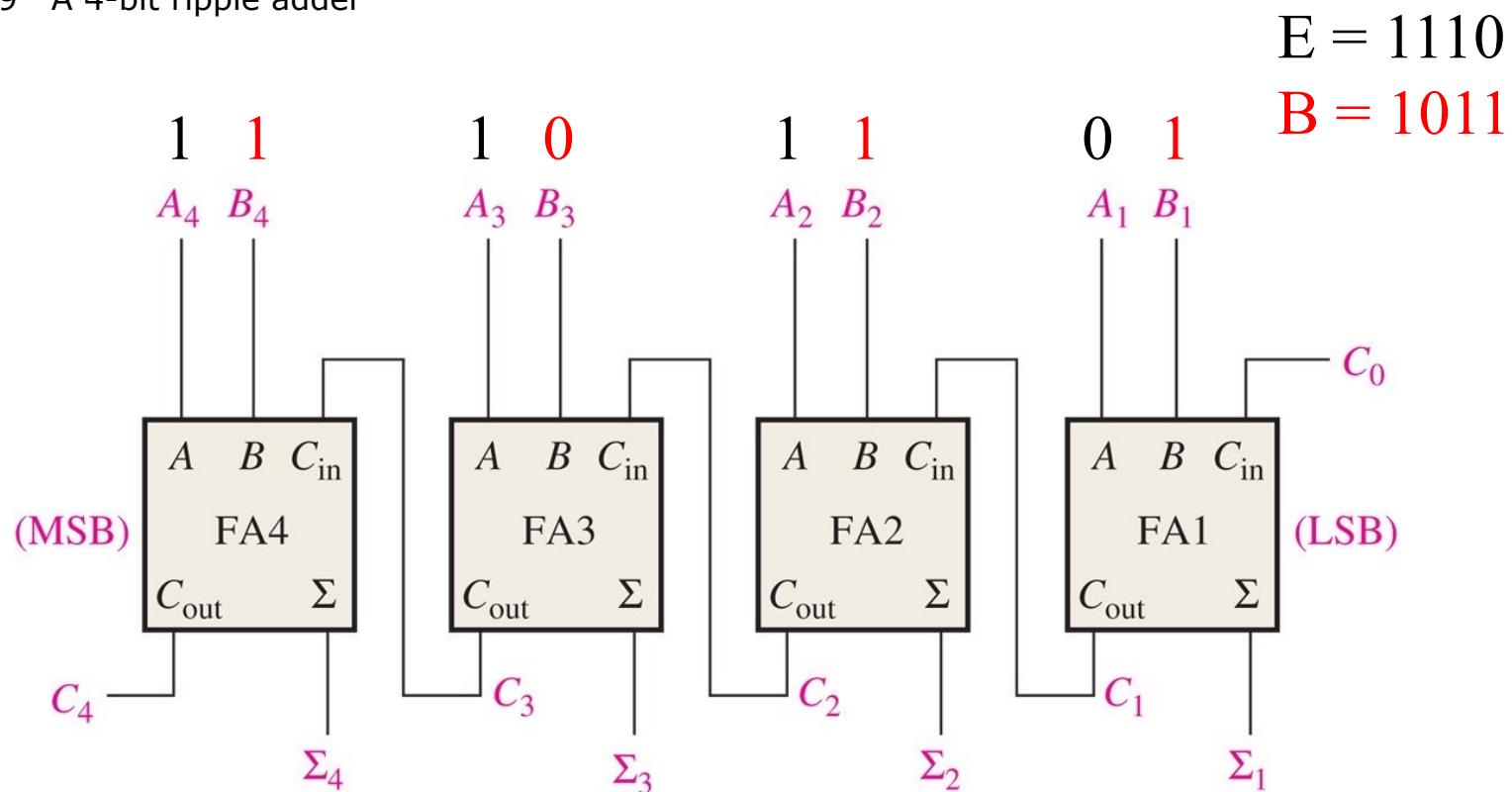
$$S = A \cdot \bar{B} \cdot \bar{C}_i + \bar{A} \cdot B \cdot \bar{C}_i + \bar{A} \cdot \bar{B} \cdot C_i + A \cdot B \cdot C_i$$

$$S = A \oplus B \oplus C_i$$

$$C_o = A \cdot B + B \cdot C_i + A \cdot C_i$$

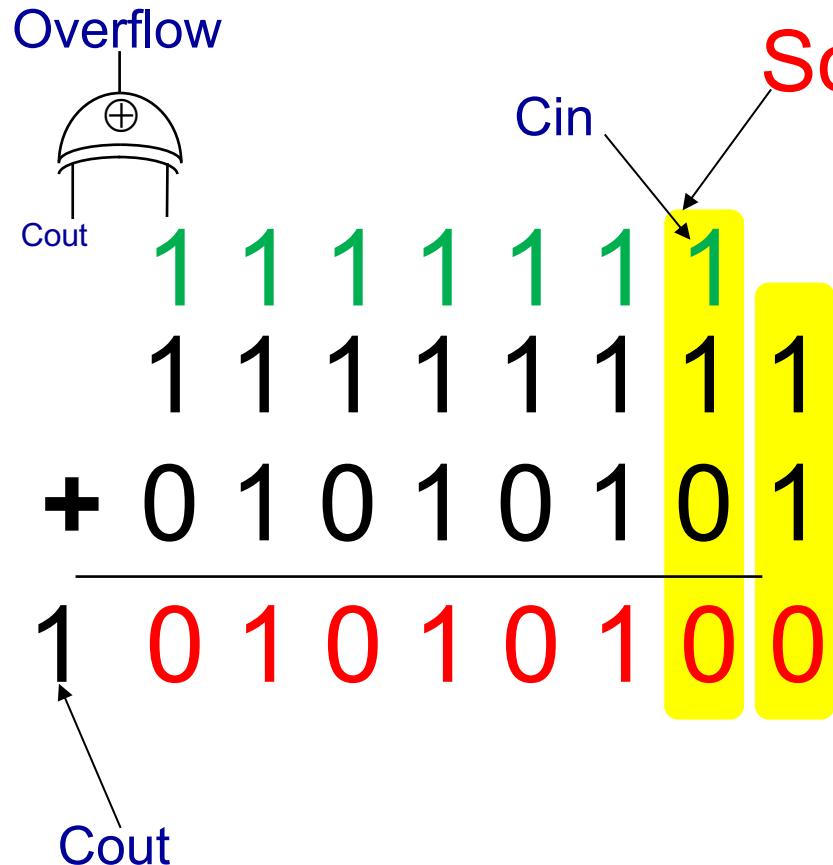
Somador "ripple"

FIGURE 6-9 A 4-bit ripple adder



Sumar $[E_{16} + B_{16}]$ escrevendo os valores de saída dos FAs

Somador "ripple"



Somador completo (FA)

Meio-somador (HA)

Hexadecimal: FF + 55 = (1) 54

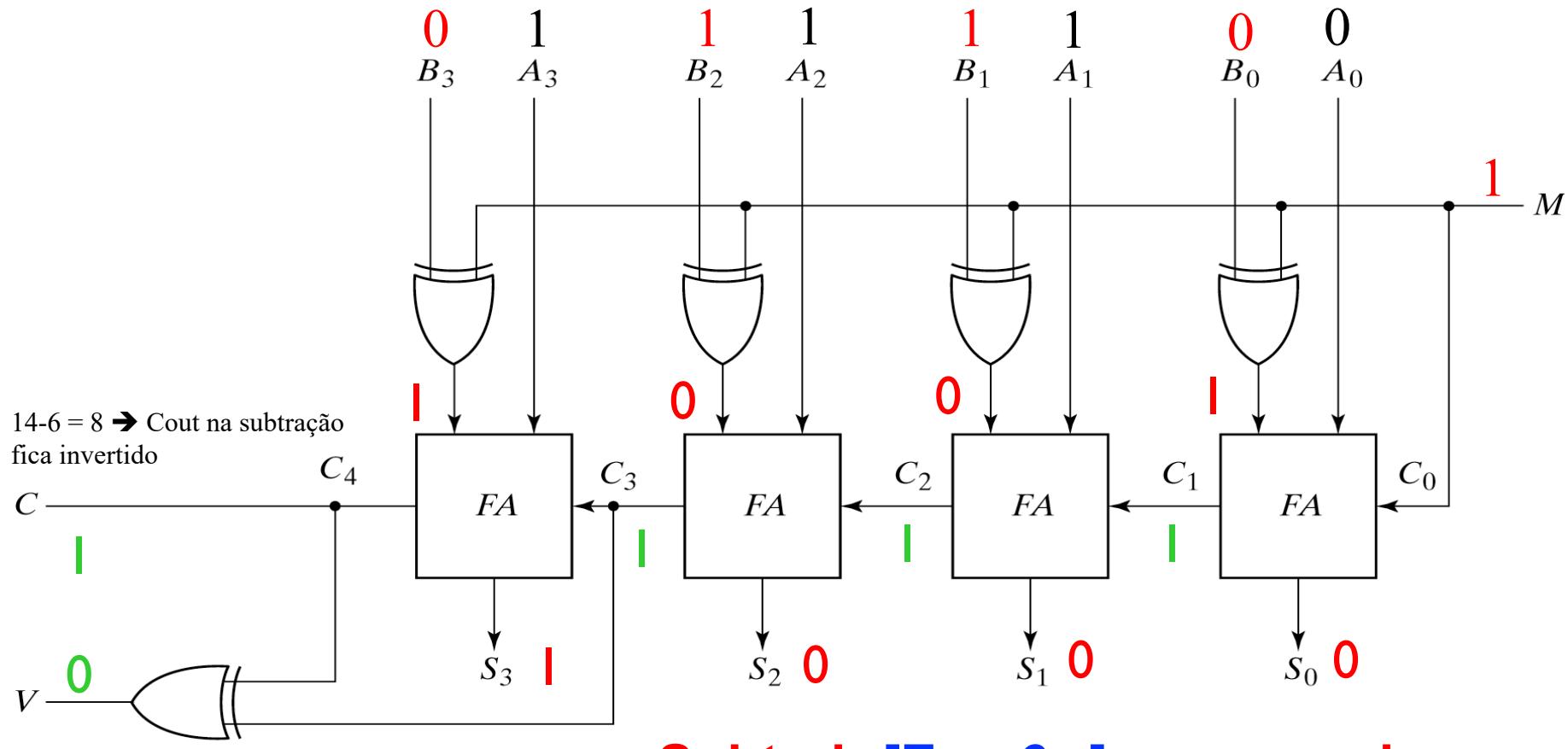
Unsigned: 255 + 85 = 84 (**errado**, Cout=1)

