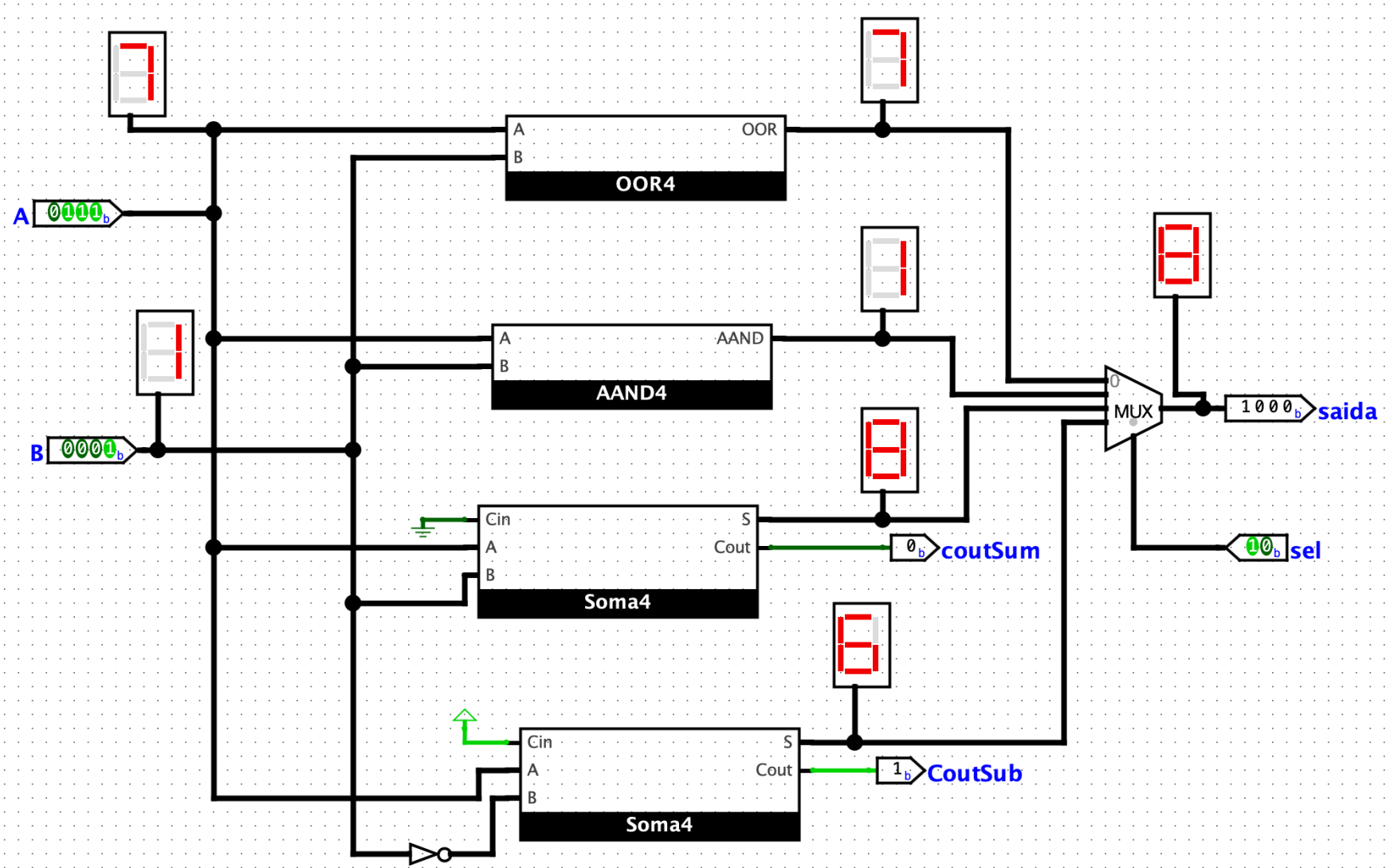
CIRCUITOS COMBINACIONAIS

Fernando G. Moraes

Fundamentos de Sistemas Digitais



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

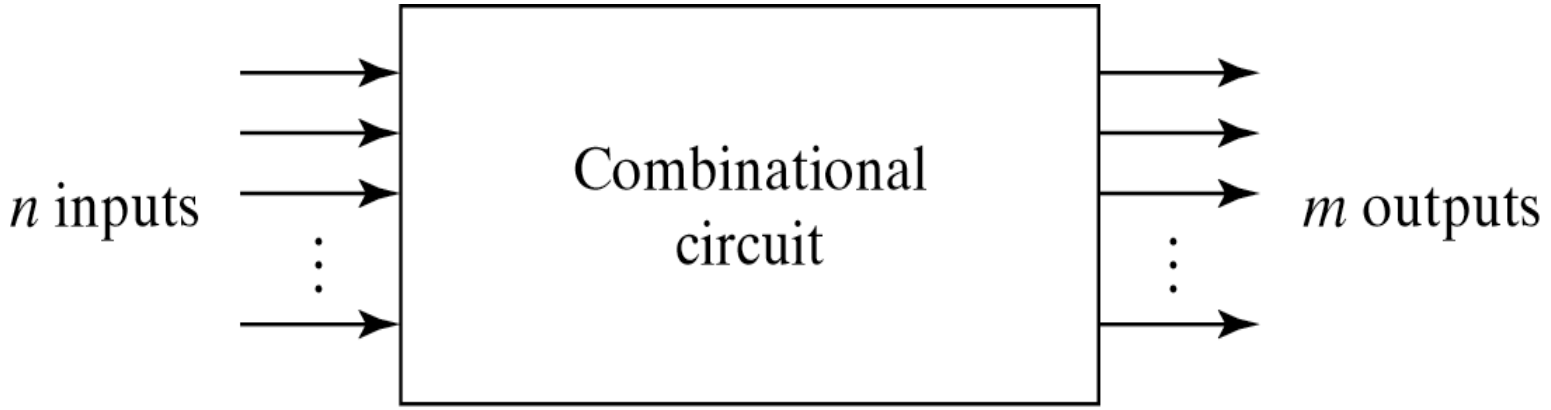
- Para cada combinação binária das variáveis de entrada, existe uma saída possível

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.

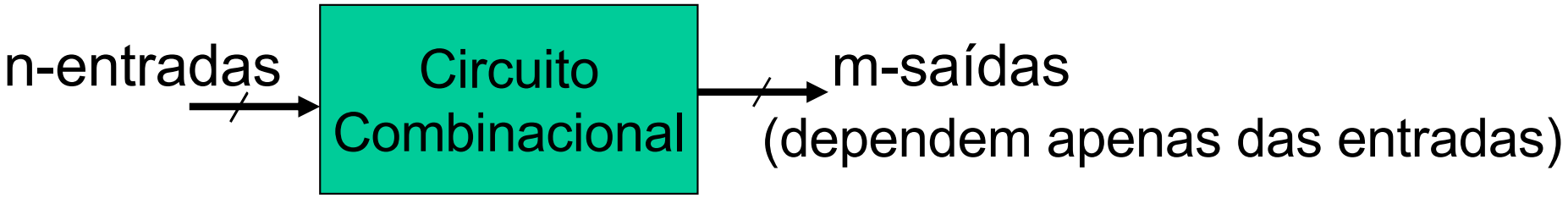


Circuitos Combinacionais versus Sequenciais

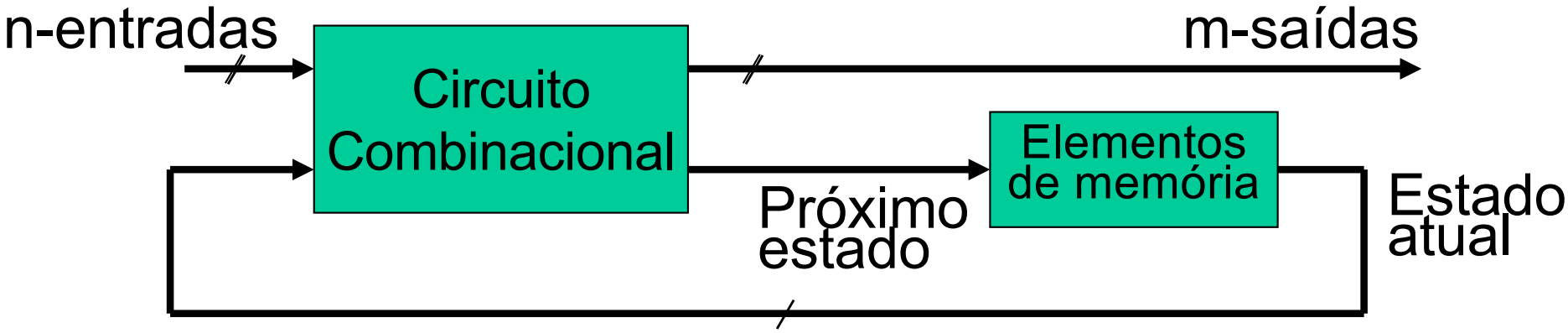
- Os circuitos combinacionais não possuem memória interna
  - O valor de saída depende **apenas** dos valores atuais de entrada

- 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

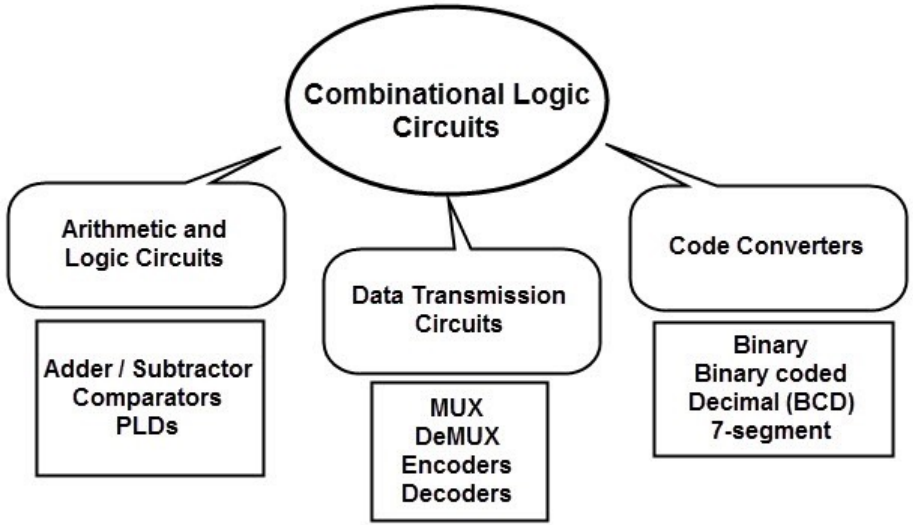


Circuito Sequencial

Circuitos básicos

- **Exemplos de circuitos combinacionais**
  1. Codificador/decodificador
  2. Multiplexador/demux
  3. Comparadores
  4. Geradores de paridade
  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

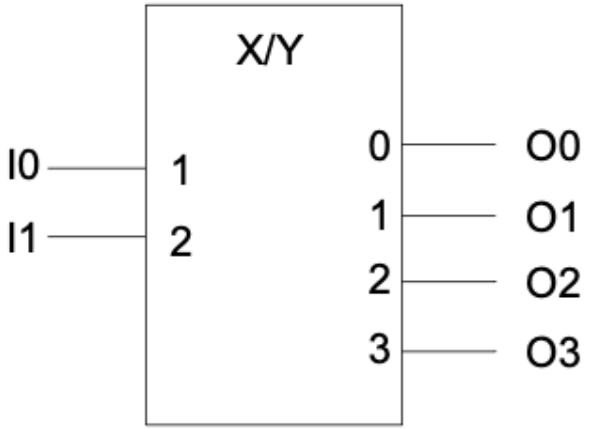


(1) (DE)CODIFICADOR

- ✓ O **decodificador** binário é um circuito combinacional que permite, perante uma combinação de entradas, **ativar uma e somente uma saída**.

$n \rightarrow 2^n$

I1	I0	O0	O1	O2	O3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

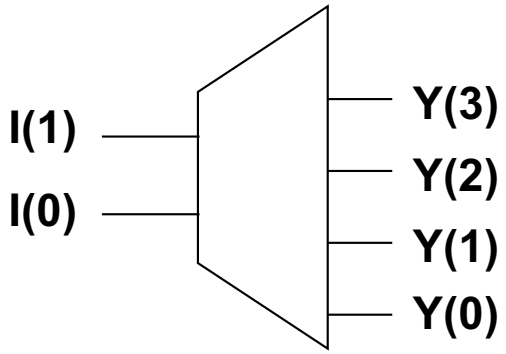


- ✓ No símbolo do componente, o índice dos sinais de entrada/saída permitem identificar claramente as saídas e o "peso" de cada um dos sinais de entrada.

DECODIFICADOR – portas lógicas (n→2^n)

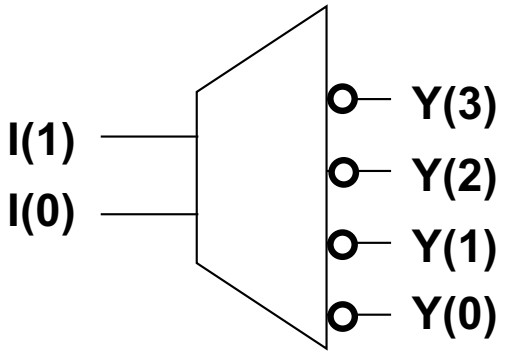
Ativo alto

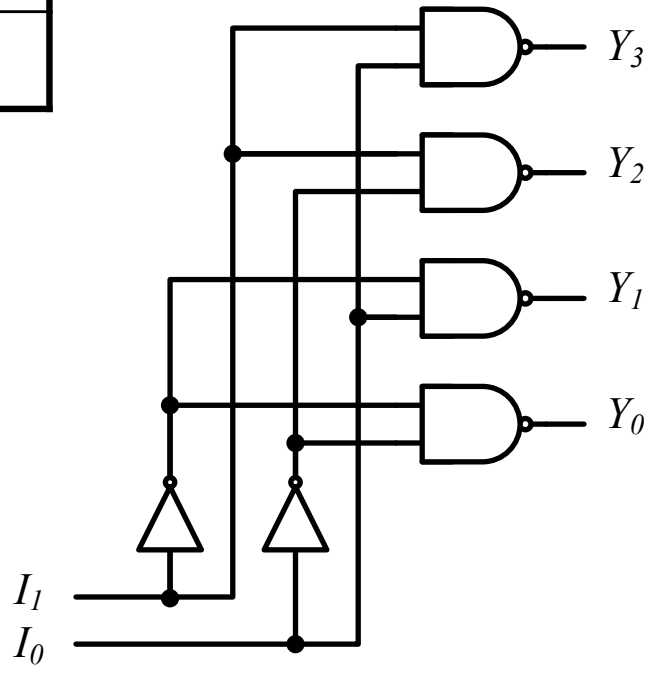
I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



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)

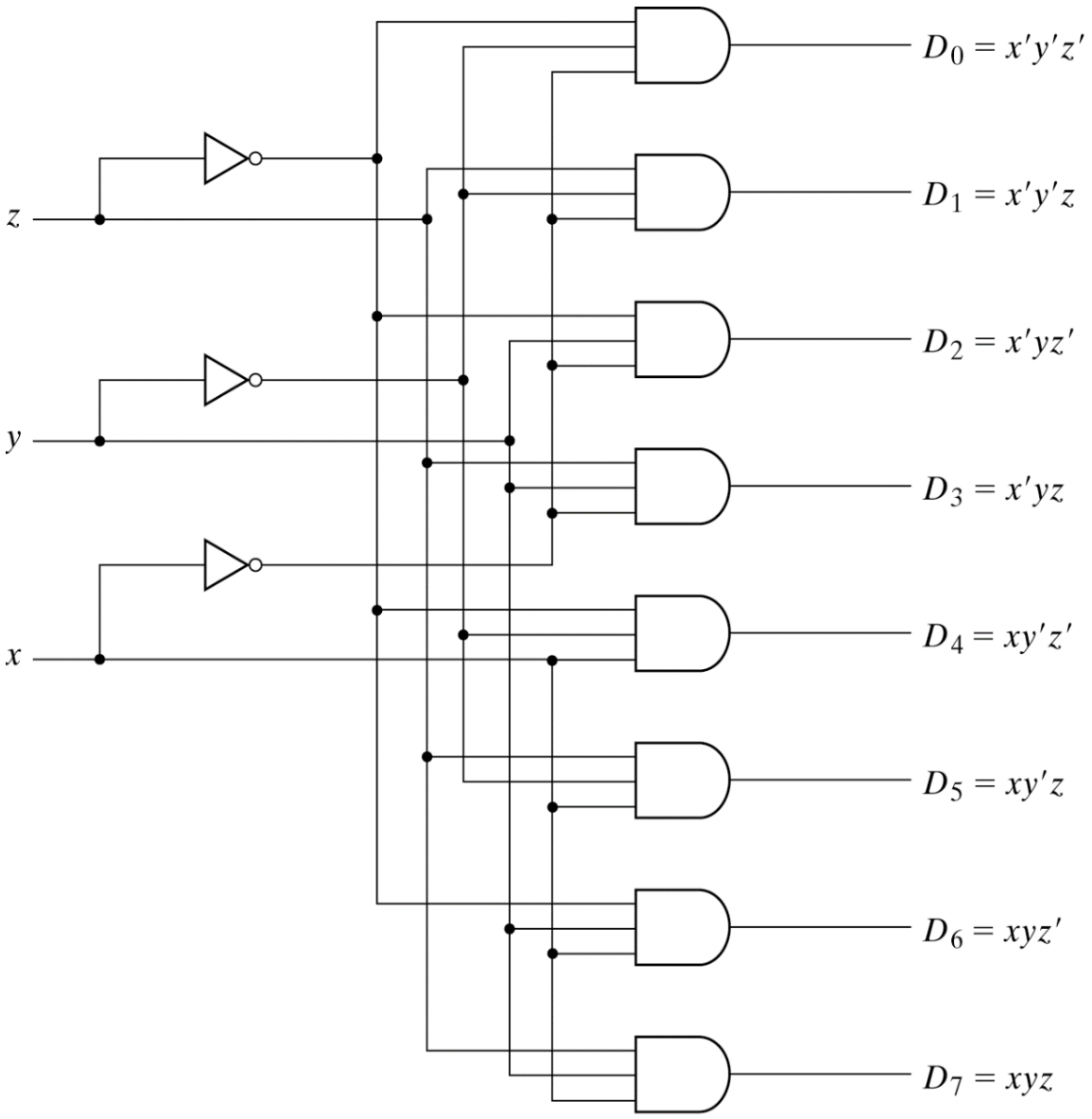


Fig. 4-18 3-to-8-Line Decoder

Cada porta **and** recebe um determinado endereço

DECODIFICADOR - 4-Bit Decoder (n→2^n)

- ✓ Exemplo de **decodificador** de 4-bits:
  - Note que apenas uma saída é ativada para cada valor binário de entrada

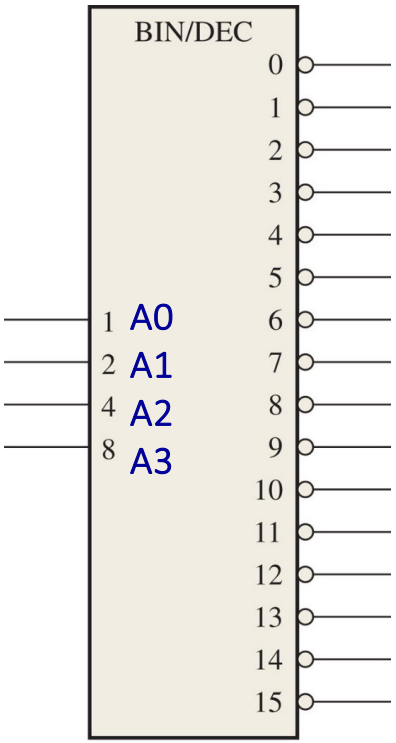


TABLE 6–4

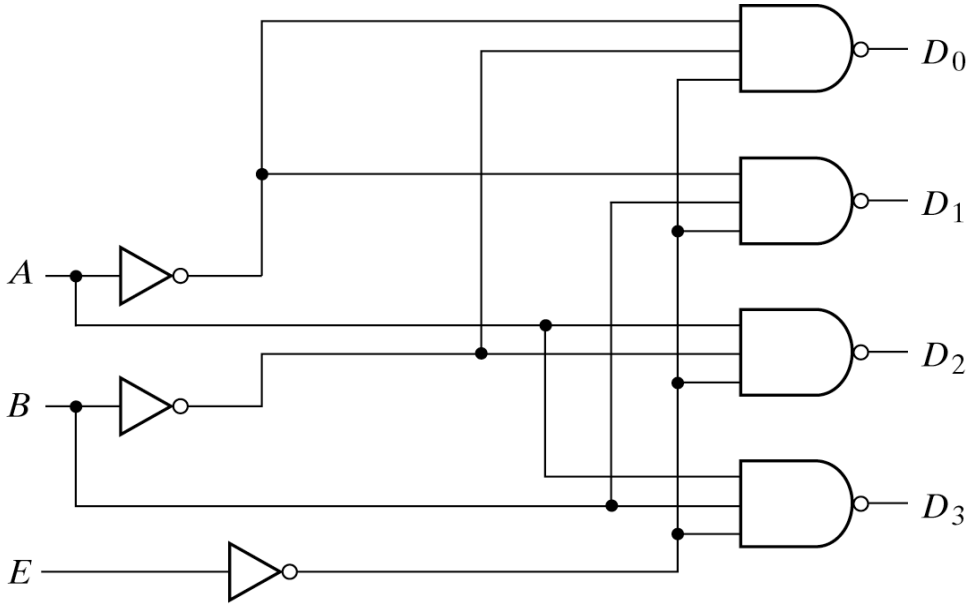
Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

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

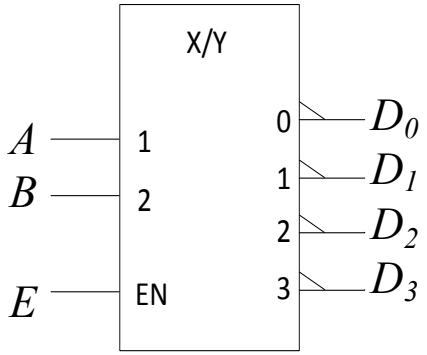
FIGURE 6-28 Logic symbol for a 4-line-to-16-line (1-of-16) decoder.