2's comp: -1 + 85 = 84 (**certo** – Overflow=0)

Somador / subtrator

- $M = 1 \rightarrow$ subtrator
 - $M = 0 \rightarrow$ somador

$$\begin{array}{l} E = 1110 \\ 6 = 0110 \end{array}$$

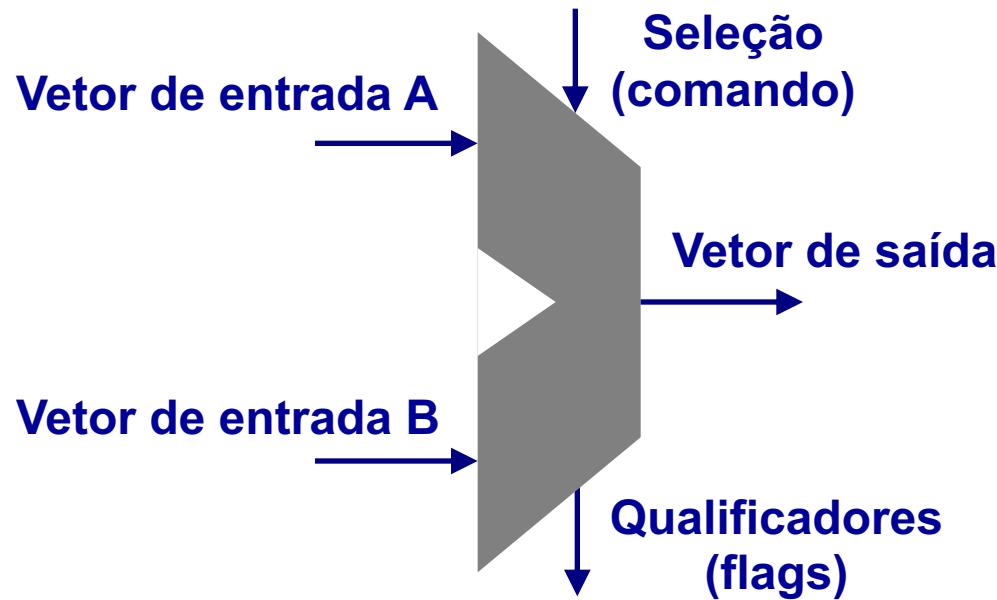


$-2-6 = -8 \rightarrow$ sem overflow

Subtrair $[E_{16} - 6_{16}]$ escrevendo os valores de saída dos FAs

(8) ULA - UNIDADE LÓGICO E ARITMÉTICA

- Unidade Lógica e Aritmética (ULA) é um circuito que realiza funções lógicas e aritméticas
- É um dos componentes de transformação de dados principais de um processador
- Normalmente implementado de forma combinacional
- Representação:



ULA – Funções Lógicas

- Diversas são as funcionalidades **lógicas**. Dentre as mais comuns estão:
 - E lógico das entradas
 - Ou lógico das entradas
 - Ou exclusivo lógico das entradas
 - Not - Negação de uma dada entrada
- A seleção de qual operação será realizada é obtida pela porta de comando
 - Normalmente controlada pela unidade de controle do processador onde se encontra a ULA
- Operações **lógicas** usam normalmente apenas os qualificadores Z (zero) e N (negativo)
 - Qualificadores de V (overflow) e C (carry) não são considerados, pois operações lógicas não alteram o valor dos mesmos

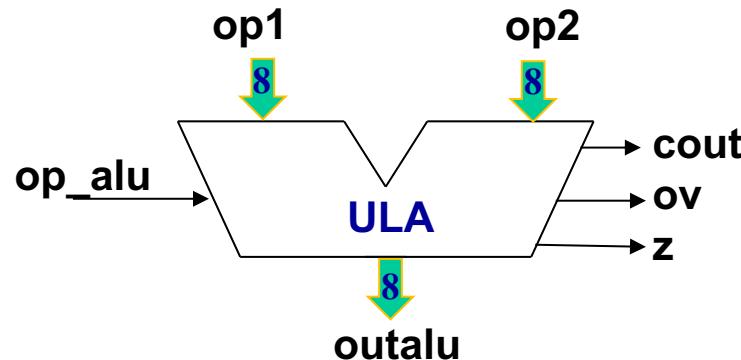
ULA – Funções Aritméticas

- **Dentre as funcionalidades aritméticas mais comuns estão:**
 - Soma das entradas
 - Subtração das entradas
 - Deslocamento de uma das entrada
 - Rotação de uma das entradas
 - Complemento de dois de uma das entradas
 - E variações das funcionalidades acima utilizando a flag C/OV
- **A seleção de qual operação será realizada é obtida pela porta de comando**
- **Operações aritméticas fazem uso dos quatro qualificadores: (Z, N, V, C)**

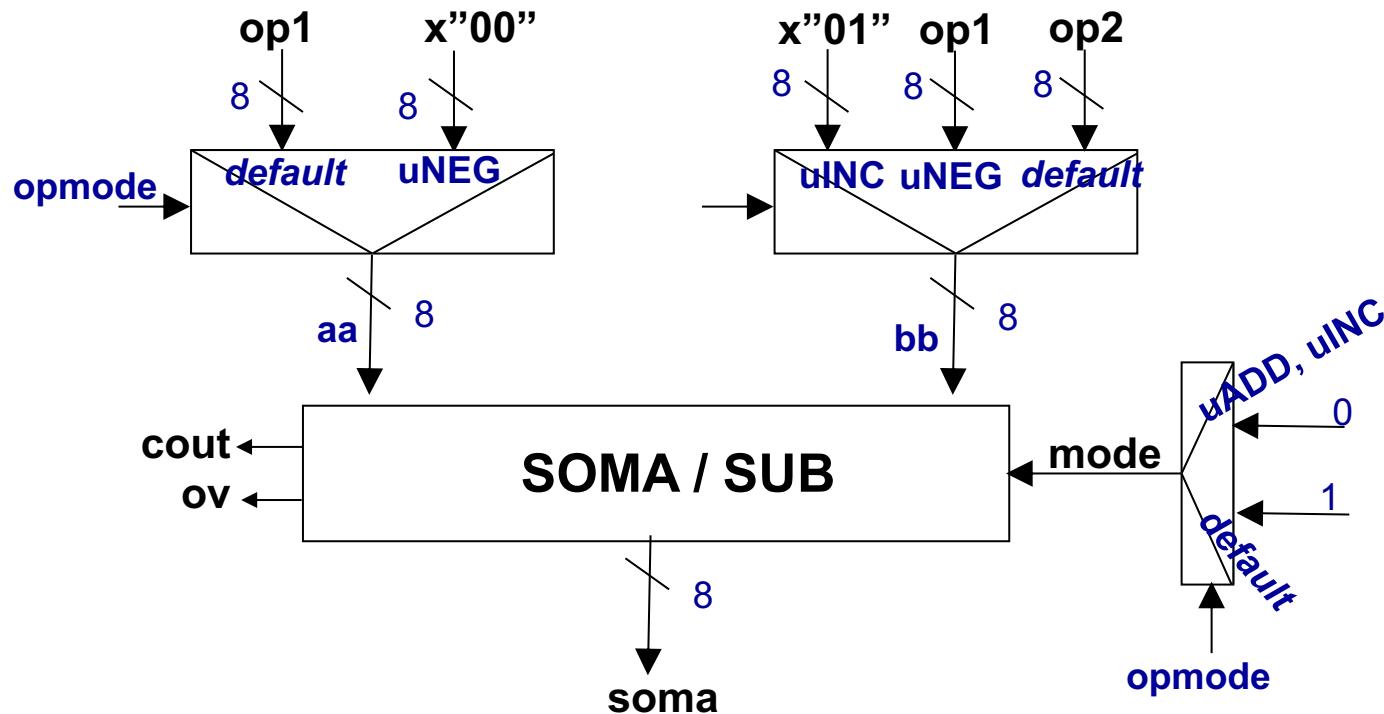
ULA - comandos

- Exemplo operações:

op_alu: AND, OR, XOR, SLL, SRL, ADD, SUB, INC, NEG



Instanciação do somador (soma/sub/inc/neg)

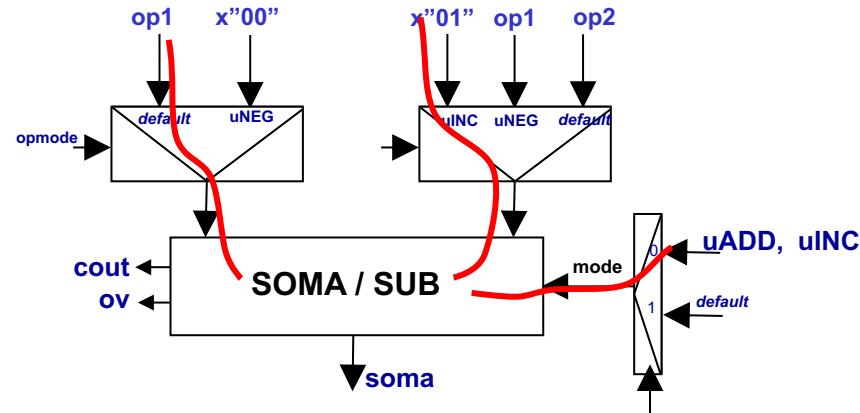


O somador
realiza 4
operações
aritméticas

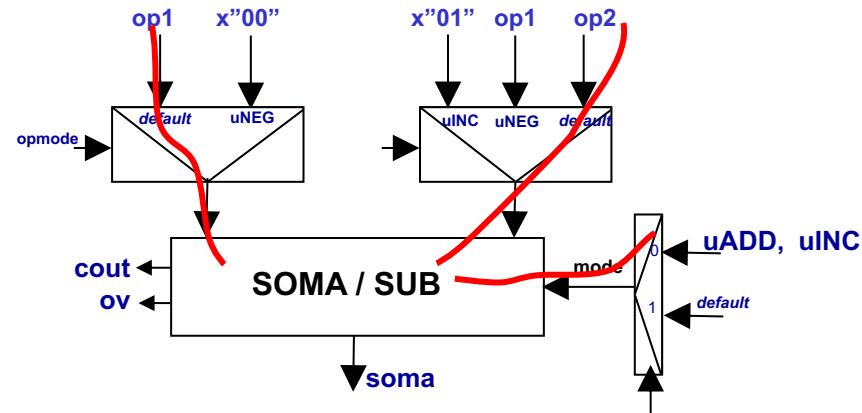
opmode	Ação
uINC	$op1 + 1 + 0$
uADD	$op1 + op2 + 0$
uSUB	$op1 + \text{not}(op2) + 1$
uNEG (2's comp)	$0 + \text{not}(op1) + 1$

Obs: o not do segundo operando deve-se ao mode=1

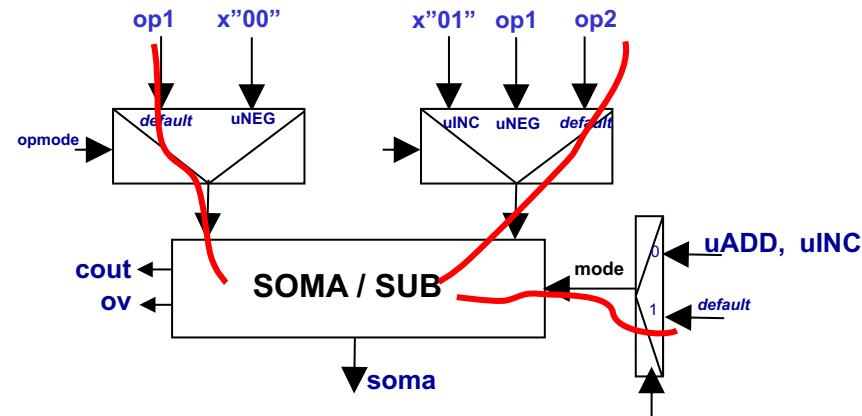
Instanciação do somador – operações



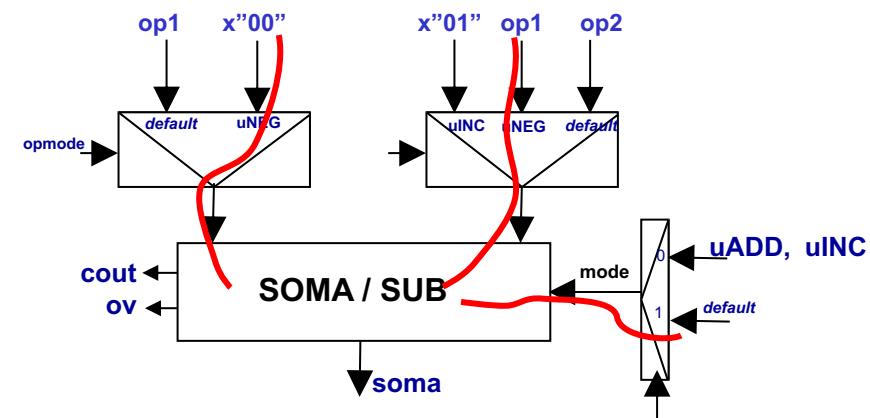
$$\text{Inc: } \text{op1} + 1 + 0$$



$$\text{ADD: } \text{op1} + \text{op2} + 0$$



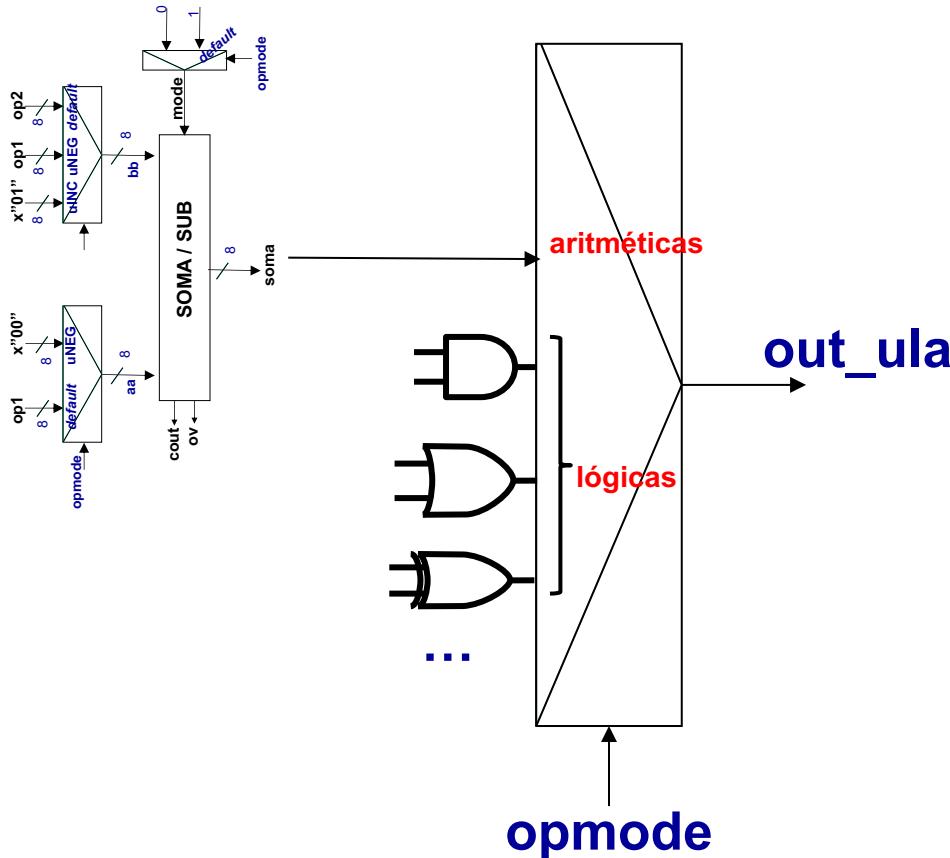
$$\text{Sub: } \text{op1} + \text{not}(\text{op2}) + 1$$