DECODIFICADOR – com sinal de habilitação

- A entrada **E** permite, quando ativa (**neste caso, "0"**), que o decodificador funcione normalmente. Quando não ativa, inibe o seu funcionamento fazendo com que todas as saídas fiquem inativas (neste caso, todas em "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

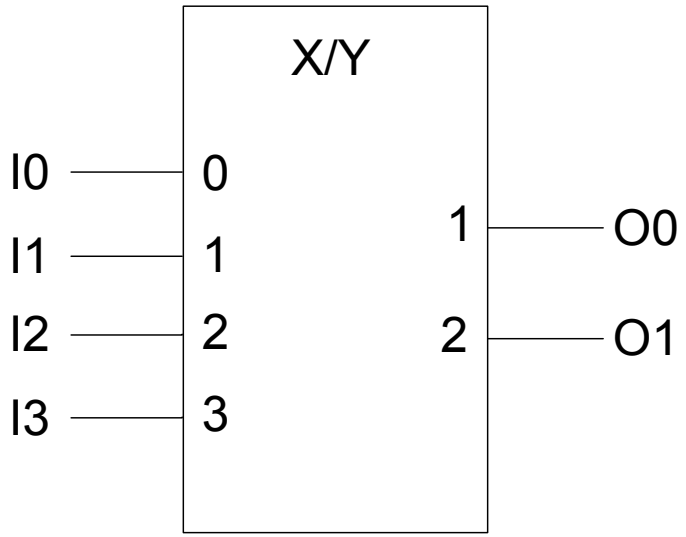
Fig. 4-19 2-to-4-Line Decoder with I

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

- O **codificador** binário é um circuito combinacional que indica qual das entradas possíveis está ativa (neste caso, "1").

$2^n \rightarrow n$

I3	I2	I1	I0	O1	O0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



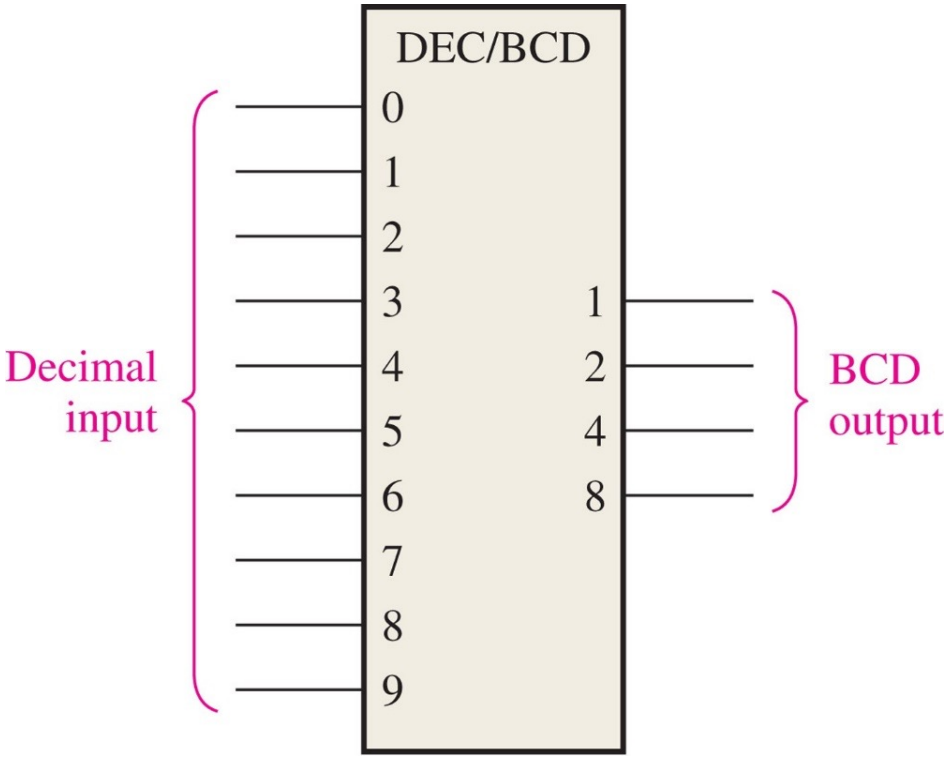
- Note que, nesta versão, o circuito não distingue a situação de todas as entradas estarem em "0".
- Note que, nesta versão, o circuito não distingue as situações em que mais de uma entrada esta em "1".

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	BCD Code 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



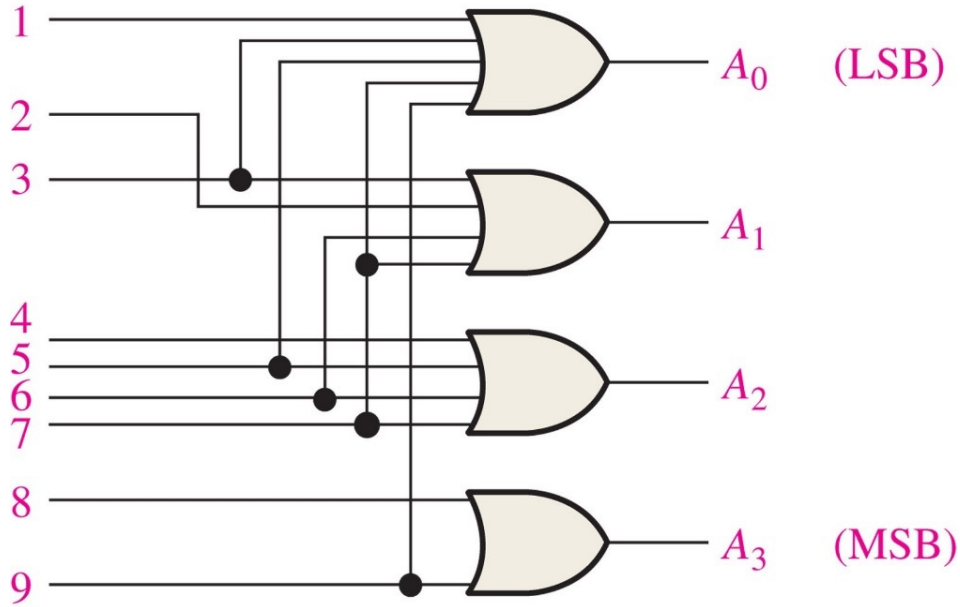
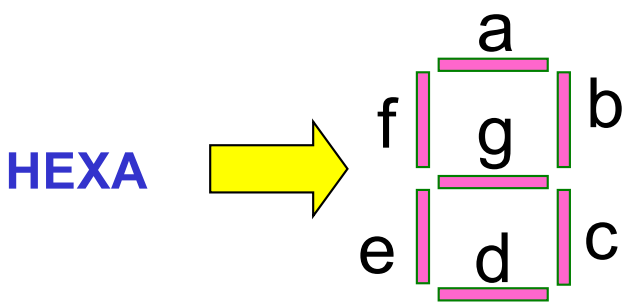


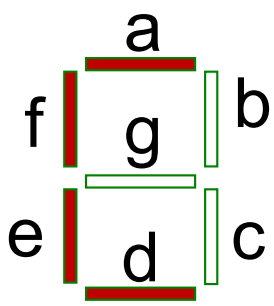
FIGURE 6-36 Logic symbol for a decimal-to-BCD encoder.

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

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

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

B

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

C

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

D

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

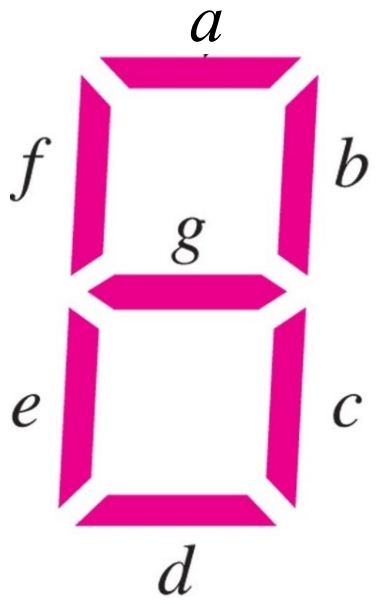
E

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

F

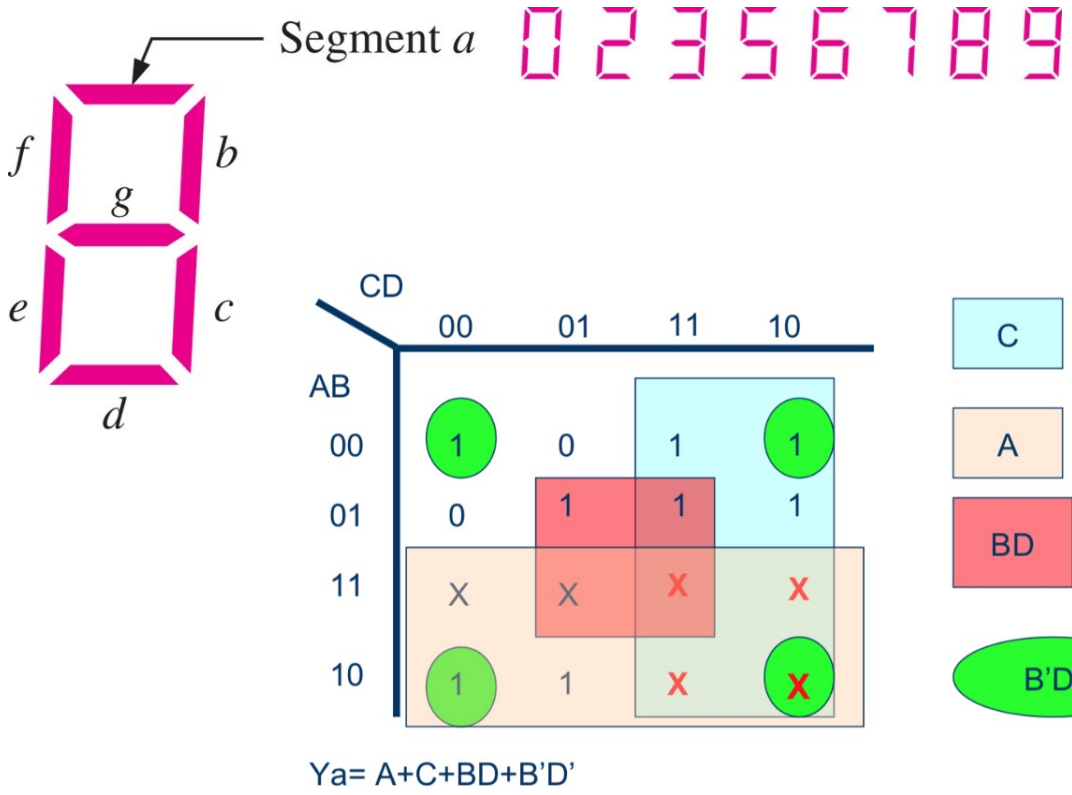
AB \ CD	00	01	11	10
00			1	1
01	1	1		1
11		1	1	1
10	1	1	1	1

G



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**

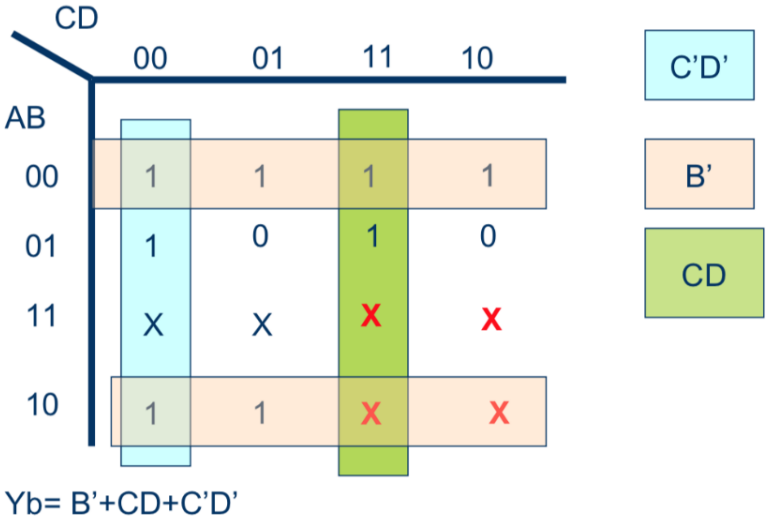


A3	A2	A1	A0	a	b	c	...
0	0	0	0	1			
0	0	0	1				
0	0	1	0	1			
0	0	1	1	1			
0	1	0	0				
0	1	0	1	1			
0	1	1	0	1			
0	1	1	1	1			
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	x			
1	0	1	1	x			
1	1	0	0	x			
1	1	0	1	x			
1	1	1	0	x			
1	1	1	1	x			

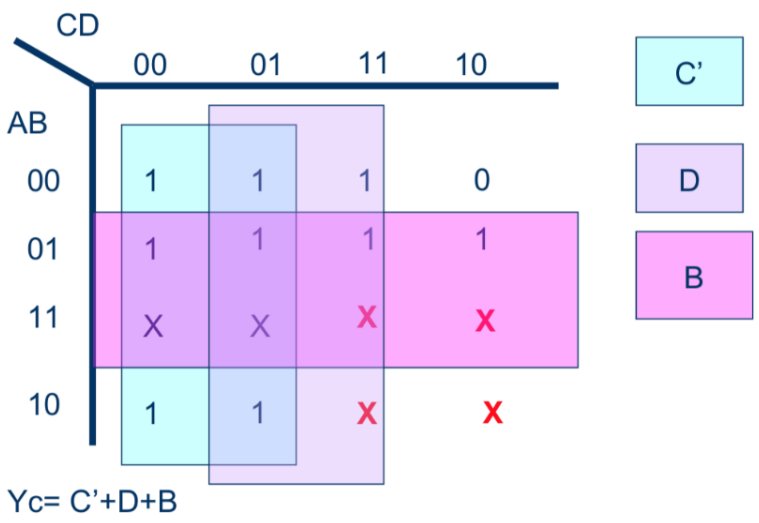
- **7 mapas de Karnaugh, um para cada dígito**

http://www.sj.ifsc.edu.br/~odilson/ELD/Display%20de%207%20segmentos.pdf

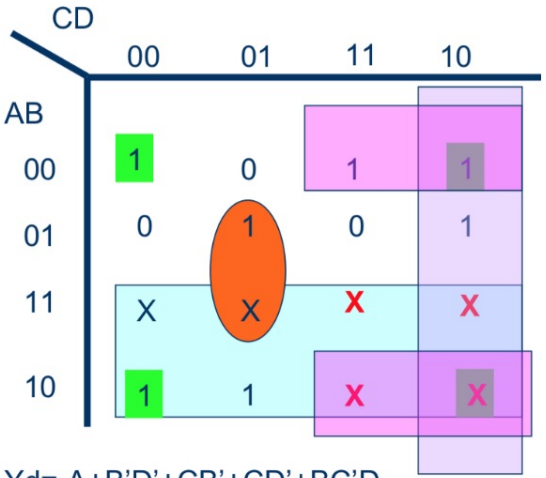
CODIFICADOR – decimal para sete-segmentos



Yb= B'+CD+C'D'



Yc= C'+D+B



Yd= A+B'D'+CB'+CD'+BC'D

Ye= B'D'+CD'

http://www.sj.ifsc.edu.br/~odilson/ELD/Display%20de%207%20segmentos.pdf

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.

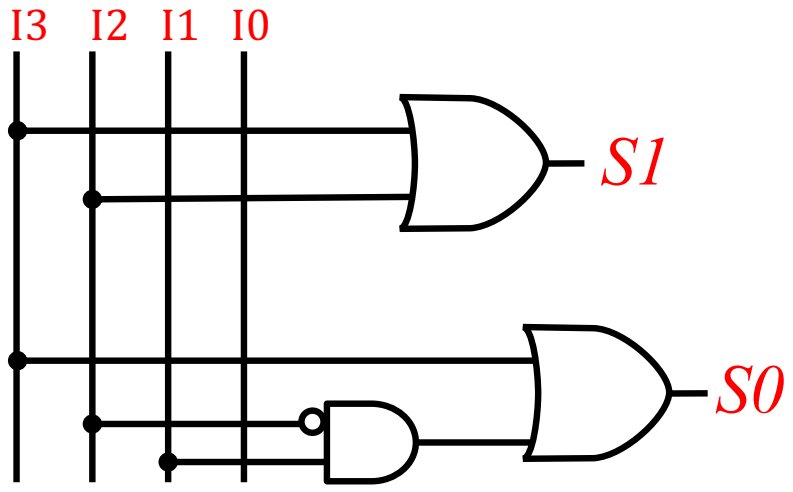
- Exemplo - I3 tem maior prioridade (todos bits em 0 não ocorre) :

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

$$S1 = I3 + I2$$

$$S0 = I3 + \overline{I2}.\, I1$$

- Circuito resultante:



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*)

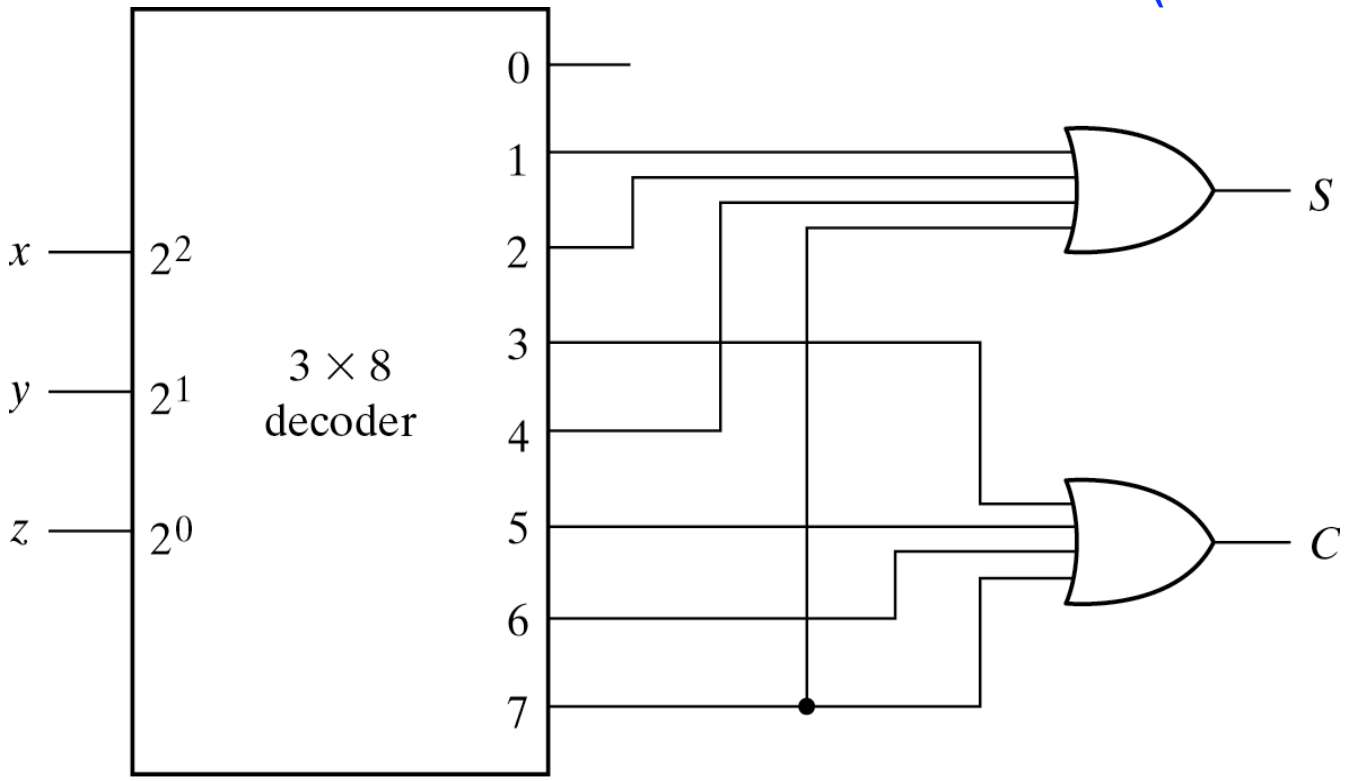
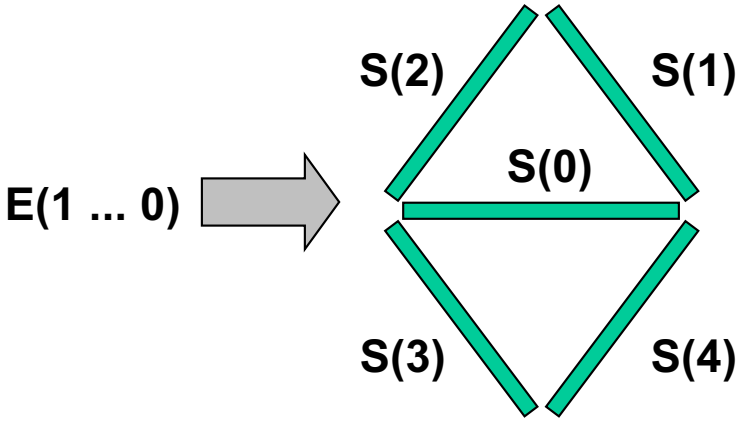