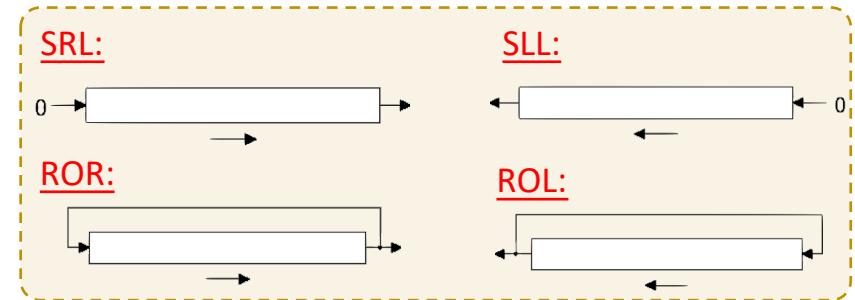
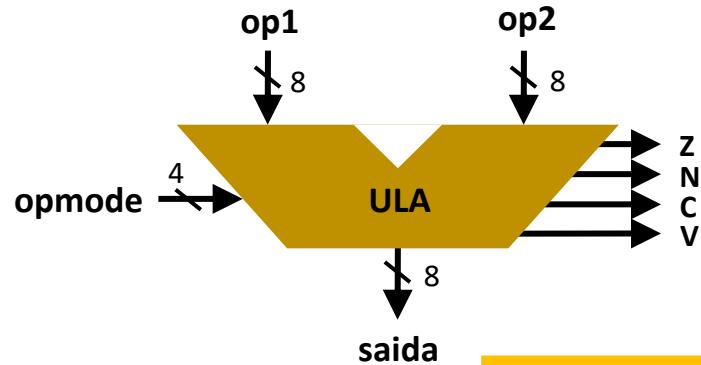
$$\text{NEG (2's com): } 0 + \text{not}(\text{op1}) + 1$$

Como integrar com as demais operações?

Utilização de multiplexadores



Exemplo de operações realizadas pela ULA

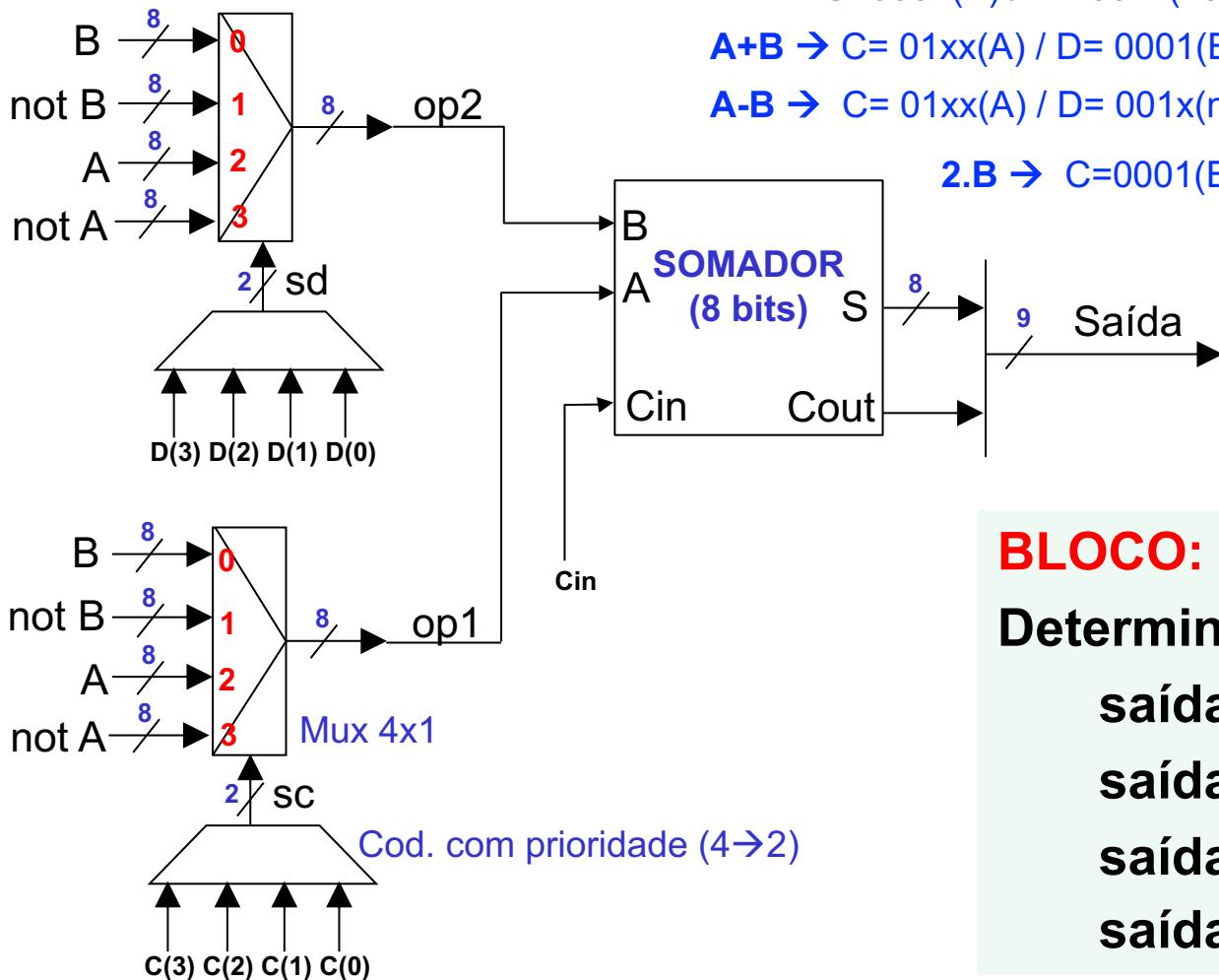


opmode	Ação	Operação
0000	$\text{saída} \leftarrow \text{op1} + \text{op2}$	Soma
0001	$\text{saída} \leftarrow \text{op1} - \text{op2}$	Subtração
0010	$\text{saída} \leftarrow \text{op1} + 1$	Incremento
0011	$\text{saída} \leftarrow \text{op1} - 1$	Decremento
0100	$\text{saída} \leftarrow \text{NOT}(\text{op1})$	Complemento
0101	$\text{saída} \leftarrow \text{op1} . \text{op2}$	AND
0110	$\text{saída} \leftarrow \text{op1} + \text{op2}$	OR
0111	$\text{saída} \leftarrow \text{op1} \oplus \text{op2}$	XOR
1000	$\text{saída} \leftarrow \text{SRL}(\text{op1})$	Deslocamento lógico à direita
1001	$\text{saída} \leftarrow \text{SLL}(\text{op1})$	Deslocamento lógico à esquerda
1010	$\text{saída} \leftarrow \text{ROR}(\text{op1})$	Rotação à direita
1011	$\text{saída} \leftarrow \text{ROL}(\text{op1})$	Rotação à esquerda
11--	$\text{saída} \leftarrow \text{op1}$	Transferência

Outro exemplo de circuito aritmético

Entradas: A e B de 8 bits, C e D controles de 4 bits, modo (Cin)

Saída: 9 bits



$100 \rightarrow A-A \text{ ou } B-B$

$C = 01xx(A) / D=1xxx(\text{not } A) / \text{Cin}=1$

$C=0001(B) / D= 001x(\text{not } B) / \text{Cin}=1$

$A+B \rightarrow C = 01xx(A) / D= 0001(B) \text{ ou } C=0001(B) \text{ e } D=01xx(A) / \text{Cin}=0$

$A-B \rightarrow C = 01xx(A) / D= 001x(\text{not } B) / \text{Cin}=1$

$2.B \rightarrow C=0001(B) / D= 0001(B) / \text{Cin}=0$

BLOCO: apenas somador

Determinar C / D / Cin para:

saída $\leftarrow 100_{16}$

saída $\leftarrow A + B$

saída $\leftarrow A - B$

saída $\leftarrow 2.B$

Recapitulando os blocos vistos

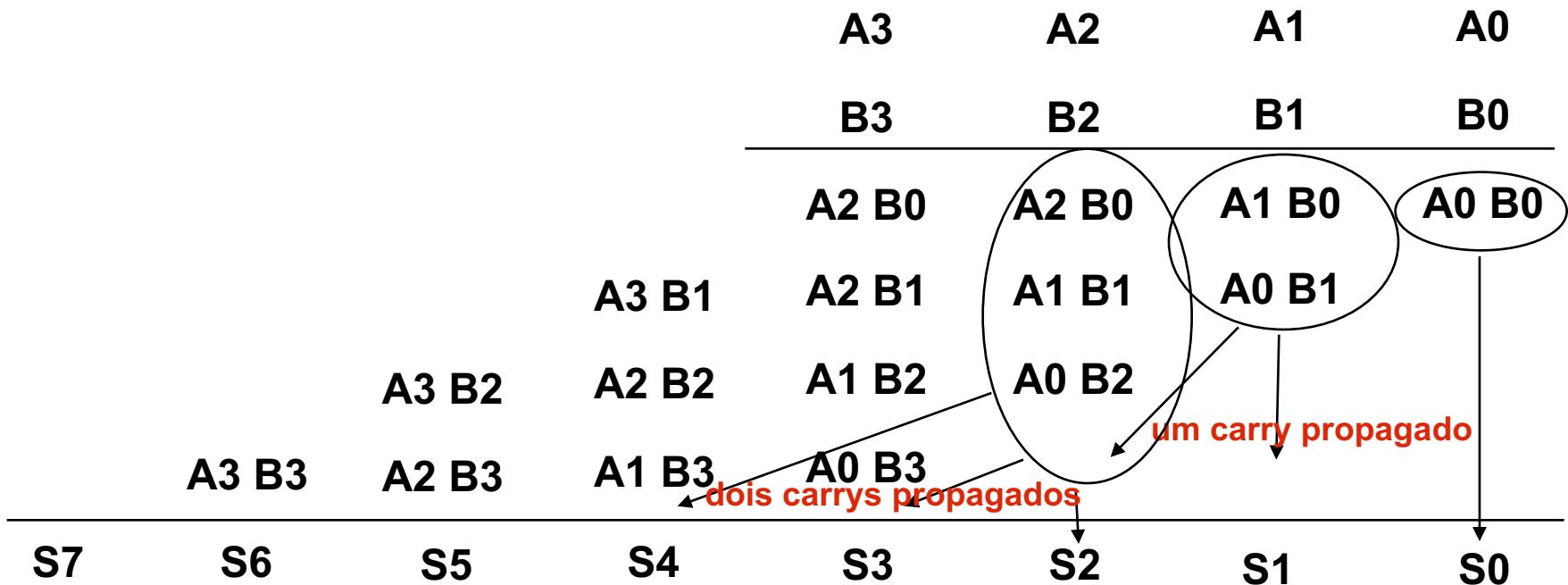
1. (De)Codificador
2. Comparador
3. Gerador de paridade e verificação de paridade
4. (De)multiplexador
5. PLA - Matrizes Lógicas Programáveis
6. Memória ROM
7. Soma e subtração (revisão de int/2's)
8. ULA – Unidade Lógica e Aritmética
9. Multiplicador (próximo bloco)

(9) MULTIPLICADOR

- Dois números de n bits quando multiplicados, geram $2n$ bits
- Exemplo:

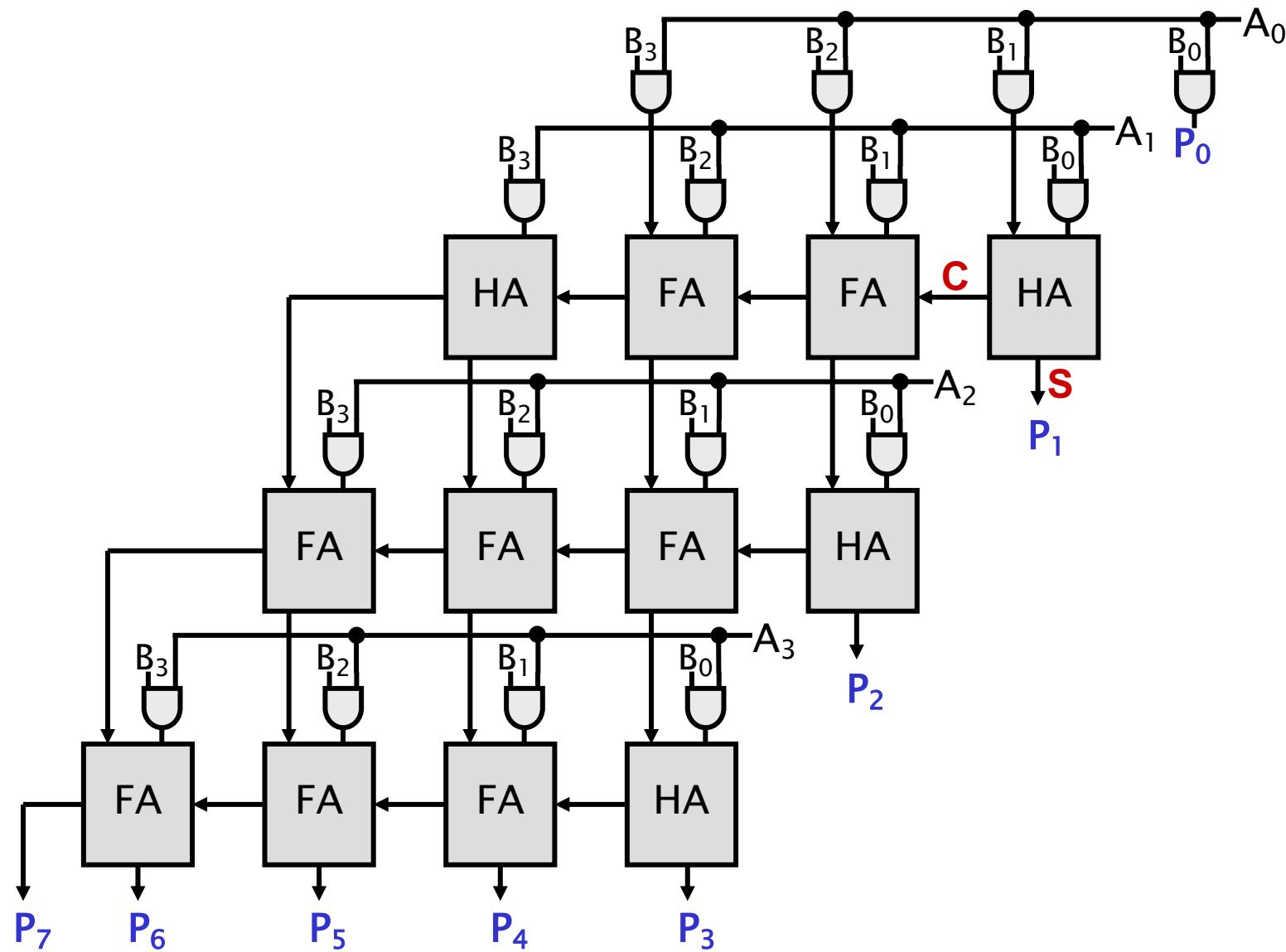
	multiplicand		1101 (13)
	multiplier		* $\frac{1011}{1101}$ (11)
Partial products	$\left\{ \begin{array}{c} 1101 \\ 1101 \\ 0000 \\ \hline 1101 \end{array} \right.$		10001111 (143)

MULTIPLICAÇÃO



Quanto maior o número de produtos parciais a somar maior o número de bits de vai-um gerados

MULTIPLICAÇÃO – arquitetura "array"