Fig. 4-21 Implementation of a Full Adder with a Decoder

Exercícios

1. Faça a codificação do display de elevador ilustrado abaixo:



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

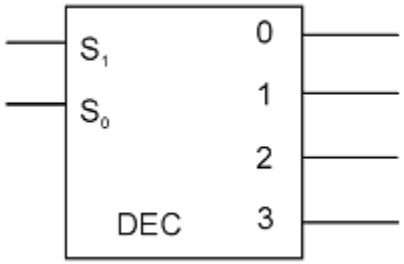
parado	**00**	**-**
subindo	**01**	**/**
descendo	**10**	**\/**
defeito	**11**	**todos segmentos acesos**

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

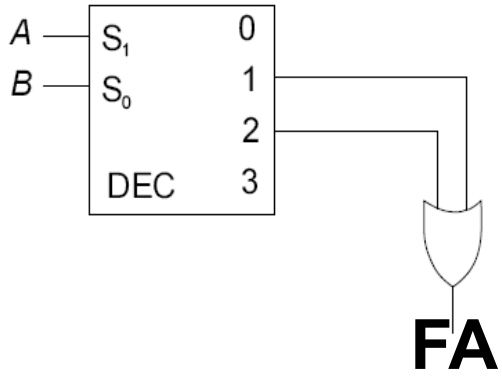
S4 = S3 = E1
S2 = S1 = E0
S0 = E1 *xnor* ***E0***

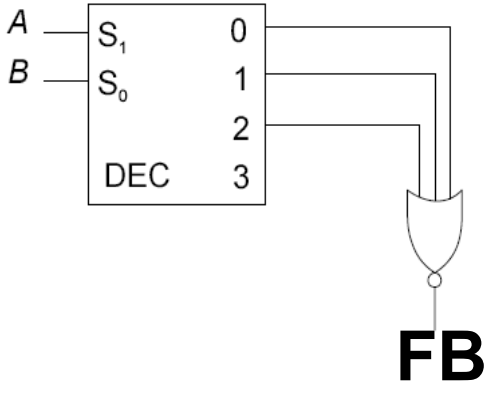
Exercícios

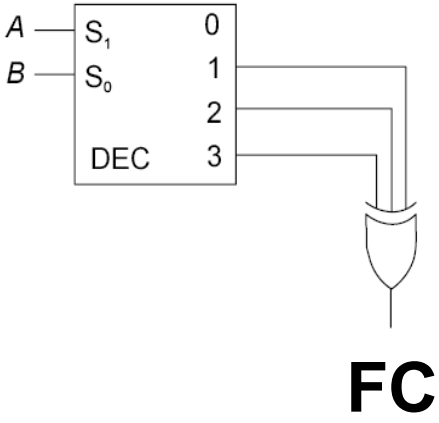
2. Considere o decodificador abaixo e sua correspondente tabela verdade.
   - ✓ Determine as funções lógicas a seguir:



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







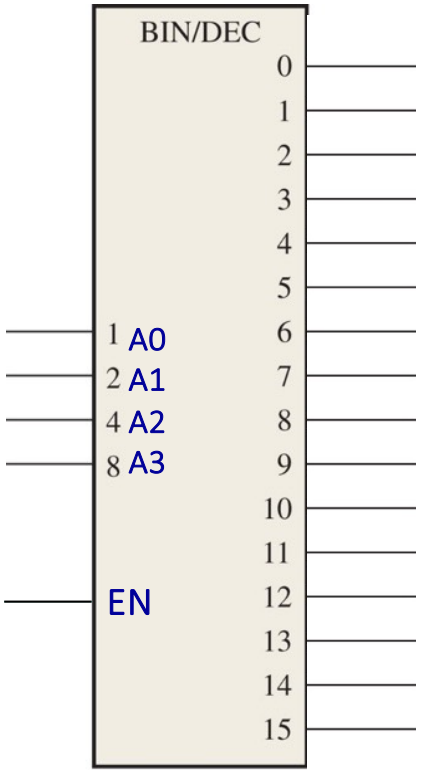
S_1	S_0	0 $\overline{A}.\overline{B}$	1 $\overline{A}.B$	2 $A.\overline{B}$	3 $A.B$	FA 1 or 2	FB (0 or 1 or 2)'	FC 1 xor 2 xor 3
0	**0**	1	0	0	0	**0**	**0**	**0**
0	**1**	0	1	0	0	**1**	**0**	**1**
1	**0**	0	0	1	0	**1**	**0**	**1**
1	**1**	0	0	0	1	**0**	**1**	**1**
						XOR	**AND**	**OR**

Exercícios

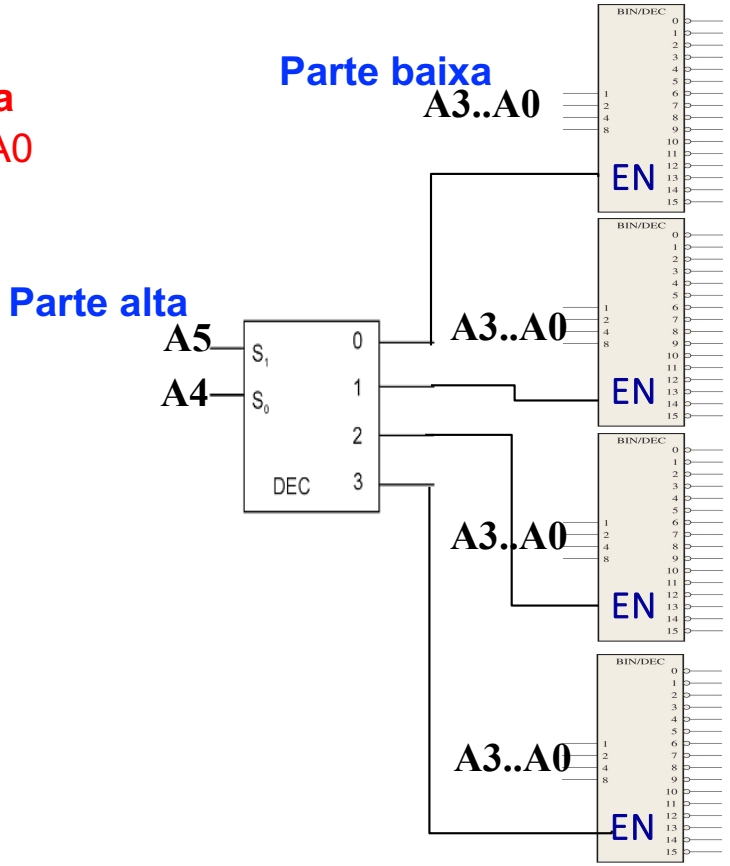
3. Supondo que desejamos realizar o acesso a uma linha de dados de uma memória de 64 palavras, mas possuímos decodificadores 4→16 e 2→4. Como podemos construir um decodificador 6→64 utilizado estes decodificadores?

Solução: dividir o endereço em parte alta e baixa

4→16



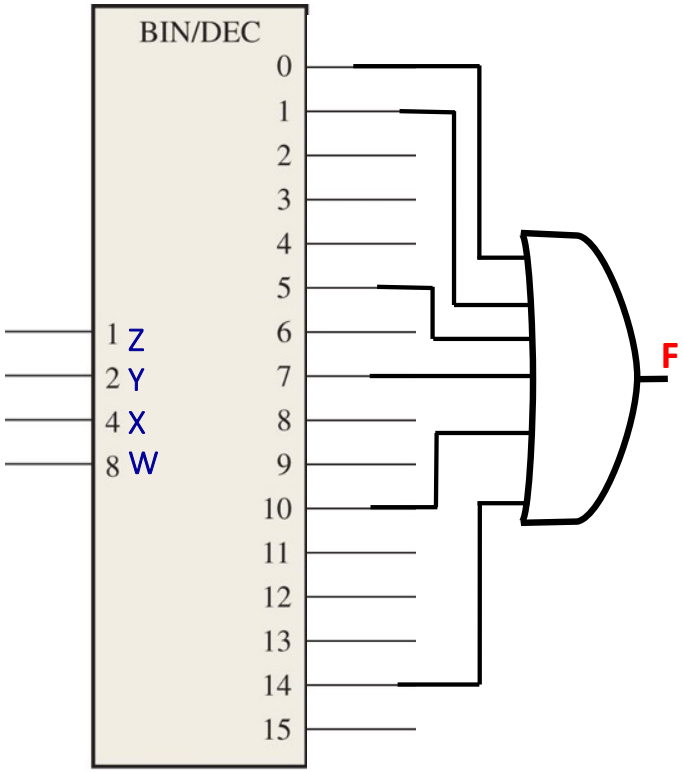
Parte alta **Parte baixa**
A5 A4 A3 A2 A1 A0



Exercícios

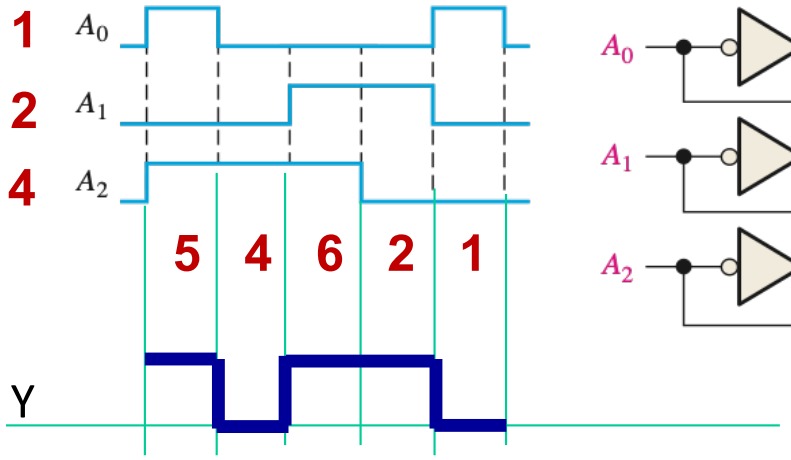
4. Dada a seguinte expressão Booleana $F(w, x, y, z) = \sum(0, 1, 5, 7, 10, 14)$

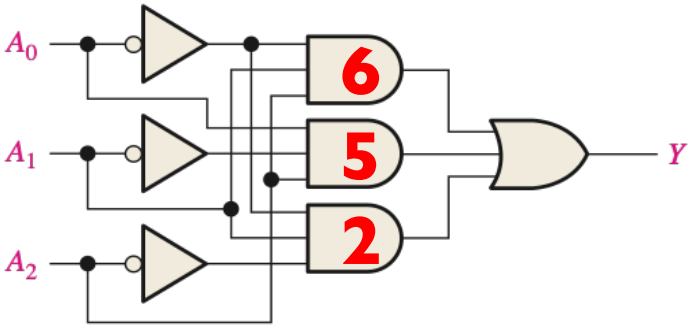
➔ Implemente esta expressão utilizando um decodificador 4x16.



Exercícios

5. Considere as formas de onda de entrada aplicadas à lógica de decodificação conforme indicado no circuito abaixo.
   - ✓ Esboce a forma de onda de saída em relação adequada às entradas.
   - ✓ Determine a função Y





A2. A1. A0’

A2.A1’.A0

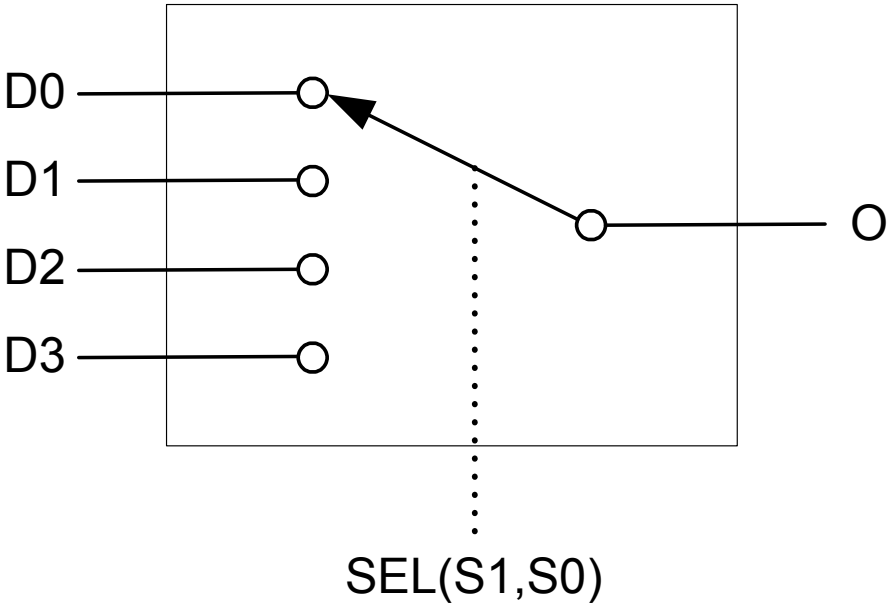
A2’.A1. A0’

Y = A2. A1. A0’ + A2.A1’.A0 + A2’.A1. A0’

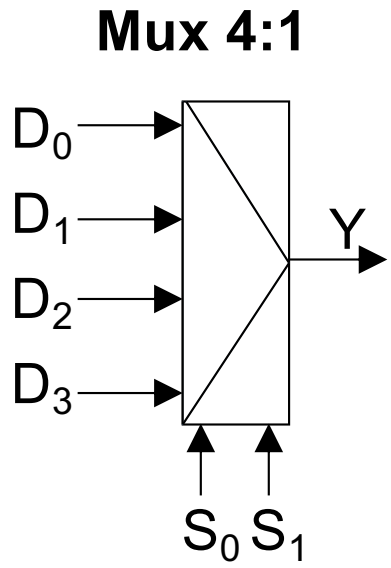
Y = $\sum$ (6 , 5 , 2)

(2) MULTIPLEXADOR

- O **multiplexador** é um circuito combinacional que permite, através da especificação dos sinais de seleção, encaminhar **uma** das N entradas de dados para a saída.



S1	S0	O
0	0	D0
0	1	D1
1	0	D2
1	1	D3

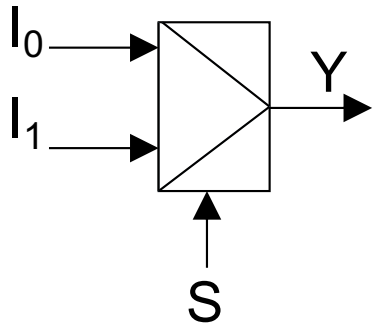


- **EQUIVALENTE EM SOFTWARE:** ***switch case / if then else***

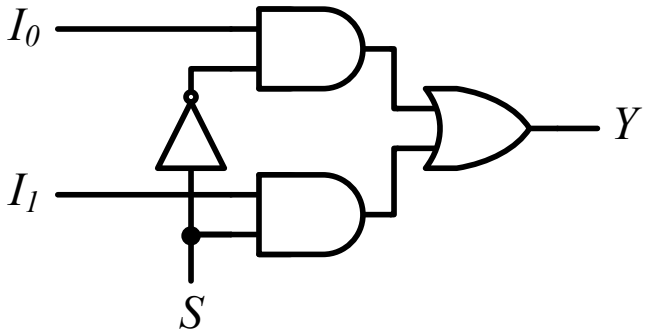
Multiplexadores

MUX 2:1

Símbolo:

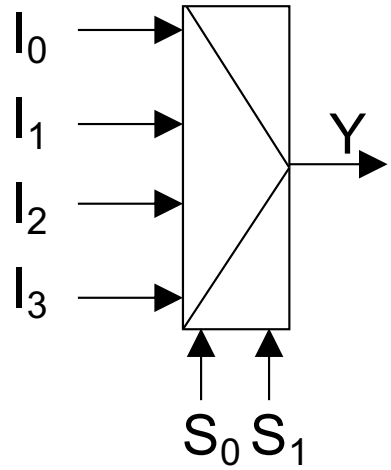


Estrutura Interna:

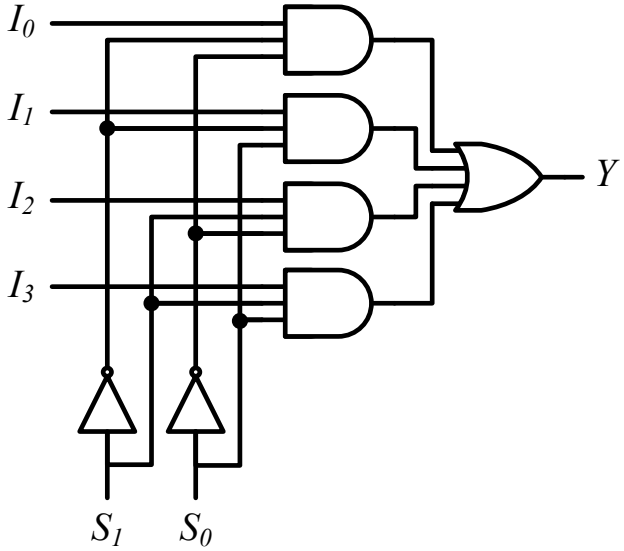


MUX 4:1

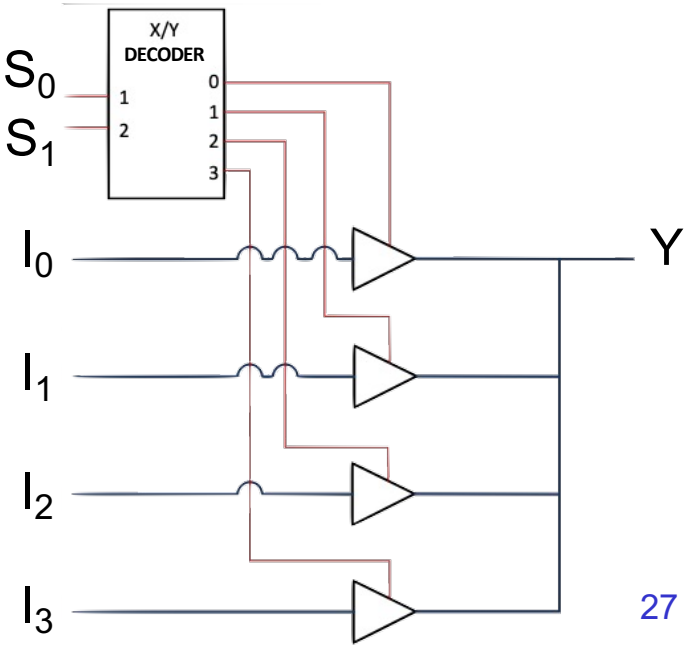
Símbolo:



Estrutura Interna:



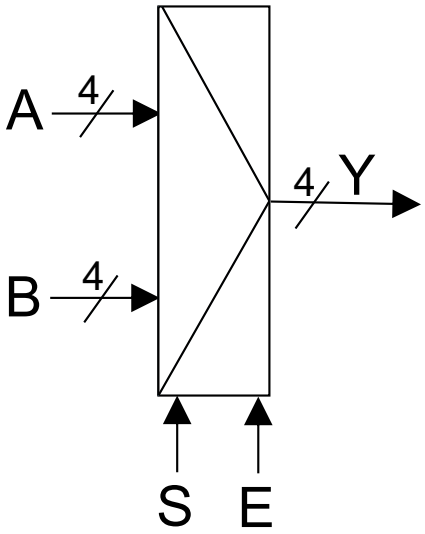
✓ Estrutura Interna Alternativa:



Multiplexadores (2x1 - 4 bits com enable)

MUX 2:1 de 4 bits com enable:

Símbolo:



Estrutura Interna:

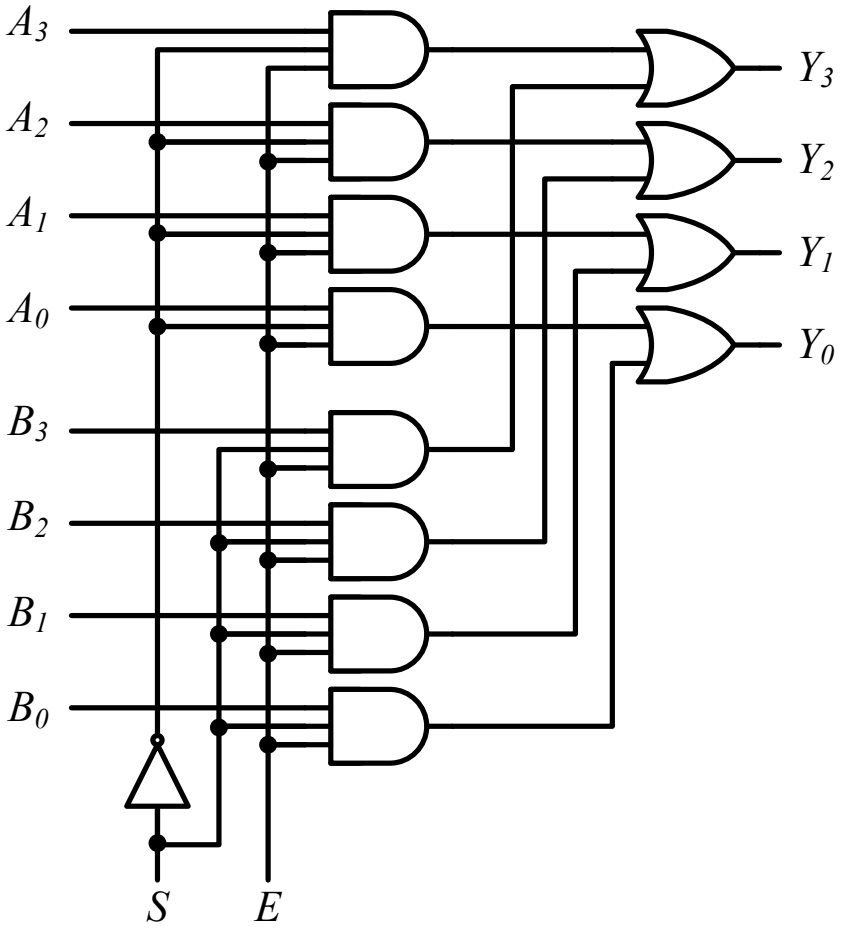
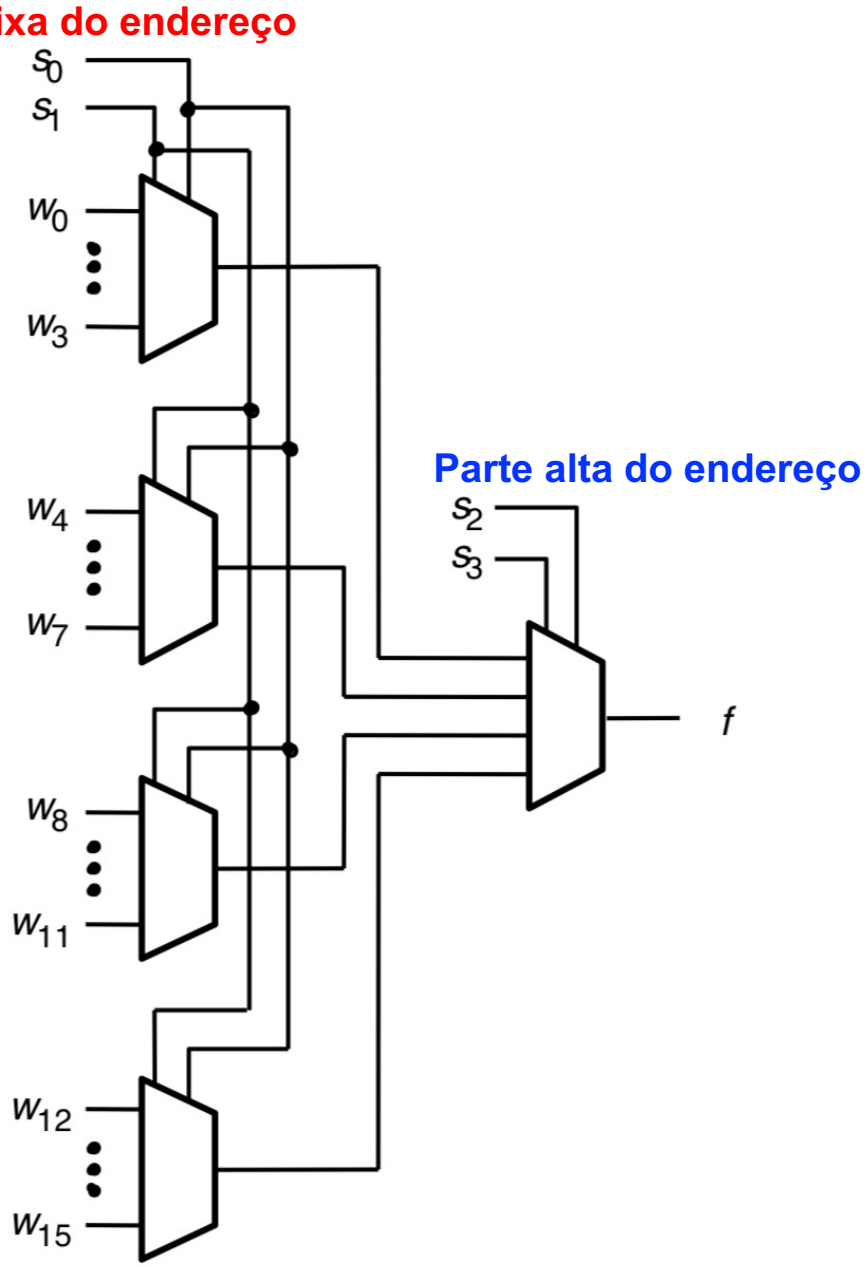


Tabela Verdade:

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

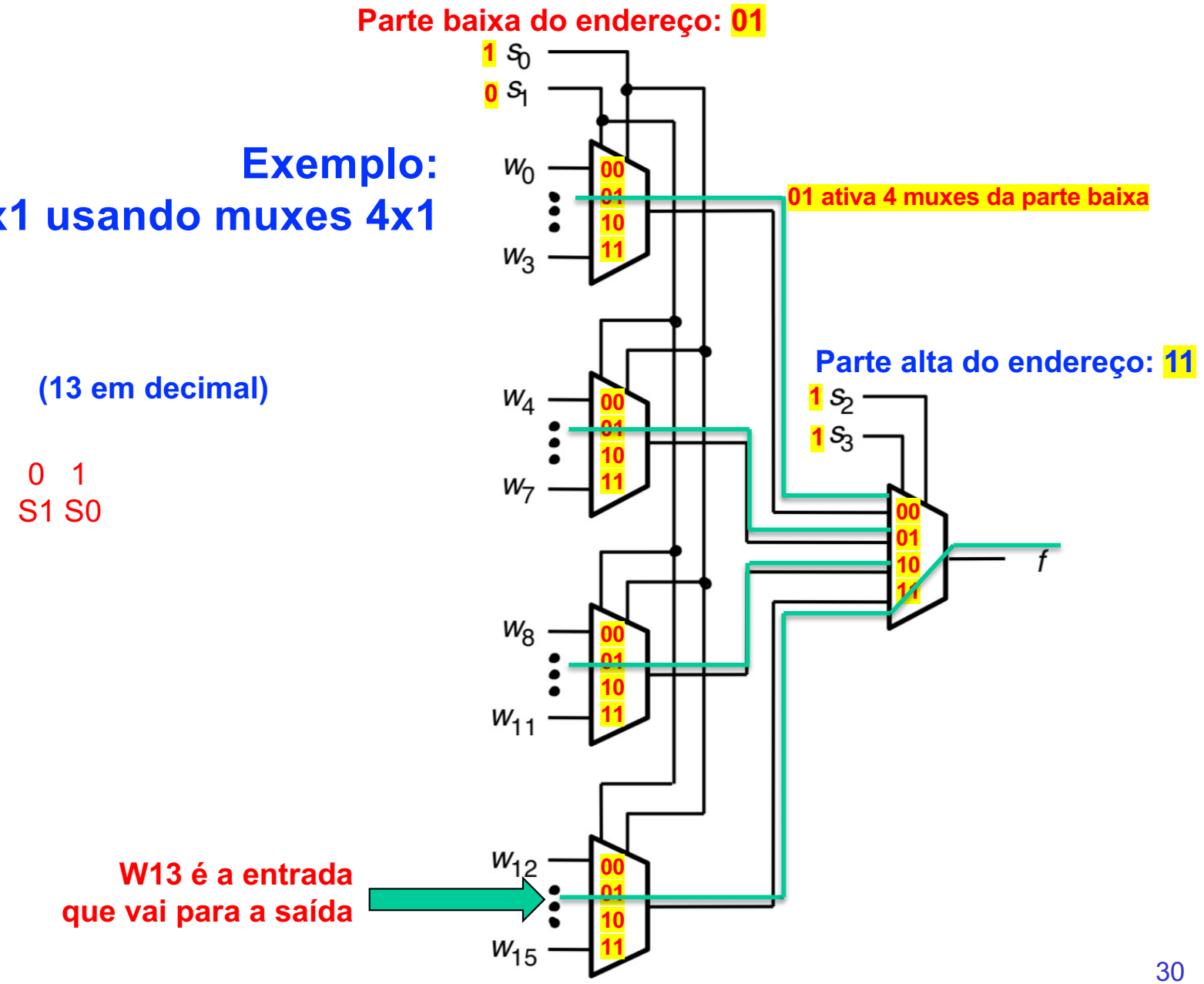
endereço:
S3 S2 S1 S0
Parte alta **Parte baixa**

Construindo multiplexadores maiores a partir de multiplexadores menores

Exemplo:
Mux 16 x1 usando muxes 4x1

Supor endereço 1101 (13 em decimal)

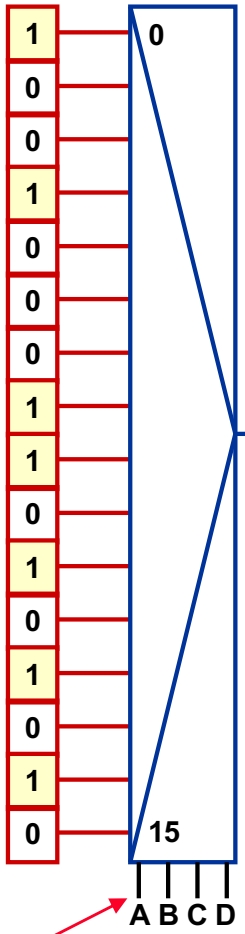
1 1 0 1
S3 S2 S1 S0



Multiplexador para gerar funções de *n* entradas (LUT)

LUT: Look-Up Tables (tabelas da busca)

A tabela verdade da função é armazenada em uma memória

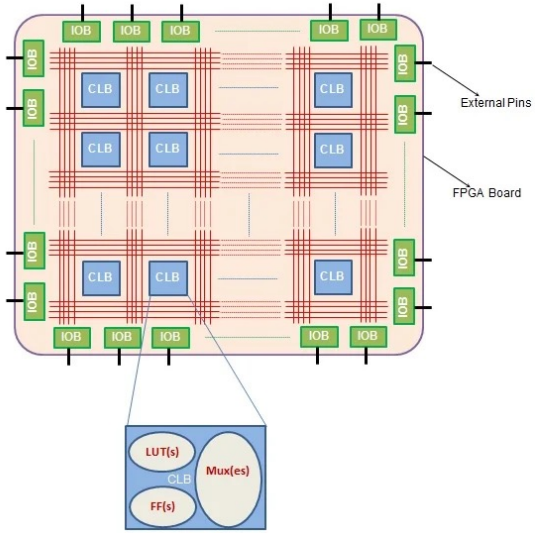


$$F(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.C.D + A.\overline{D}$$

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

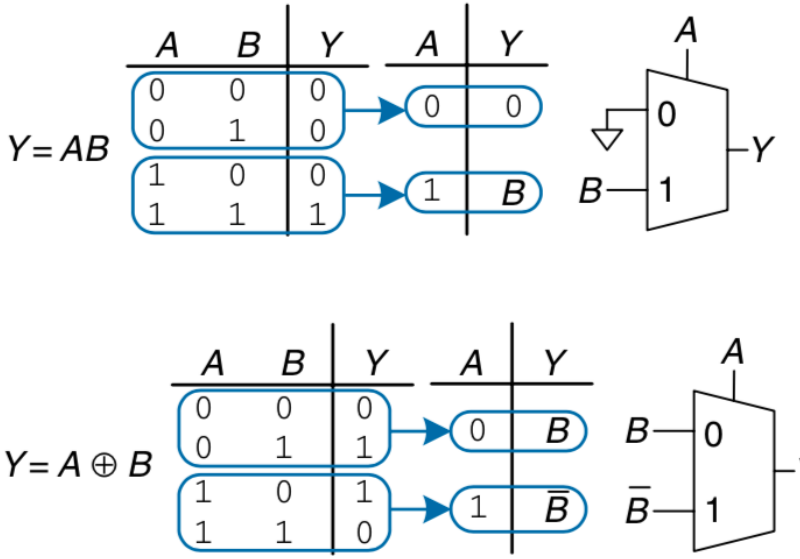
As entradas (variáveis Booleanas) controlam um multiplexador 2^n:1

Onde LUTs são utilizadas? FPGAs

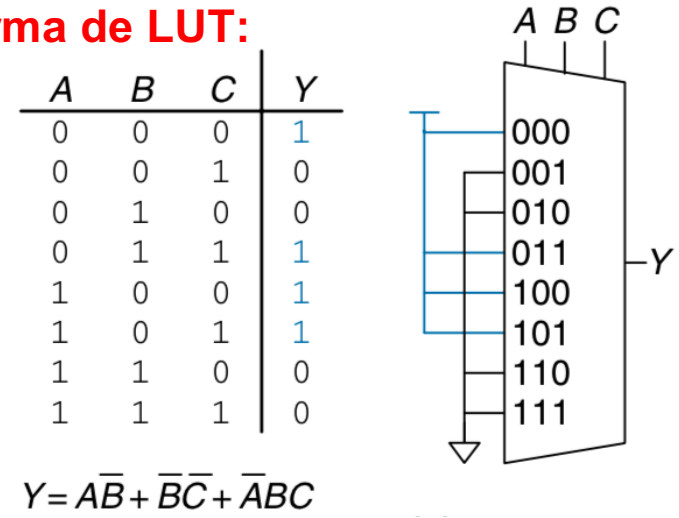


https://www.allaboutcircuits.com/technical-articles/purpose-and-internal-functionality-of-fpga-look-up-tables/

Multiplexador para gerar função booleanas



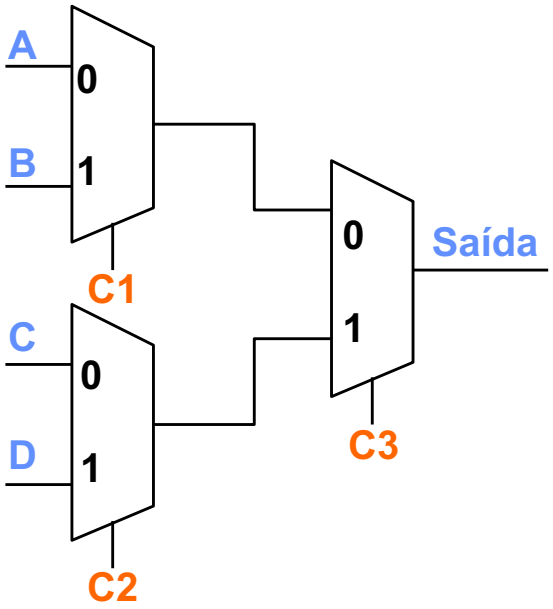
Na forma de LUT:

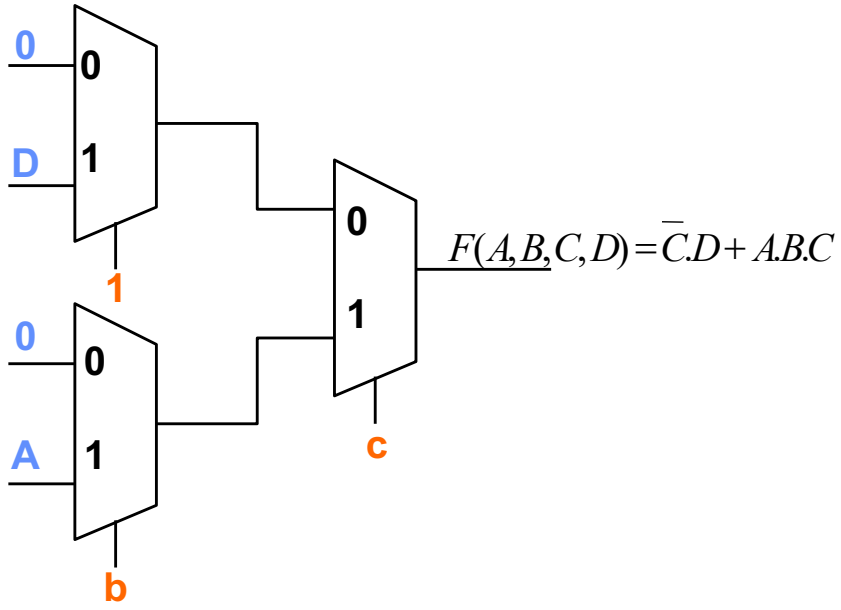


Digital Design and Computer Architecture
p. 86

Multiplexador para gerar funções booleanas

- estrutura conhecida como “gerador universal de funções lógicas” (Universal Logic Gate – ULG)
- não implementa todas as funções lógicas de *n* entradas
- funções lógicas mais complexas requerem diversos ULGs

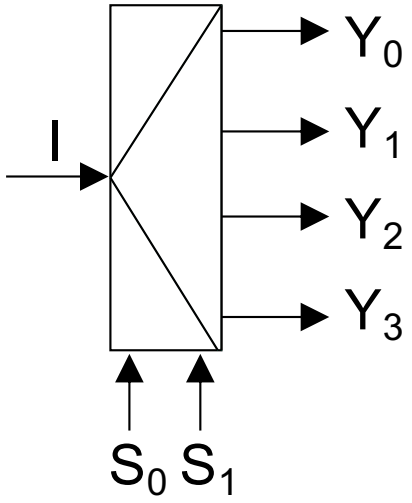


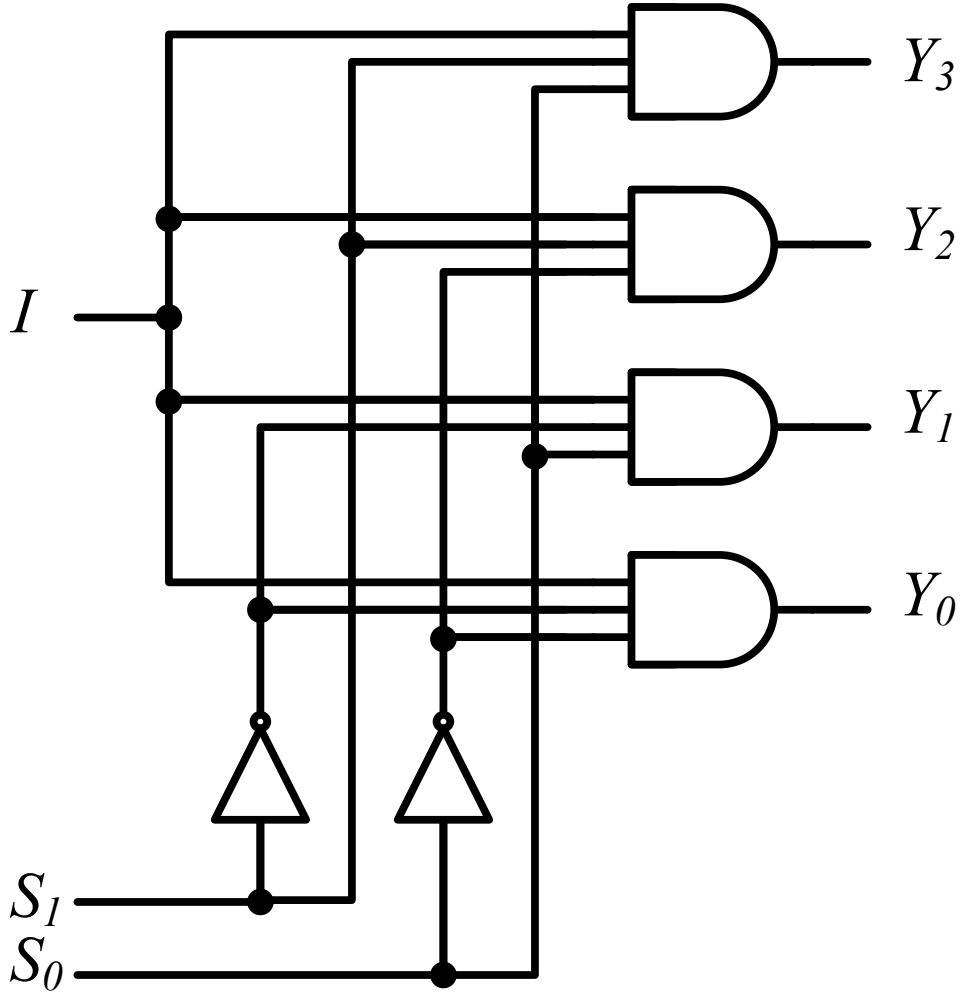


Demultiplexador

- O **demultiplexador** é um circuito combinacional que permite, através da especificação dos sinais de seleção, encaminhar a entrada para uma das N saídas.