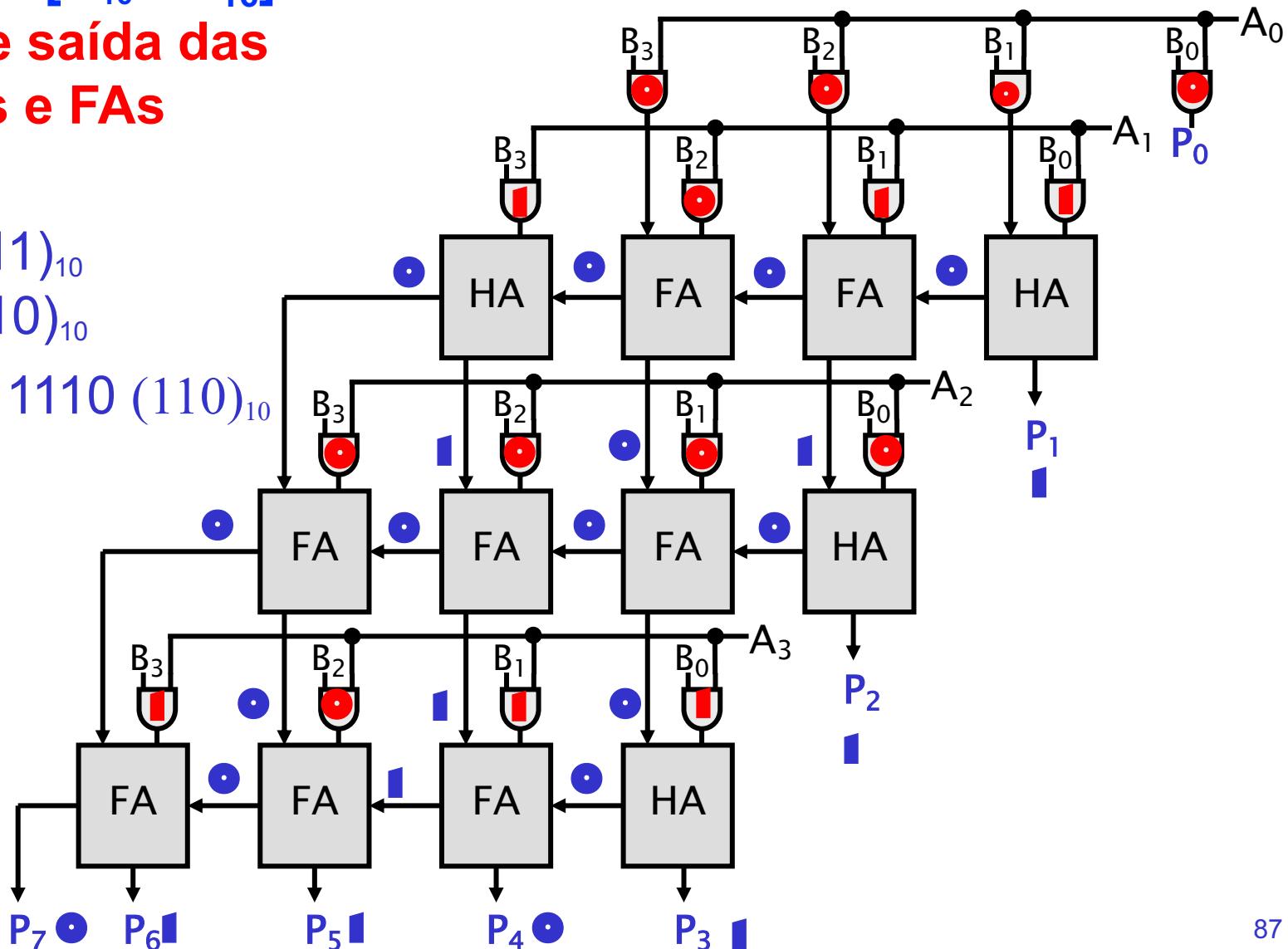
MULTIPLICAÇÃO – arquitetura "array"

Multiplique $[B_{16} * A_{16}]$ escrevendo os valores de saída das ands, HAs e FAs

$$B = 1011 \ (11)_{10}$$

$$A = 1010 \ (10)_{10}$$

$$6E \rightarrow 0110 \ 1110 \ (110)_{10}$$



Exercício (1/7)

Diferenciar as saídas `cout` e `overflow`, explicando o significado destes qualificadores.
Considerando números de 8 bits, preencha a tabela abaixo.

	Decimal sem sinal	Complemento de Dois
Valor mínimo	0	-128
Valor máximo	255	127

Avalie a soma abaixo considerando os números representados em inteiro sem sinal e complemento de dois. Quais os valores de `Cout` e `Ov` obtidos?

$$\begin{array}{r} & \boxed{1} \\ & 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ + & 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\ \hline & \boxed{1}\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

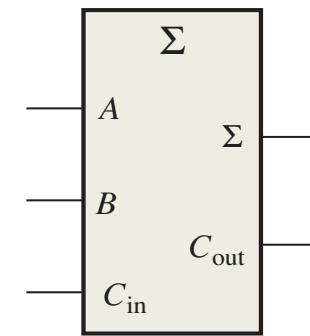
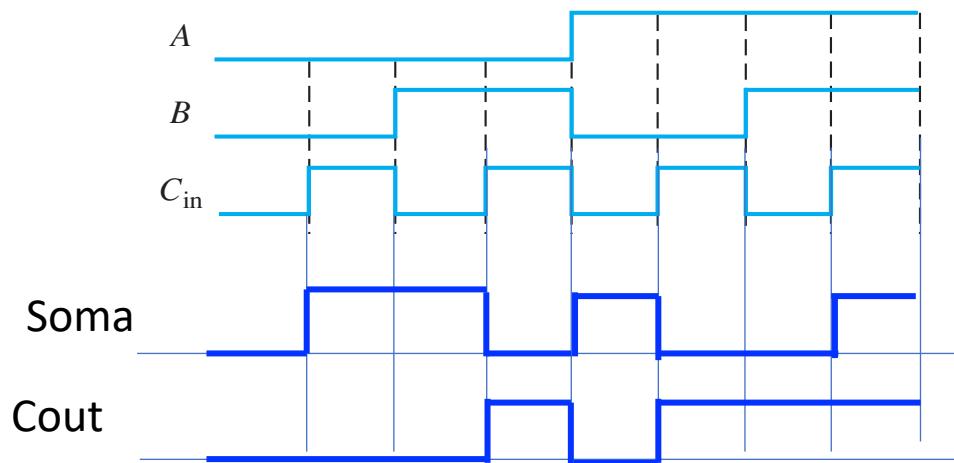
$\underbrace{}_{44}$

$(100)_{10}$ $(100)_{10}$
 $(200)_{10}$ $(-56)_{10}$

Soma correta em 2's mas incorreta em int. positivos

Exercício (2/7)

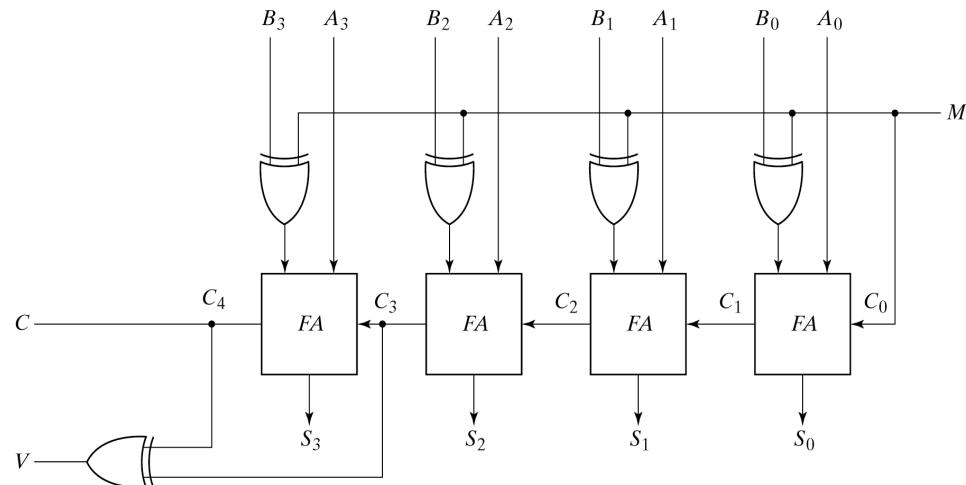
Determine as saídas *Soma* a *Cout* para os estímulos fornecidos.



Exercício (3/7)

Considere o circuito SOMA/SUB detalhado abaixo.

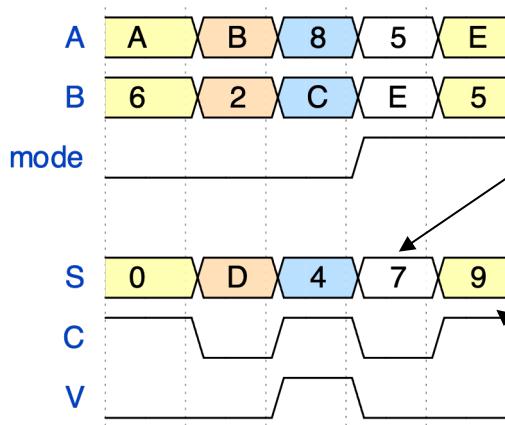
Determine a saída soma (S), carry out (C) e overflow (V) para os estímulos fornecidos.



$$\begin{array}{r}
 110 \\
 1010 \\
 0110 \\
 \hline
 1\ 0000
 \end{array}$$

$$\begin{array}{r}
 0101 \\
 -1110 \\
 \hline
 0\ 1 \\
 0101 \\
 0001 \\
 \hline
 0\ 0111
 \end{array}$$

N	int
8	-8
9	-7
A	-6
B	-5
C	-4
D	-3
E	-2
F	-1

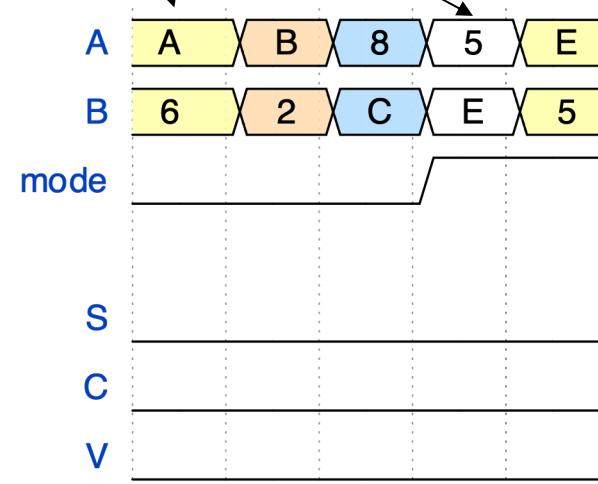


$$5 - (-2) : 5 + 2 = 7 \text{ (sem ov)}$$

5-14 (ok, cout=0 indicando que há erro - não há negativo nos Naturais)

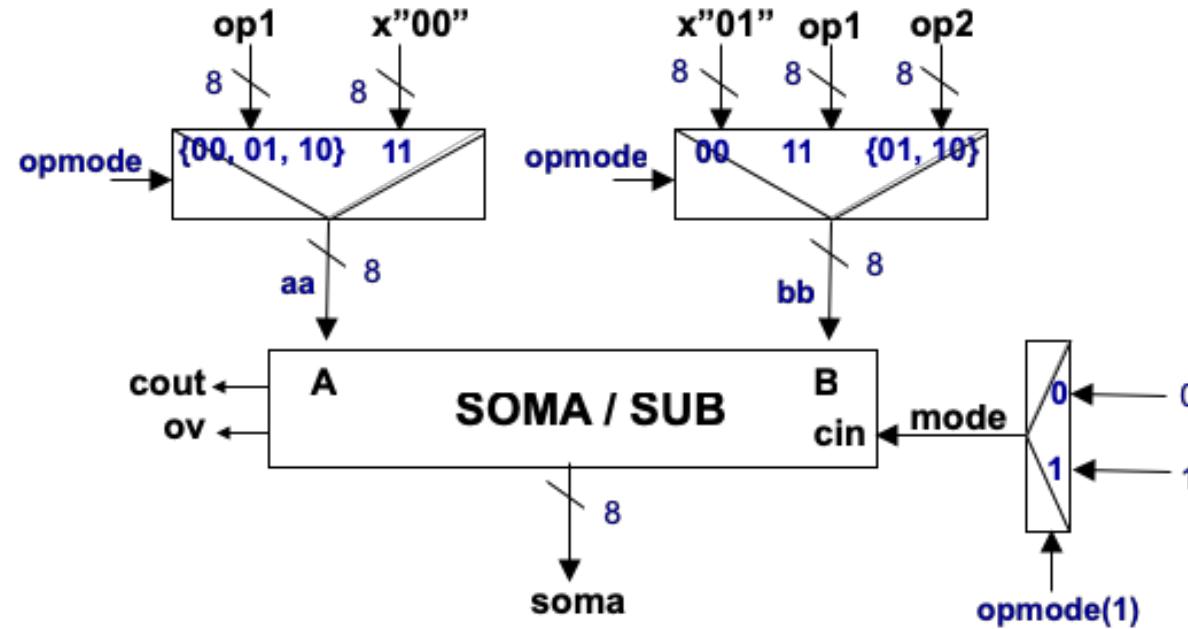
$$-2 - 5 = -7 (1001) \text{ (sem ov)}$$

14 - 5 = 9 (ok, cout=1 pois cout é invertido na subtração)



Exercício (4/7)

2. O circuito somador/somador é parte integrante de uma ULA. Para um *opmode* de 2 bits, determine quais operações circuito soma/sub realiza, conforme a tabela abaixo.



OPMODE	aa	bb	mode	Soma	Operação
00	op1	01	0	op1 + 1	INC op1
01	op1	op2	0	op1 + op2	soma
10	op1	op2	1	op1 + op2' + 1	op1 - op2
11	00	op1	1	op1' + 1	- op1

Exercício (5/7)

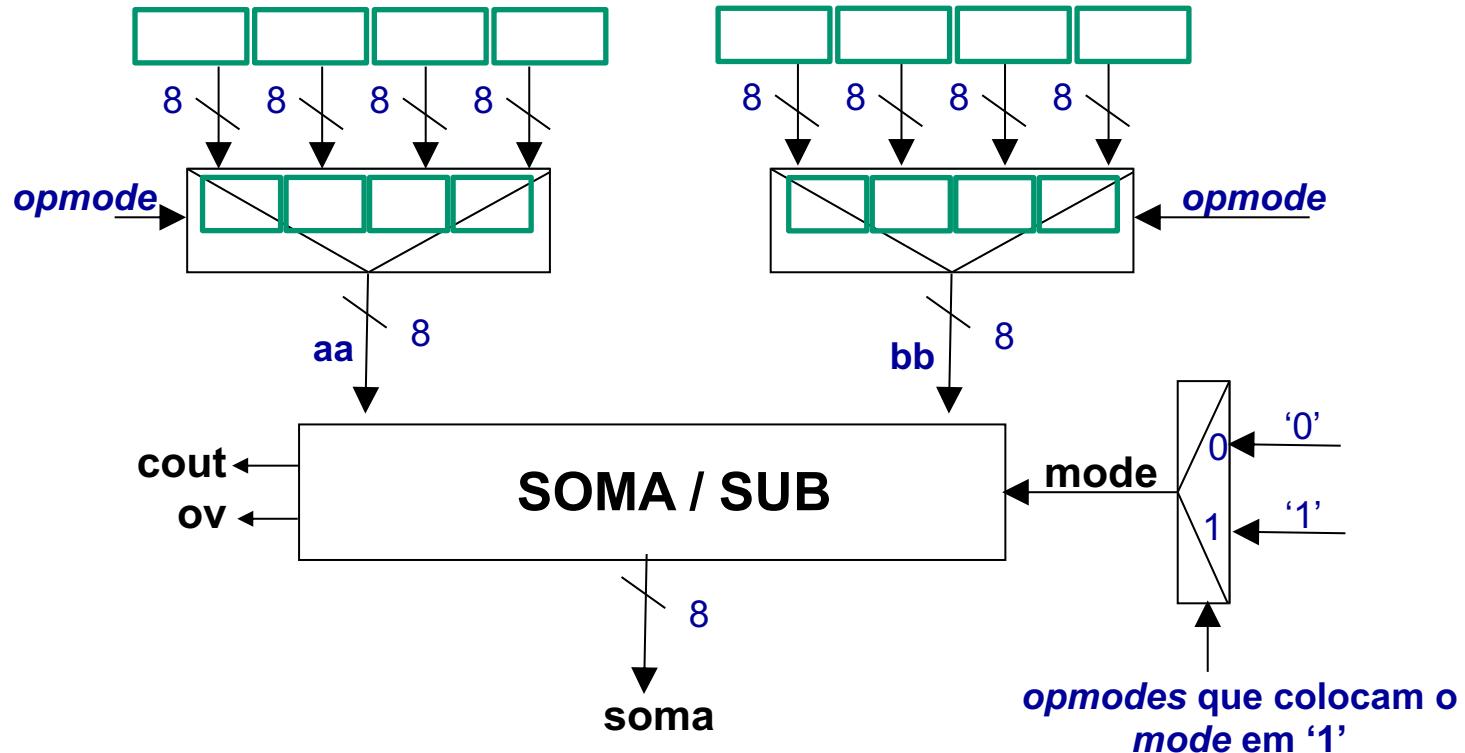
Projete a parte aritmética de uma ULA que realize 8 funções:

opmode = { SUM, SUB, MU2, INC, DEC, NEG, P1, Z }

Funções realizadas por opcode:

op1+op2, op1-op2, 2*op1, op1+1, op1-1, 2's de op1, op1 (P1: deixa passar op1), Z: sai zero

- Os multiplexadores de entrada não precisam ser 4 para 1 – utilizar o que for necessário. Preencher para os multiplexadores de entrada o operando de entrada (por exemplo, op1, op2, x"00", x"01", x"FF") e qual opcode seleciona este operando (pode ser *default*, ou seja, uma entrada padrão). No multiplexador do *mode* indicar quais operações colocam a saída deste multiplexador em '1'.