- Exemplo de um DEMUX 1x4





Exemplo de aplicação com Mux/Demux

- Multiplexação do meio físico para enviar diferentes sinais

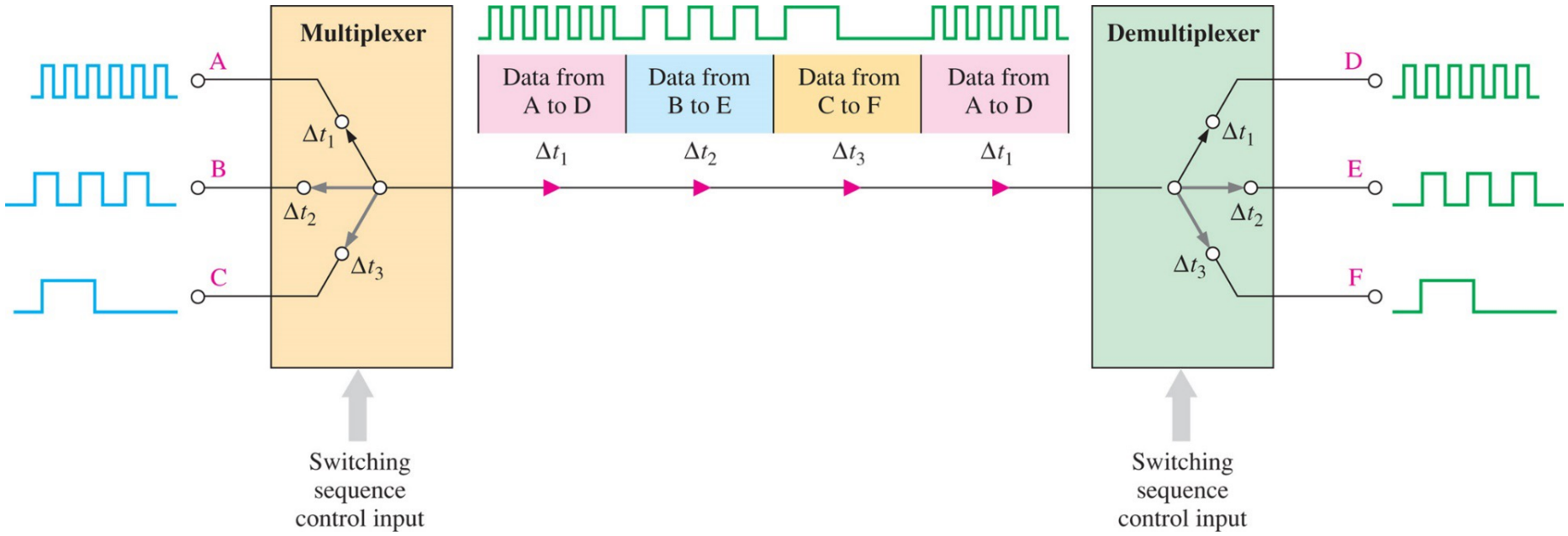
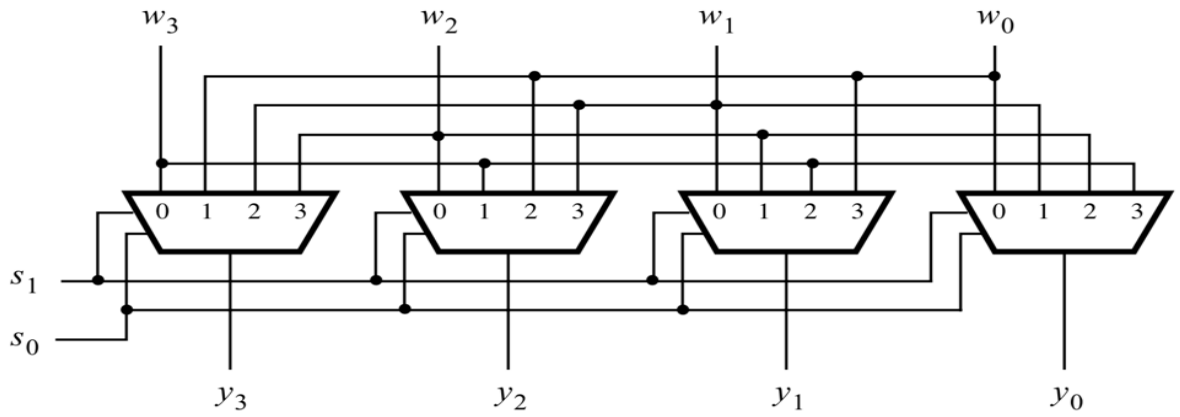


FIGURE 1-24 Illustration of a basic multiplexing/demultiplexing application.

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)

Exercícios

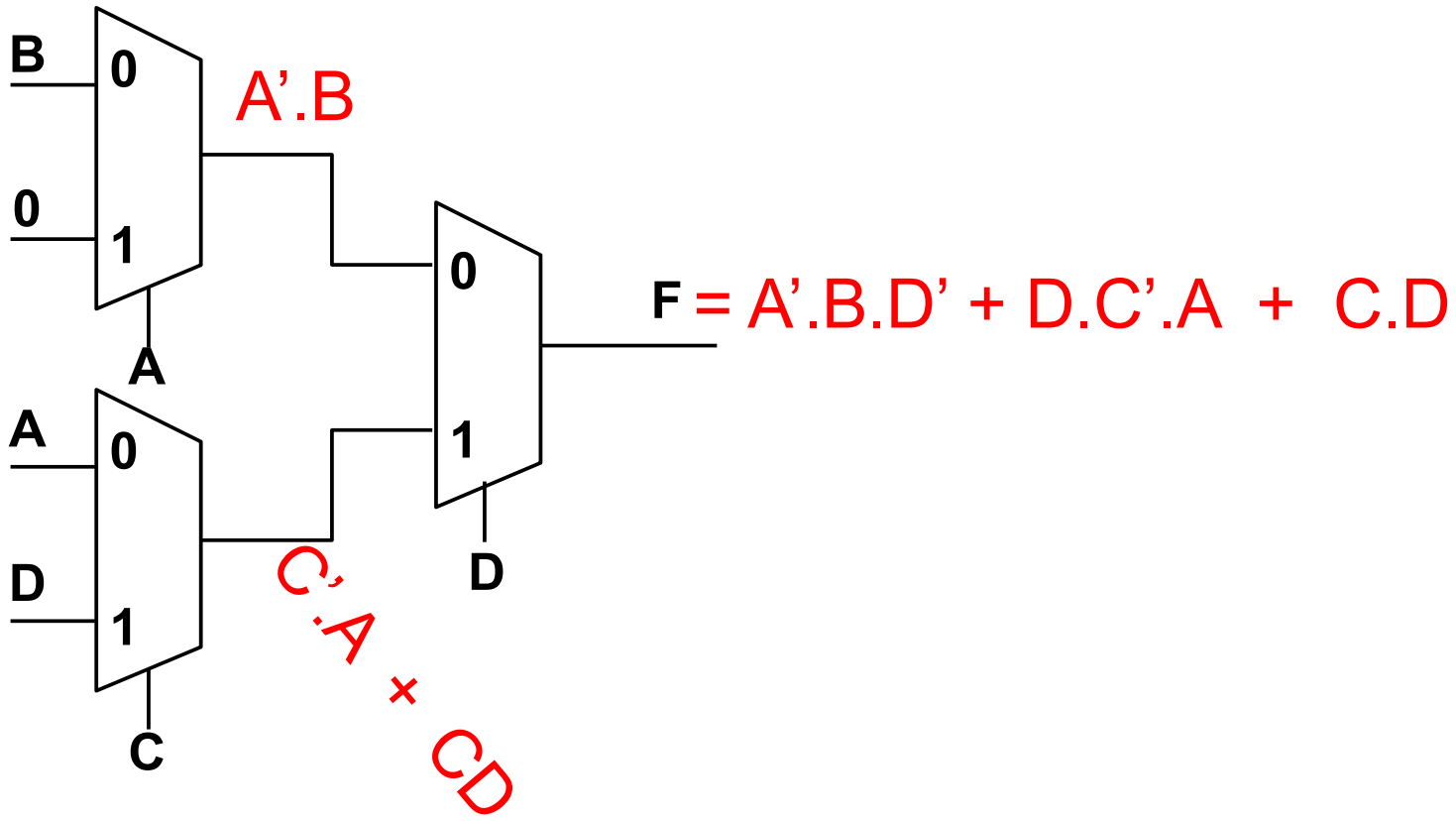
1. O circuito abaixo é composto por 4 multiplexadores. As entradas são: uma palavra de dados (w3-w0) e dois bits de controle (s1-s2). A saída é o vetor y (y3-y0). Preencha a tabela verdade correspondente a este circuito, e interprete o que este circuito realiza. No preenchimento da tabela verdade utilizar os valores w3/w2/w1/w0.



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

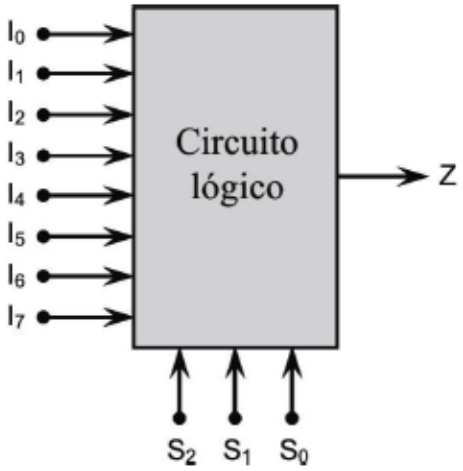
Exercícios

2. 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.



Exercícios

3. **(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

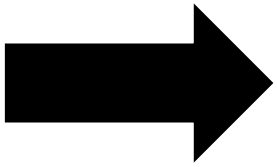
e) multiplexador

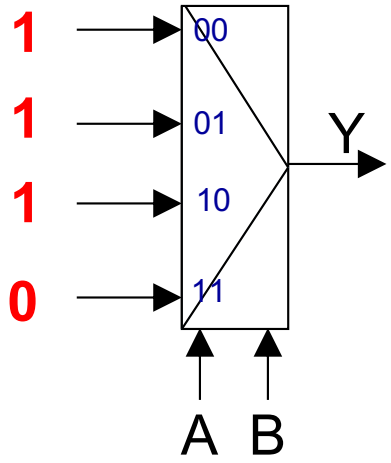
Exercícios

4. Utilizando um **MUX 4:1**, crie um circuito que realize a seguinte função combinacional de 2 variáveis:

F = A' + A.B'

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



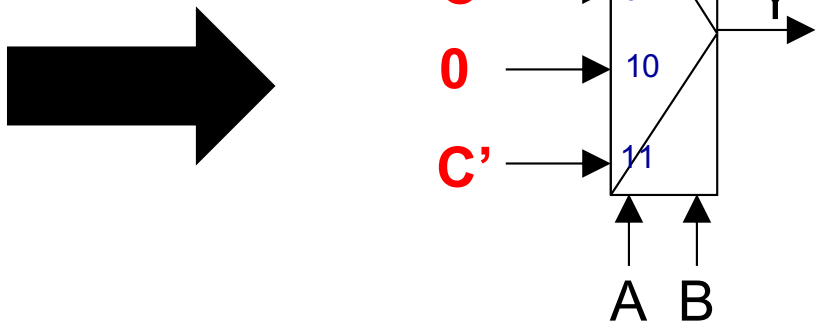


Exercícios

5. Utilizando um **MUX 4:1**, crie um circuito que realize a seguinte função combinacional de 2 variáveis:

F = A'.B' + A'.C + A.B.C'

A	B	C	F	
0	0	0	1	F = 1
0	0	1	1	
0	1	0	0	F = C
0	1	1	1	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = C'
1	1	1	0	

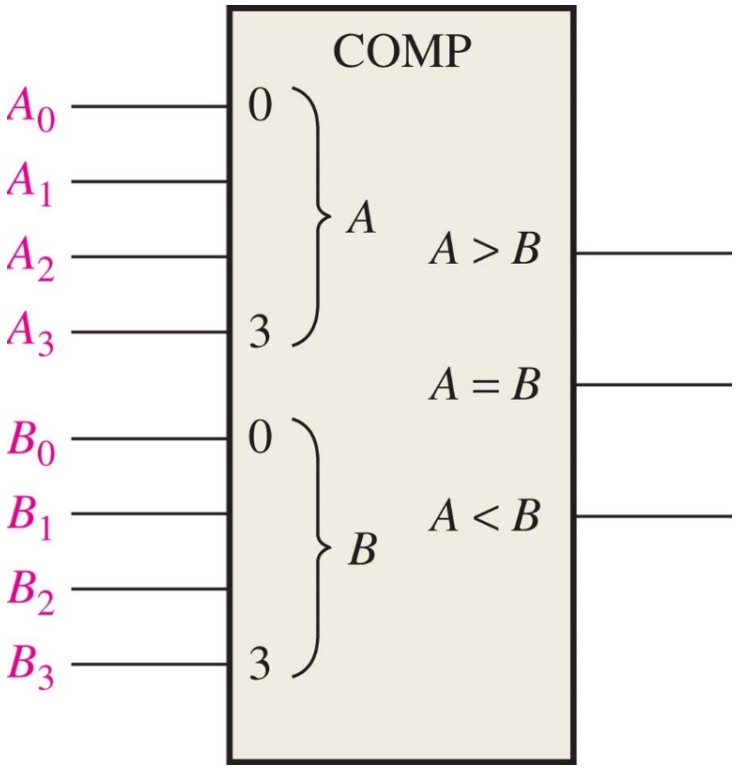


(3) COMPARADOR

– 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:

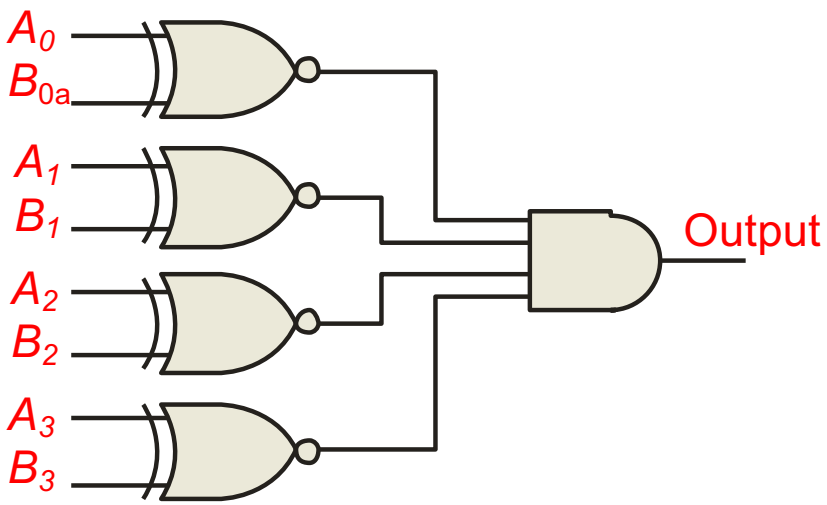
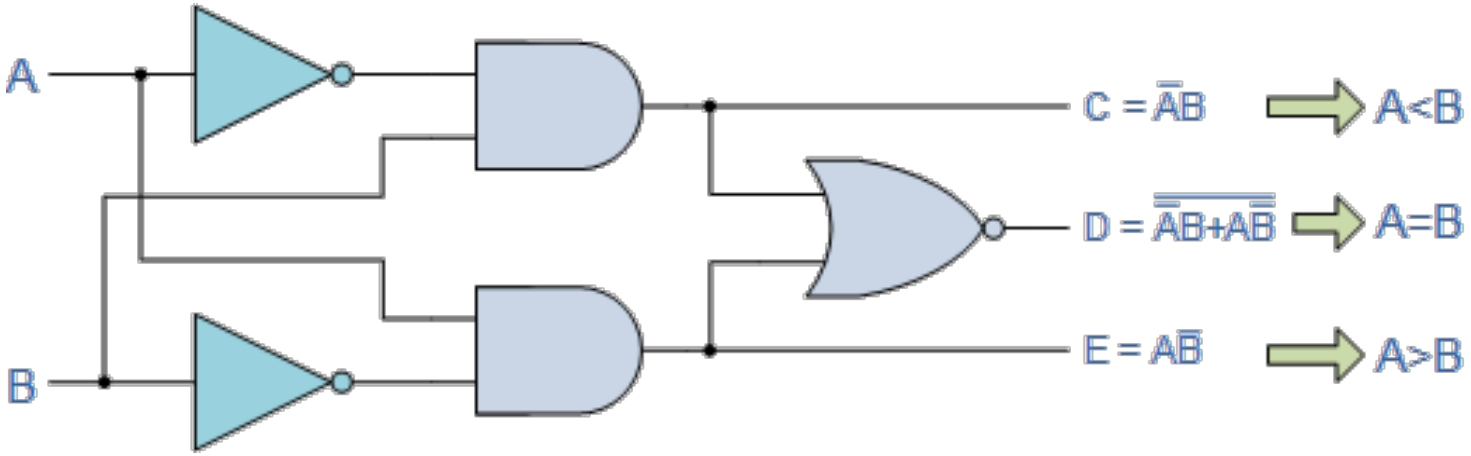


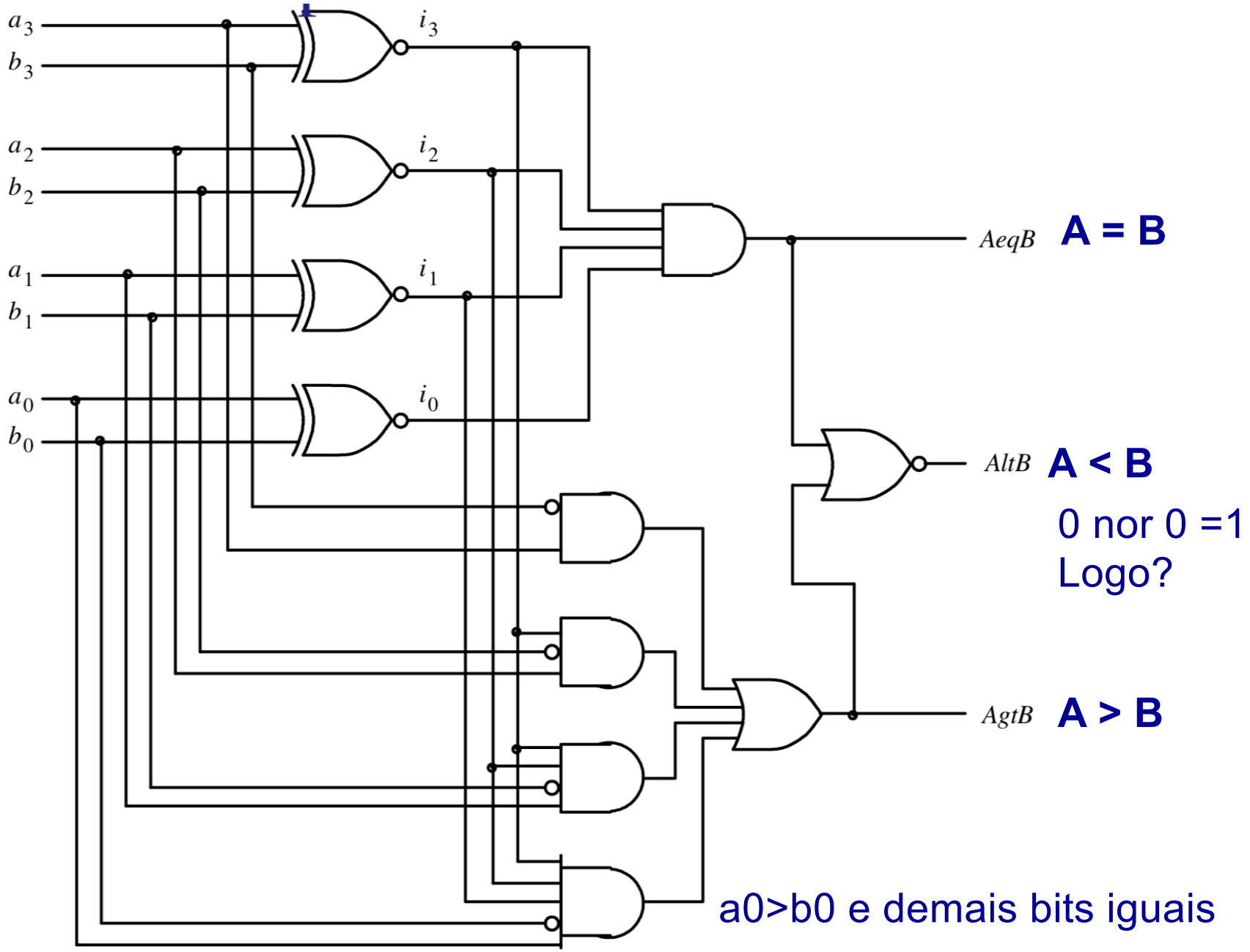
FIGURE 6-21 Logic symbol for a 4-bit comparator with inequality indication.