Exercício (solução)

Projete a parte aritmética de uma ULA que realize 8 funções:

opmode = { SUM, SUB, MU2, INC, DEC, NEG, P1, Z }

Funções realizadas por *opcode*:

$op1+op2$, $op1-op2$, 2^*op1 , $op1+1$, $op1-1$, 2's de $op1$, $op1$ ($P1$: deixa passar $op1$), Z : sai zero

1: sum sub mu2 inc dec P1 Z

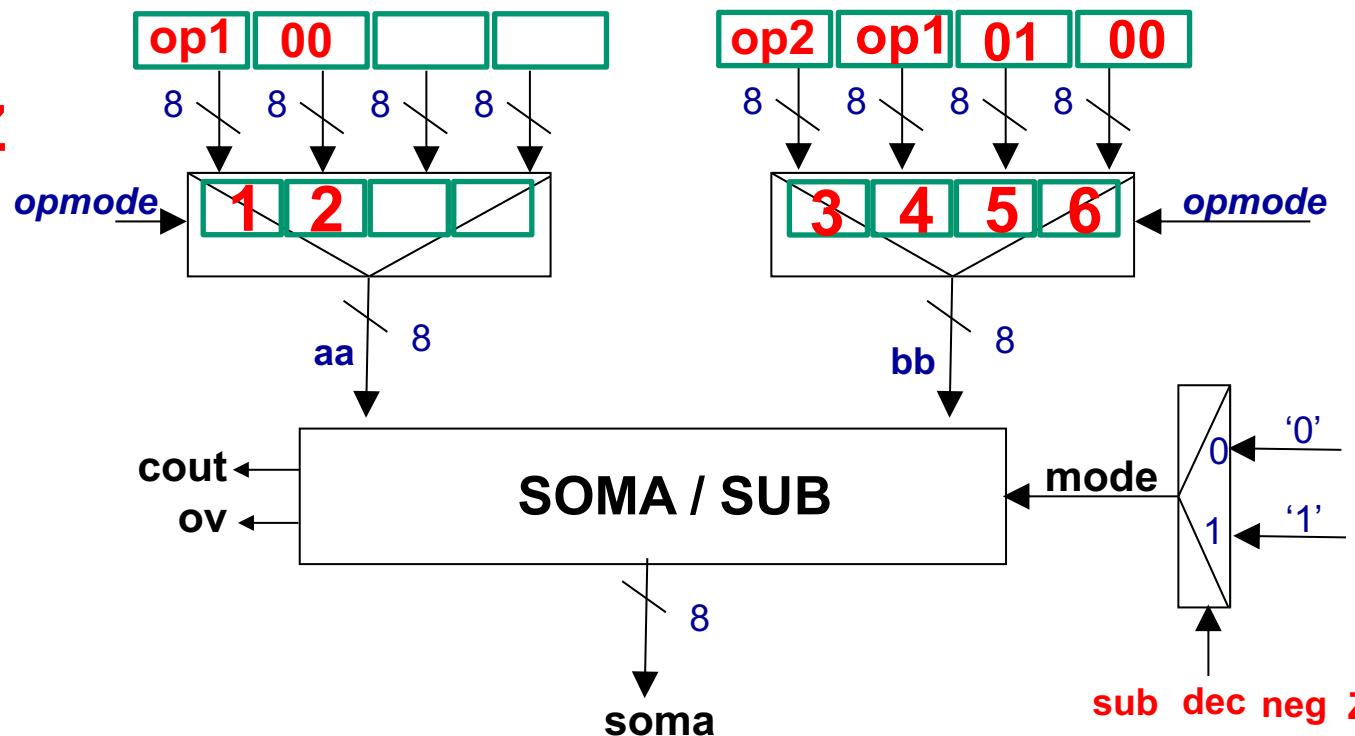
2: neg

3: sum sub

4: mu2 neg Z

5: inc dec

6: P1



Exercício (6/7)

Unidade Lógico Aritmética. Considere uma ULA abaixo capaz de realizar 9 operações lógico-aritméticas. Determine o valor de *outalu* para o vetor de entrada (**x"34", x"7E"**).

outalu <= op1 and op2	when opmode=AND else
op1 or op2	when opmode=OR else
op1 xor op2	when opmode=XOR else
op1(6 downto 0) & '0'	when opmode=SLL else
'0' & op1(7 downto 1)	when opmode=SRL else
soma;	--- ADD, SUB, INC, NEG

Op1	Op2	AND	OR	XOR	SLL	SRL	ADD	SUB	INC	NEG
x"34"	x"7E"									

Exercício (6/7) - SOLUÇÃO

Unidade Lógico Aritmética. Considere uma ULA abaixo capaz de realizar 9 operações lógico-aritméticas. Determine o valor de *outalu* para o vetor de entrada (**x"34", x"7E"**).

outalu <= op1 and op2	when opmode=AND else
op1 or op2	when opmode=OR else
op1 xor op2	when opmode=XOR else
op1(6 downto 0) & '0'	when opmode=SLL else
'0' & op1(7 downto 1)	when opmode=SRL else
soma;	--- ADD, SUB, INC, NEG

Op1	Op2	AND	OR	XOR	SLL	SRL	ADD	SUB	INC	NEG
x"34"	x"7E"	x"34"	x"7E"	x"4A"	x"68"	x"1A"	x"B2"	x"B6"	x"35"	x"CC"

x"34" : 0011 0100

X"7E": 0111 1110

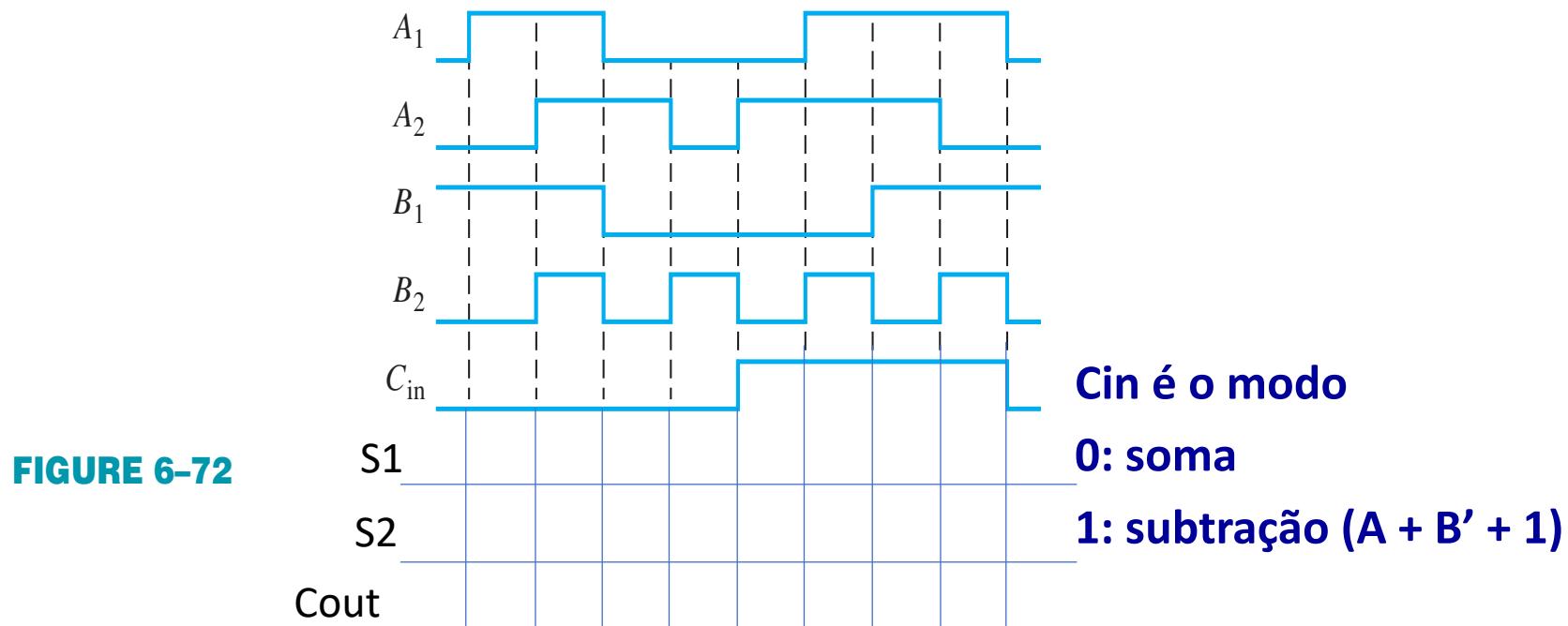
0110 1000

0001 1010

$$\begin{array}{r} \begin{array}{r} 1111 \\ 0011\ 0100 \\ + 0111\ 1110 \\ \hline (0)\ 1011\ 0010 \end{array} & \begin{array}{r} 1 \\ 0011\ 0100 \\ + 1000\ 0001 \\ \hline 1011\ 0110 \end{array} & \begin{array}{r} 1100\ 1011 \\ + 1 \\ \hline 1100\ 1100 \end{array} \end{array}$$

Exercício (7/7)

8. The input waveforms in Figure 6–72 are applied to a 2-bit adder. Determine the waveforms for the sum and the output carry in relation to the inputs by constructing a timing diagram.



Exercício (7/7) - solução

8. The input waveforms in Figure 6–72 are applied to a 2-bit adder. Determine the waveforms for the sum and the output carry in relation to the inputs by constructing a timing diagram.

Apenas FA (não é subtração)