COMPARADOR DE 1 BIT

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



COMPARADOR DE DUAS PALAVRAS DE 4 BITS



(4) 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

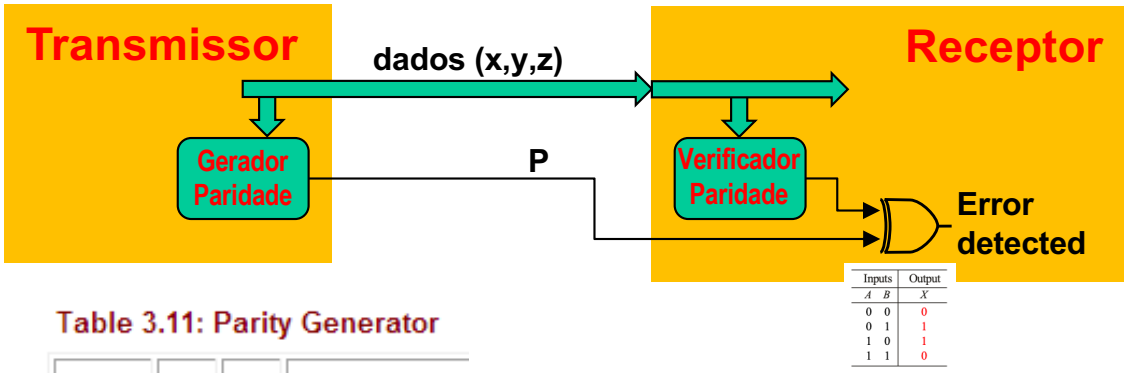
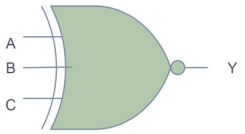


Table 3.11: Parity Generator

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

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



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

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

Em computadores:
ECC memory – Error-correcting code memory

Gerador de paridade e verificação de paridade

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

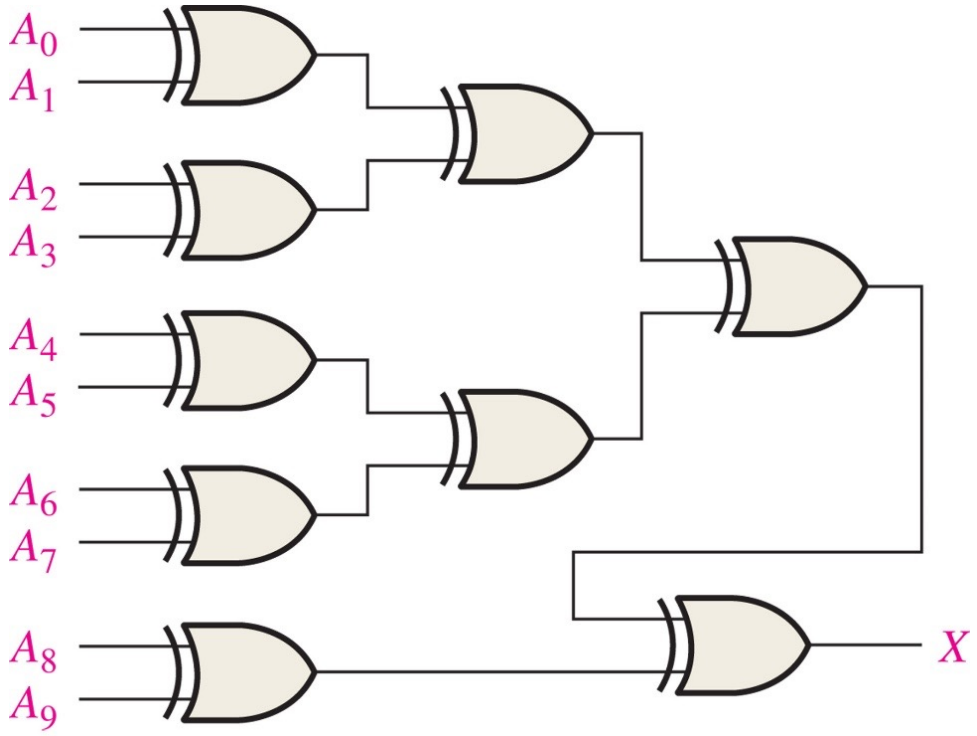


FIGURE 6-57

Uso de portas XOR

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

(5) PLA Matrizes Lógicas Programáveis

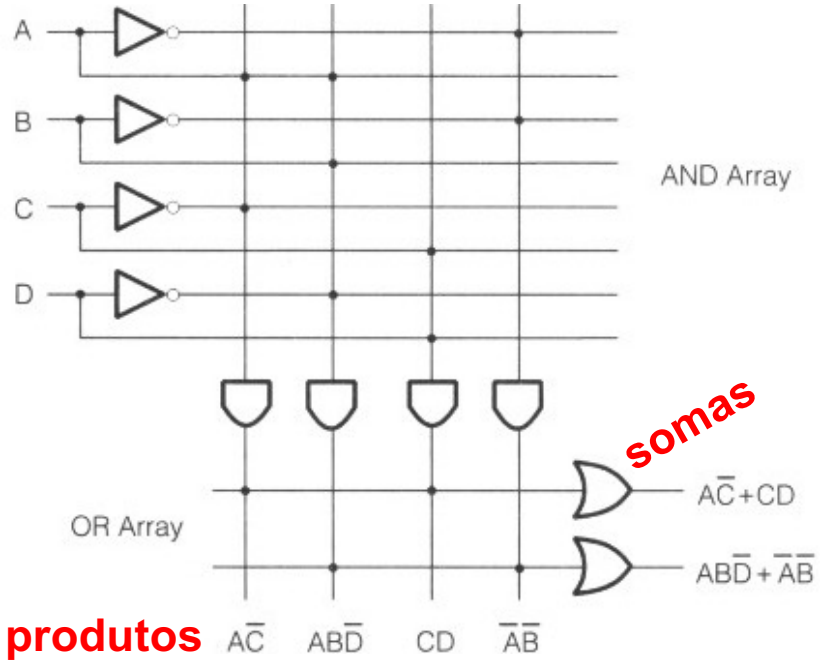
- Matrizes lógicas programáveis são tipos de circuitos que têm hardware pré-definido (parte estática) que implementa diversas funcionalidades conforme este for programado (parte dinâmica)

- 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)

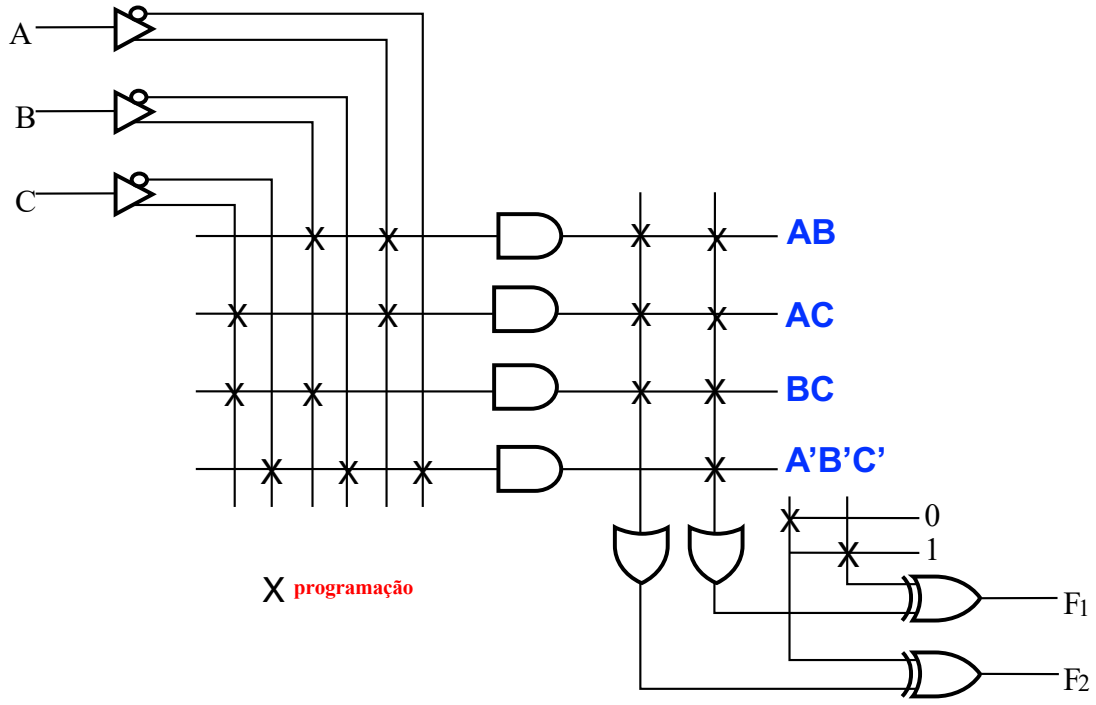
- O conjunto de funções a serem implementadas é primeiro transformado em somas de produto

- Uma vez que a inversão de saída está disponível, as funções podem ser implementadas com o seu complemento

Exemplo:

F1 = (AB+AC+BC+A'B'C')'

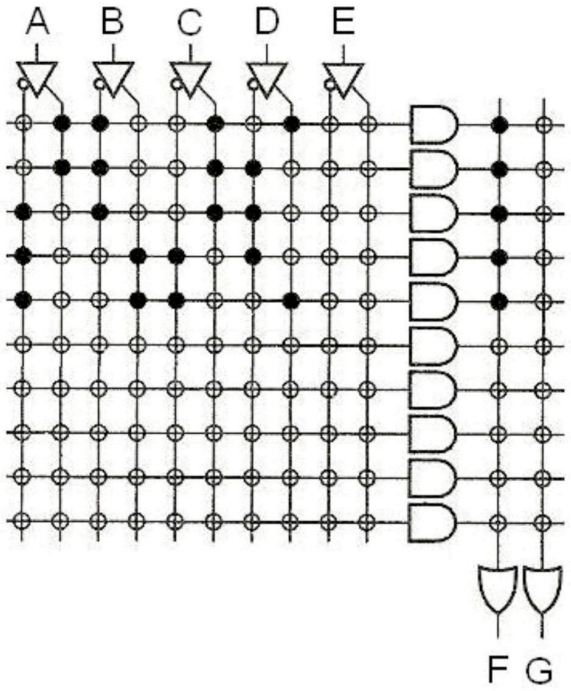
F2 = AB+AC+BC



Exercícios

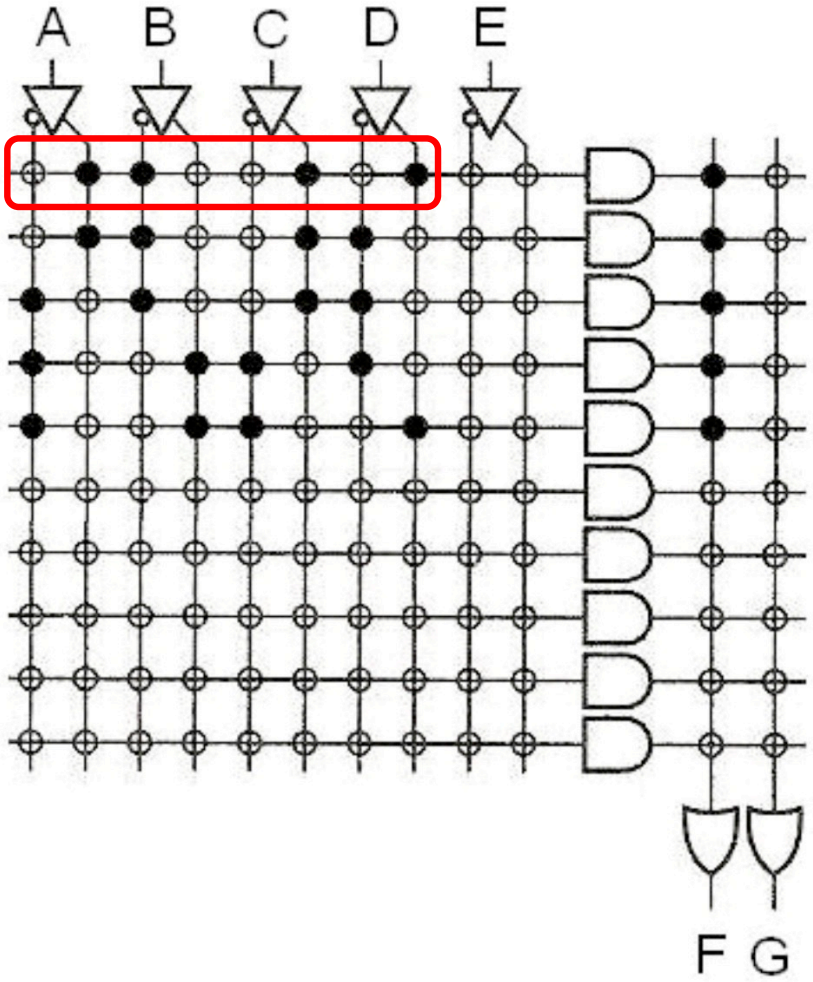
6. **(POSCOMP 2010, Questão 41)** Considere o circuito digital apresentado no diagrama a seguir. Ressalte-se que, por convenção, chaves representadas por círculos escuros representam conexões fechadas e chaves representadas por círculos vazados representam conexões abertas.

Determine a função F.



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

$D+D'=1$ $A+A'=1$ $D+D'=1$



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

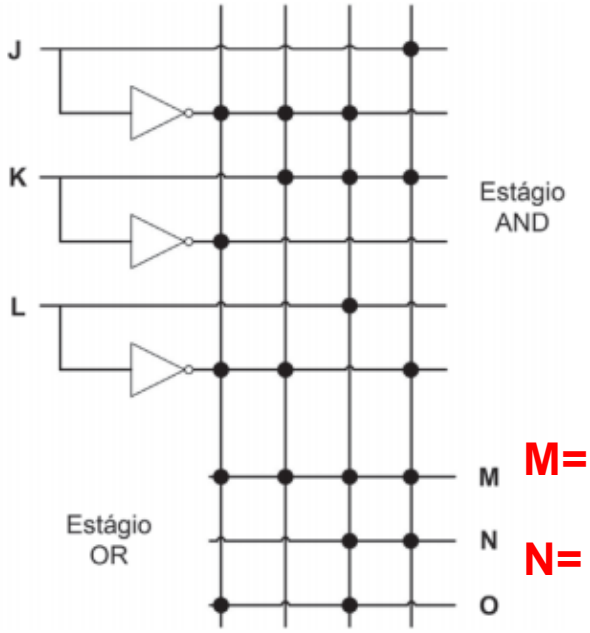
Exercícios

7. 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.



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

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

(6) Memória ROM

- Memória de apenas leitura – na prática um vetor de constantes

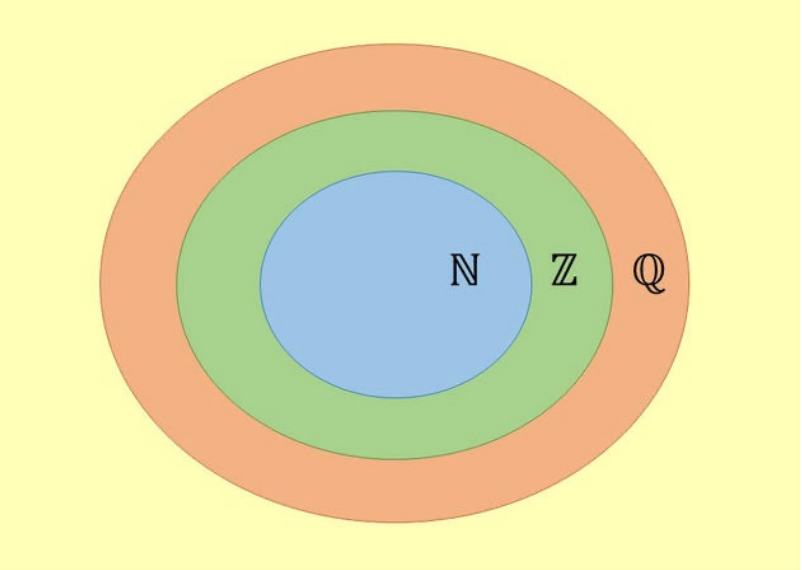
- Utilizada em hardware para constantes em determinado circuito (por exemplo: em código para boot em um processador)

REVISÃO DE NUMERAÇÃO BINÁRIA

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

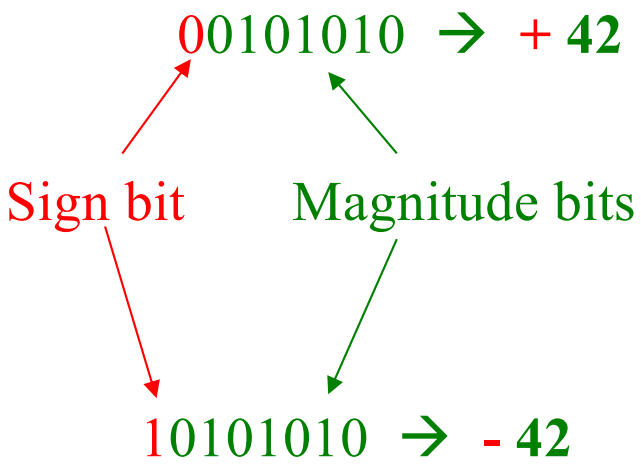
INTEIROS: Z = {..., –4, –3, –2, –1, 0, 1, 2, 3, 4, ...} (N ⊂ Z)

RACIONAIS: Q -> todos os números que podem ser escritos na forma p/q, sendo p e q números inteiros e q≠0.



SM – sinal magnitude

→ o bit mais significativo é o sinal



Complemento de 2 → 2's (representa inteiros - Z)

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

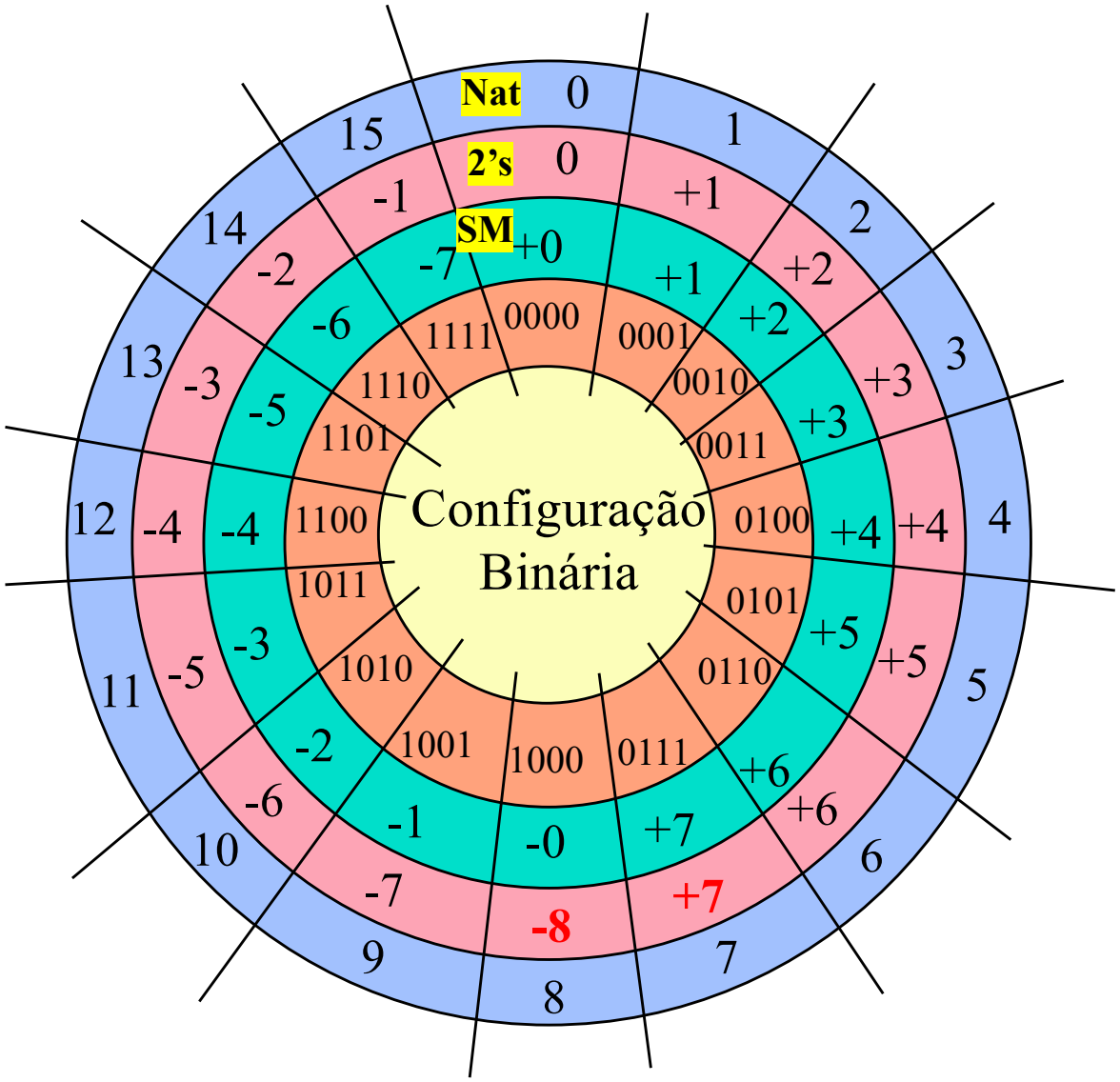
Um número binário de *n* bits poder ser interpretado de várias formas:

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

- 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

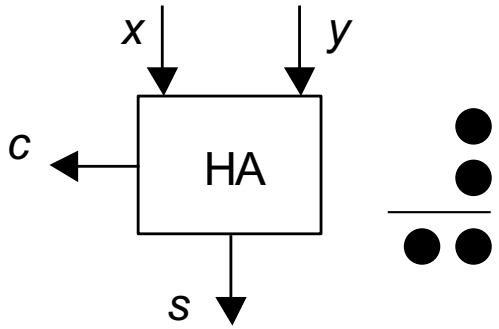


- SM é fácil de entender e separa sinal de valor
  $[-(2^{n-1}-1) .. 2^{n-1}-1]$
  notar que o zero possui duas representações (+0 e -0)
- **2's facilita soma/sub**
  $[-(2^{n-1}).. 2^{n-1}-1]$
- Naturais (*unsigned*)
  $[0 .. 2^{n}-1]$

(7) Soma e subtração

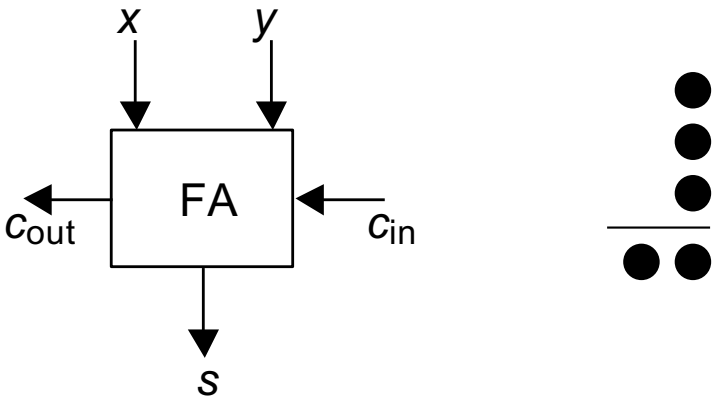
Somadores de 1 bit

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

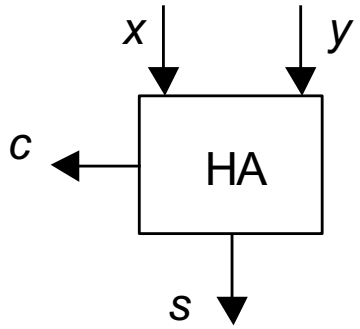
Inputs			Outputs	
x	*y*	c_{in}	c_{out}	*s*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

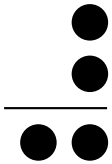


Full-adder (FA): Truth table and block diagram

Half-adder

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





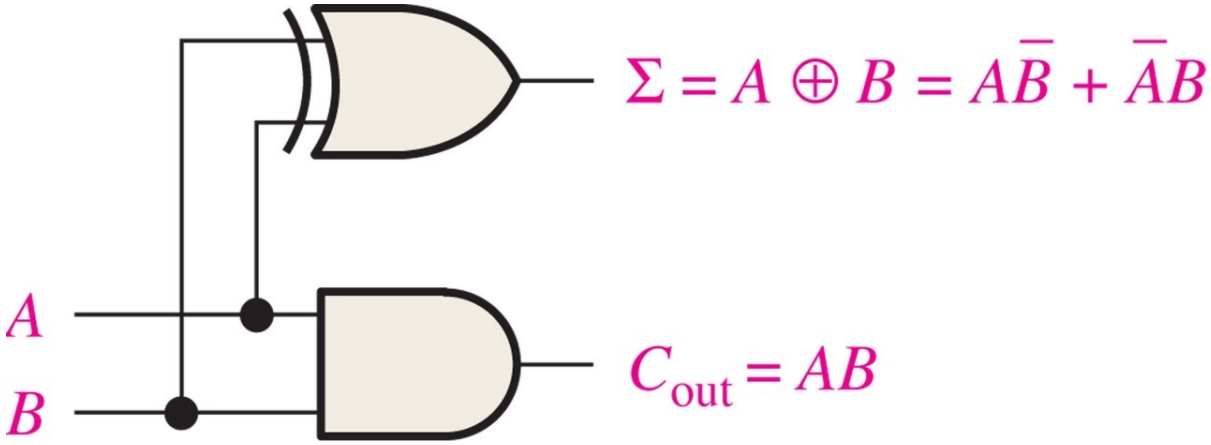
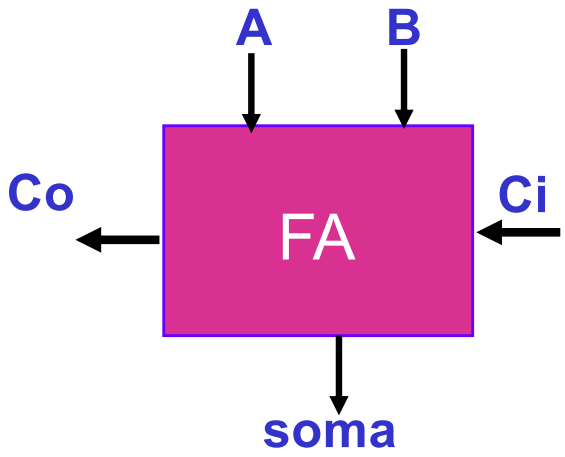


FIGURE 6-2 Half-adder logic diagram.

Somador Completo - FA



Soma:

a\bc	00	01	11	10
0		1		1
1	1		1	

Co:

a\bc	00	01	11	10
0			1	
1		1	1	1

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

$$S = A.\overline{B}.\overline{C_i} + \overline{A}.B.\overline{C_i} + \overline{A}.\overline{B}.C_i + A.B.C_i$$

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

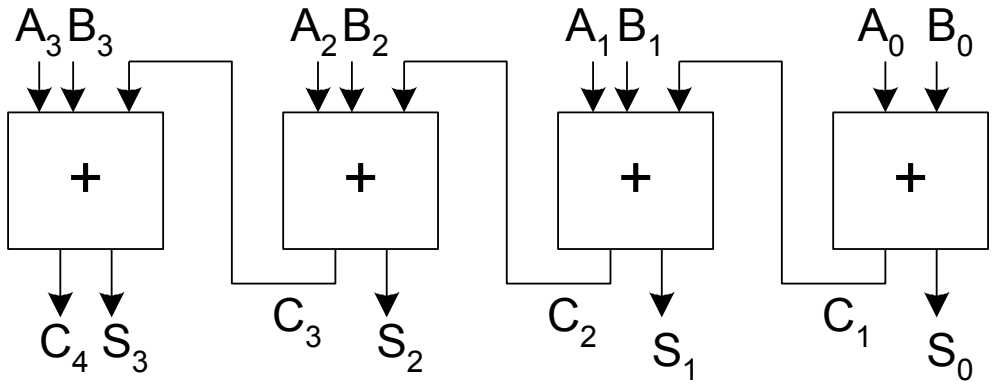
$$C_o = A.B + B.C_i + A.C_i$$

Somador ”ripple” (cascata)

- ✓ Soma 1 bit de cada vez:
- ✓ Exemplo: Somador de 2 números de 4 bits cada

C4	C3	C2	C1	
	A3	A2	A1	A0
+	B3	B2	B1	B0
C4	S3	S2	S1	S0

		0	1	1	0 ← Carry
	7		0	1	1 1
+	2	+	0	0	1 0
	9		1	0	0 1

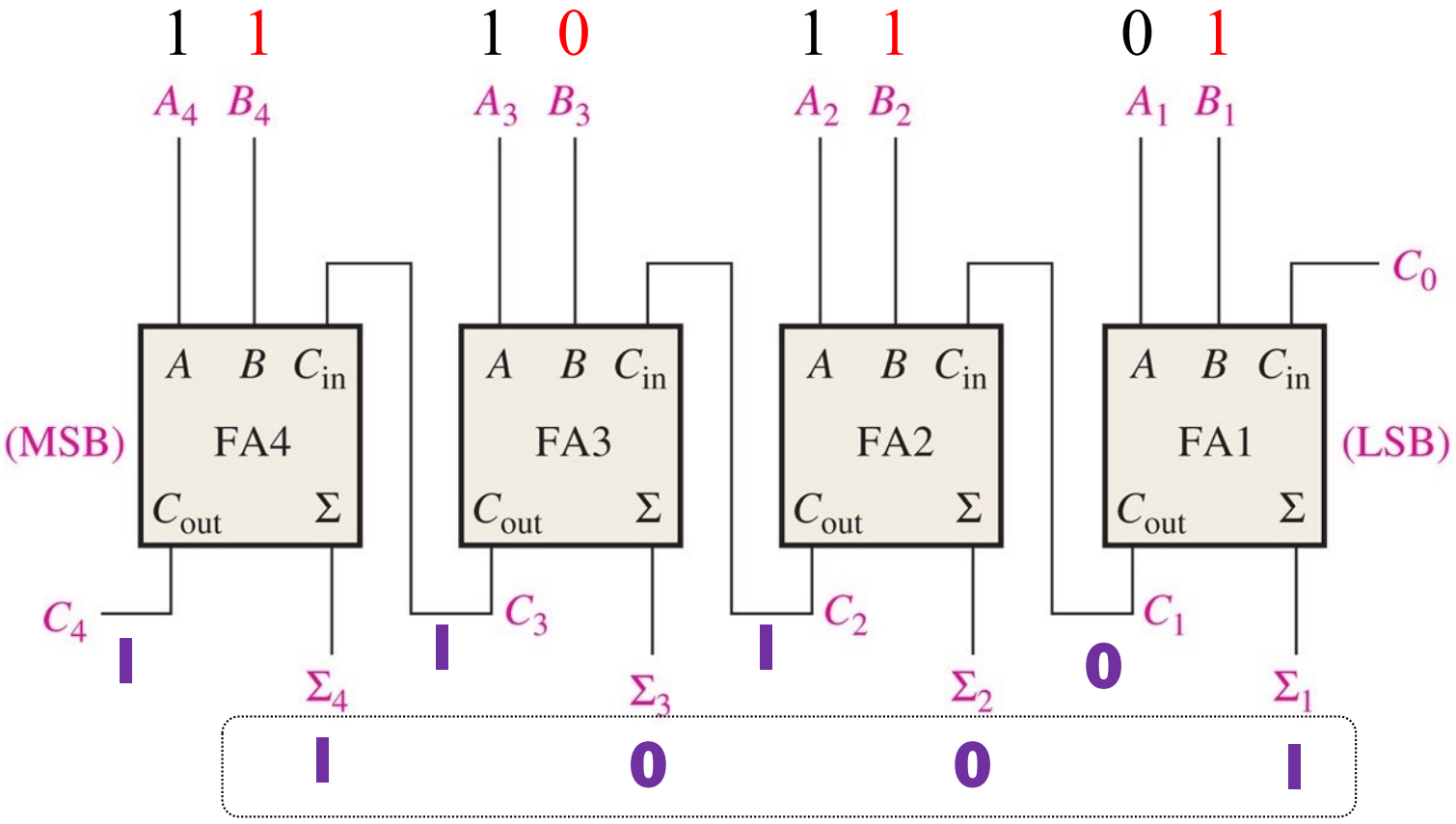


Somador "ripple"

Exemplo: E_{16} + B_{16}

E = 1110 = 14/-2

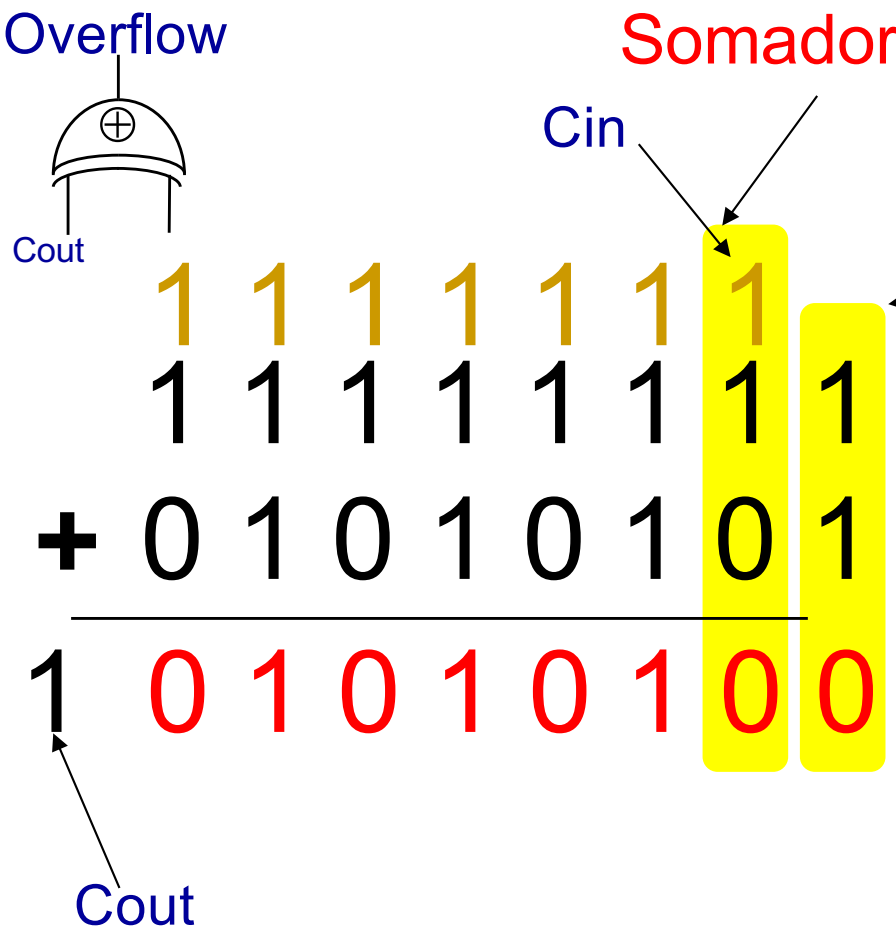
B = 1011 = 11/ -5



9 / -7

➔ **C_{out} indica transbordo para inteiros positivos (N)**

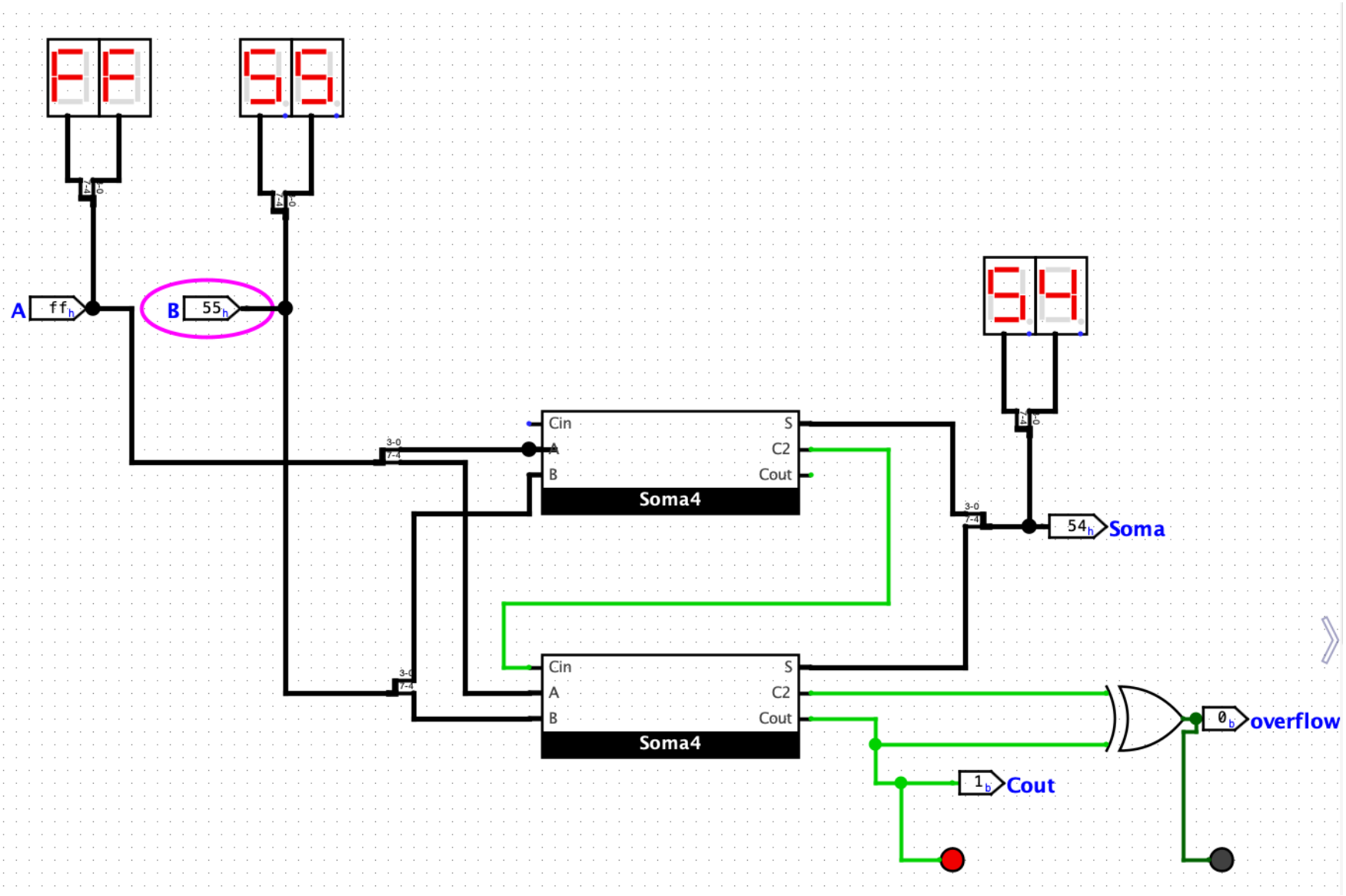
Somador ripple




```
  1
  F F
+ 5 5
-----
1 5 4
```

Hexadecimal: FF + 55 = (1) 54
Unsigned: 255 + 85 = 84 (**errado**, Cout=1)
2's comp: -1 + 85 = 84 (**certo** – Overflow=0)

Somador ripple



Subtração

Subtração de números, naturais ou inteiros:

- ✓ A subtração de dois números binários com sinal, representados em complemento de 2, é obtida do seguinte modo:
  1. Forma-se o complemento de 2 do subtraendo
  2. Soma-se ao minuendo

Em decimal: 9's+1

```
  83        83
- 28      + 71
----      + 1
  55      ----
            55
```

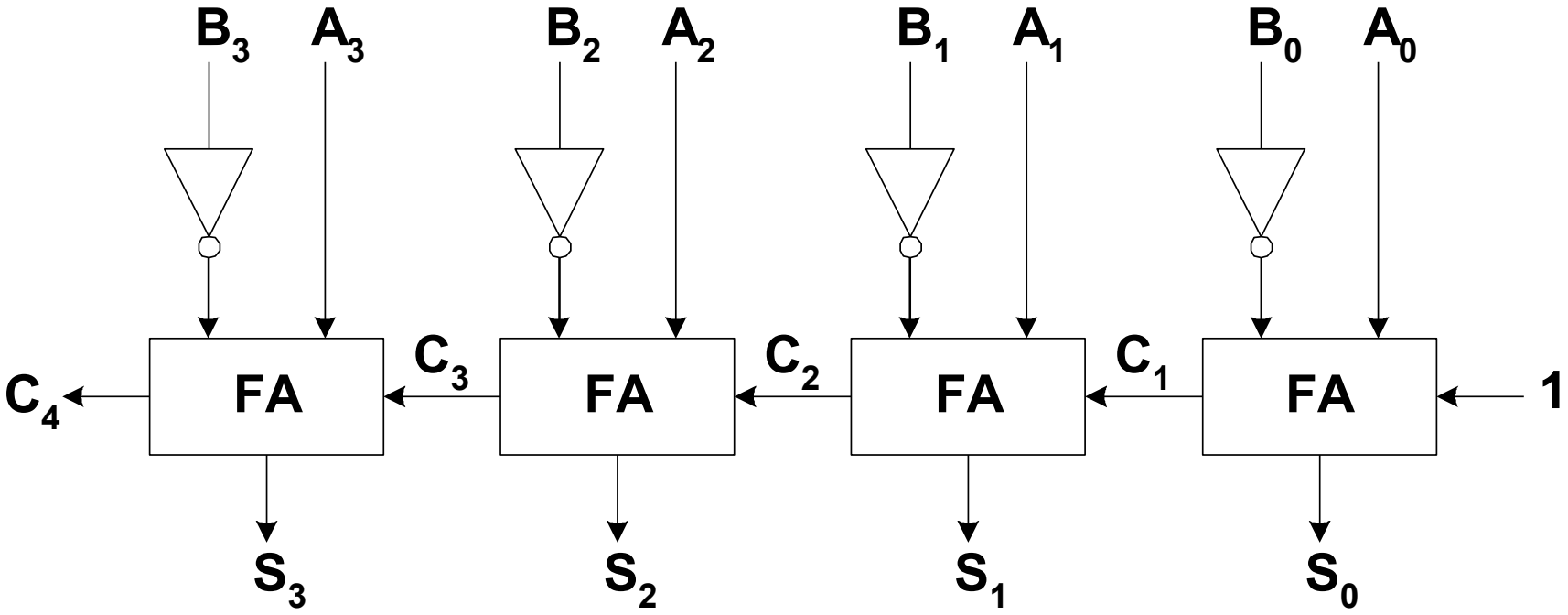
- ✓ Exemplo:

```
   4      0 1 0 0          4      0 1 0 0          0 1 0 0
 - 3    - 0 0 1 1      +(-3)  + 1 1 0 1      + 1 1 0 0
---     ---------      -----  ---------      ---------
   1                       1    0 0 0 1        0 0 0 1
```

Subtração

Subtração de números, naturais ou inteiros:

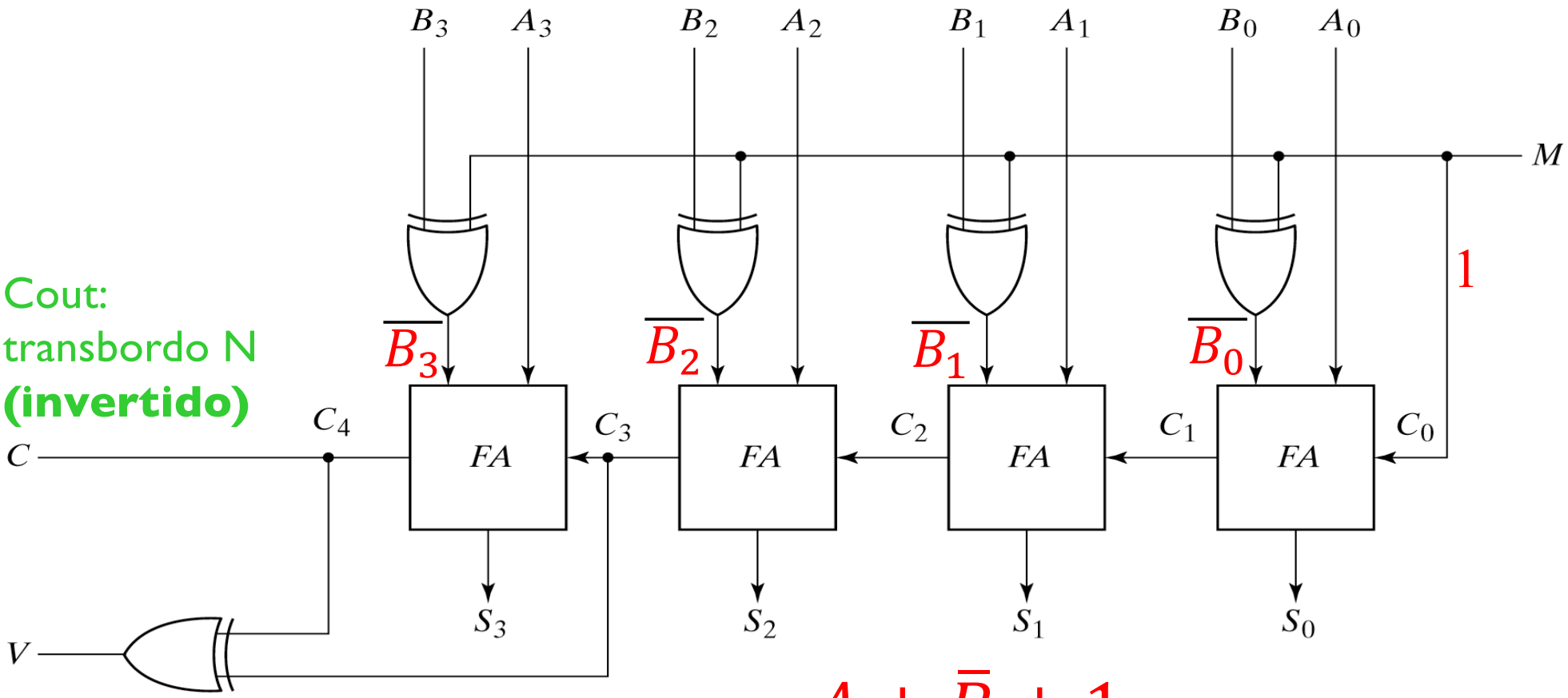
- ✓ Complemento de 2 = (Complemento de 1) + 1
  - O complemento de 1 é realizado invertendo todos os bits do subtraendo
  - A adição de 1 é efetuada pondo o *carry* inicial a 1



$$A - B = A + \bar{B} + 1$$

Somador / subtrator

M = 1→ subtrator / M = 0→ somador



Cout: transbordo N **(invertido)**

Overflow: transbordo em 2's

$$A + \overline{B} + 1$$

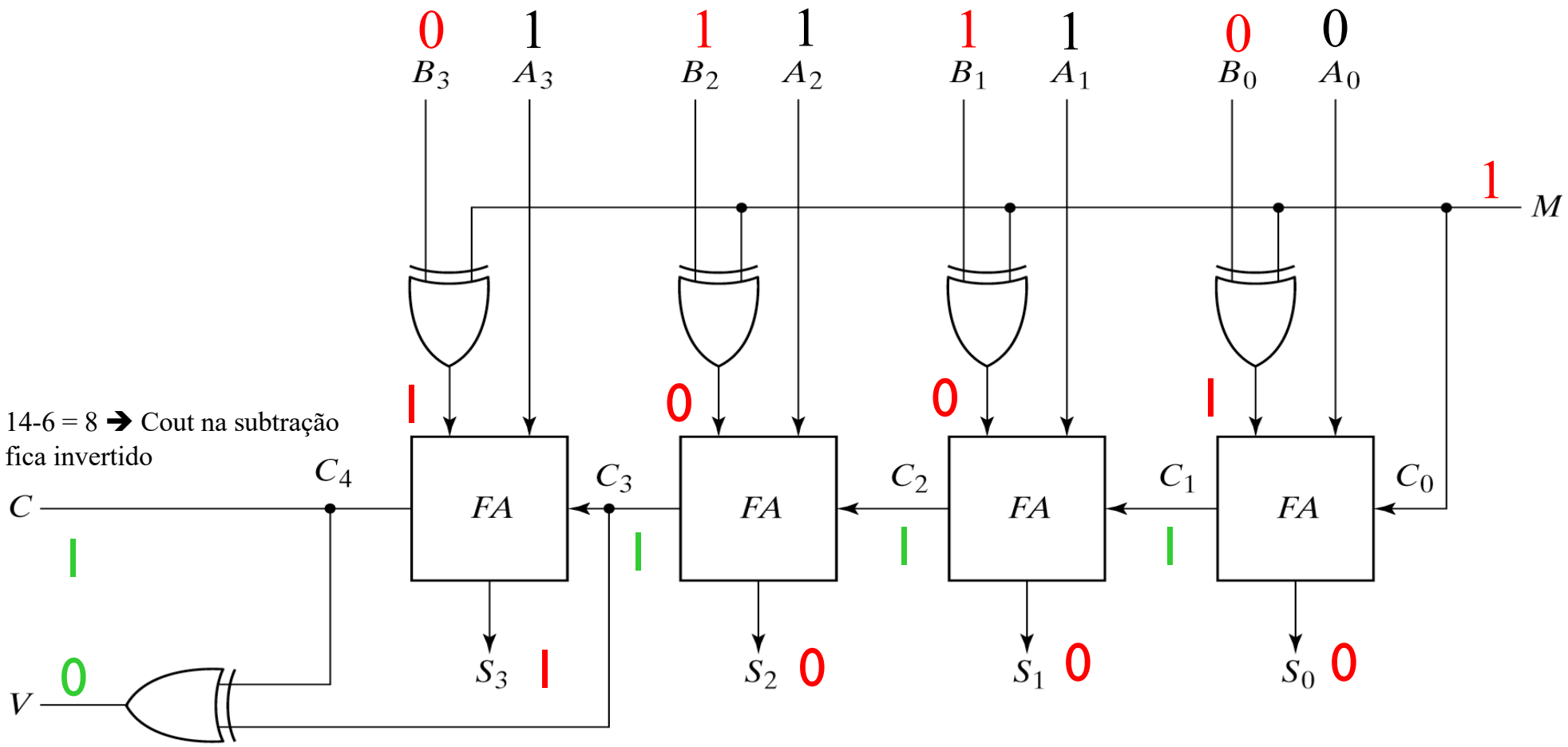
Somador / subtrator

M = 1→ subtrator / M = 0→ somador

E = 1110 (14/-2)

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

6 = 0110

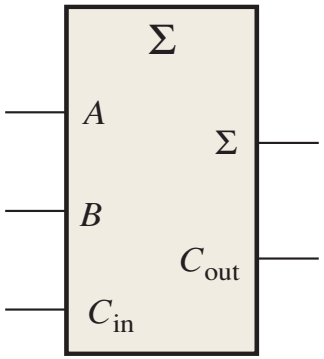


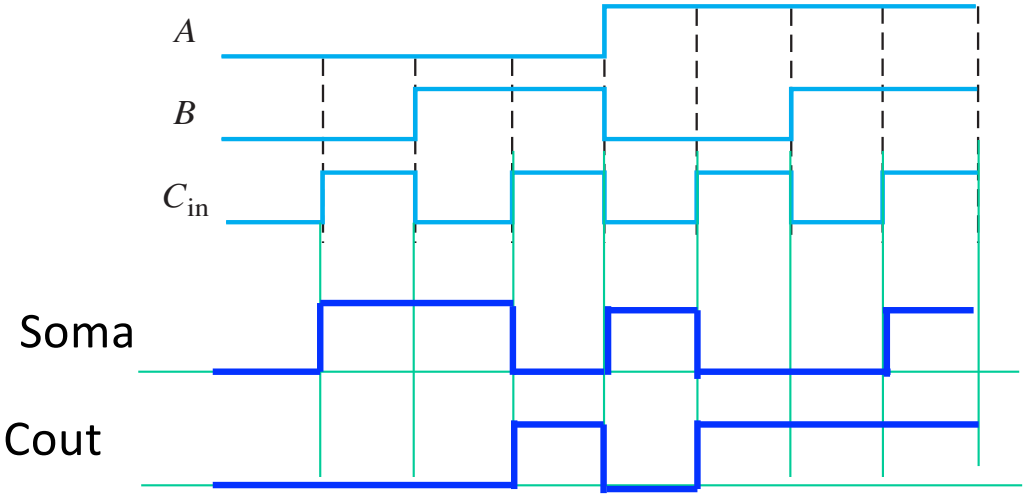
-2-6 = -8 → sem overflow

4 bits: $[0..15]_N$ e $[-8..7]_I$

Exercícios (1/8)

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





Exercícios (2/8)

Determine as saídas Soma a Cout para os estímulos fornecidos, considerando **apenas** o circuito **somador**.

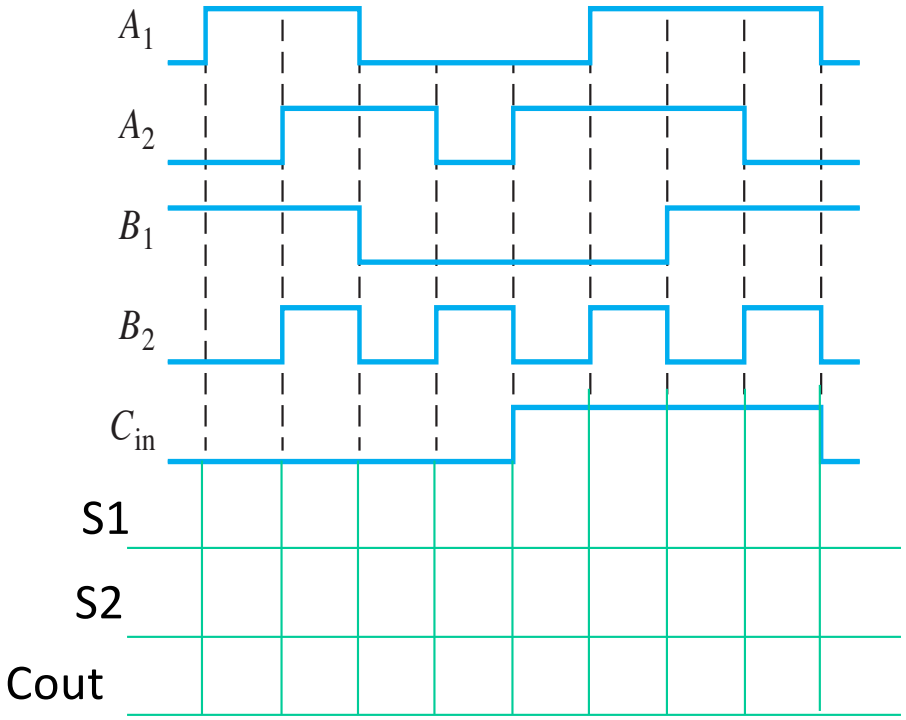
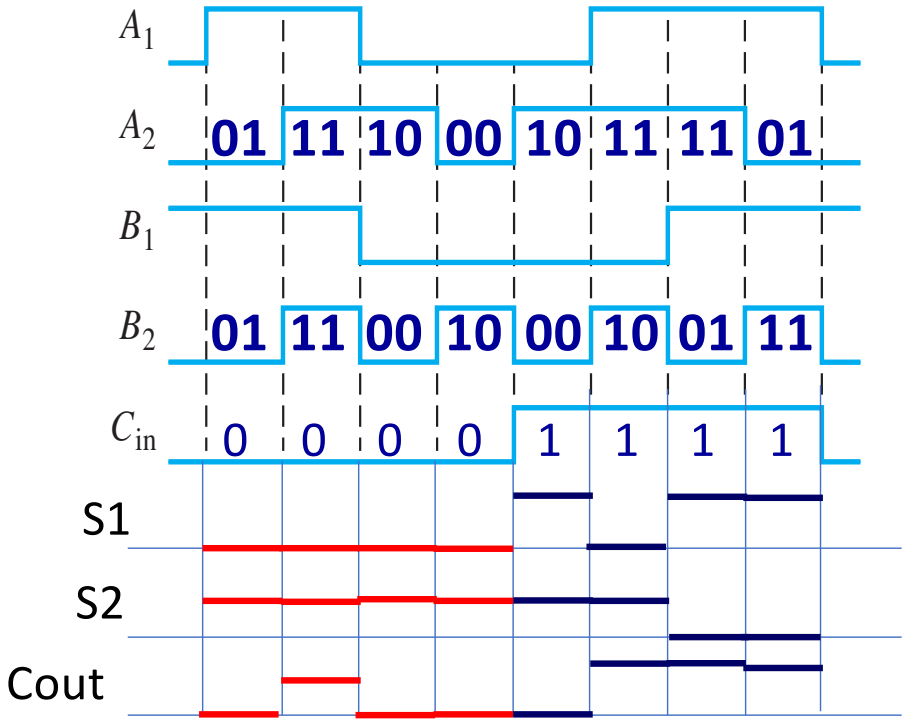
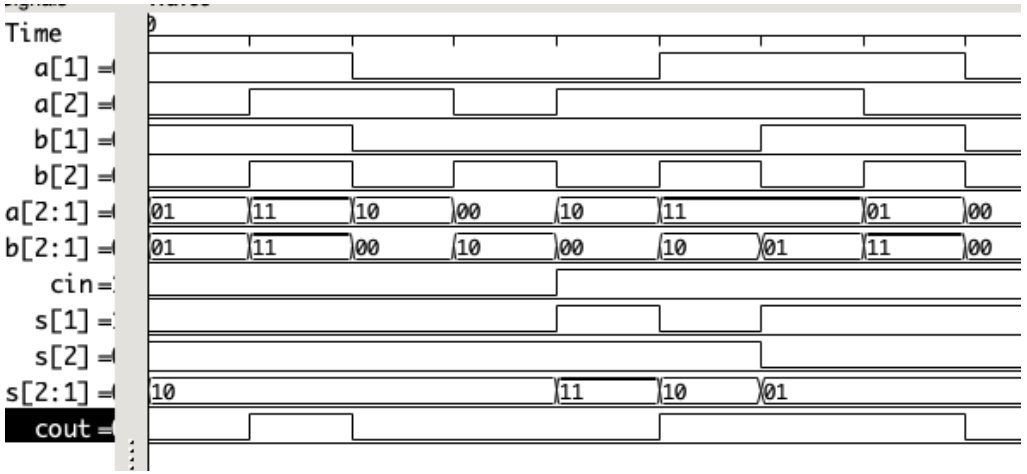


FIGURE 6–72

Apenas FA


A_1
A_2 01 11 10 00 10 11 11 01
B_1
B_2 01 11 00 10 00 10 01 11
C_{in} 0 0 0 0 1 1 1 1
S1
S2
Cout


Time
a[1]
a[2]
b[1]
b[2]
a[2:1] 01 11 10 00 10 11 01 00
b[2:1] 01 11 00 10 00 10 01 11 00
cin
s[1]
s[2]
s[2:1] 10 11 10 01
cout

Exercícios (3/8)

Determine a faixa de representação para 8 bits

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

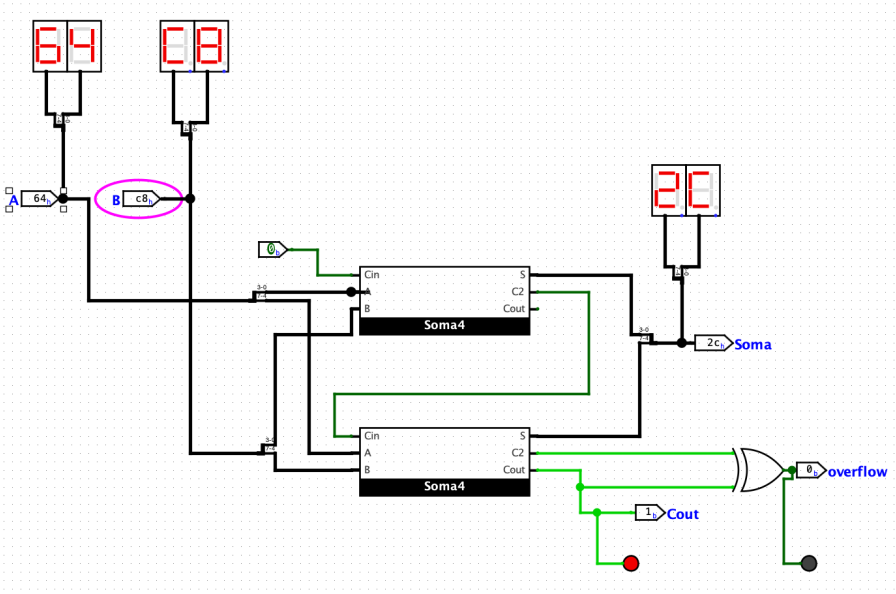
Avalie a soma abaixo considerando os números representados em inteiro sem sinal e complemento de dois. Quais os valores de Cout e Ov obtidos?

```
  1
  0 1 1 0  0 1 0 0
+ 1 1 0 0  1 0 0 0
1 0 0 1 0 1 1 0 0
```

$(44)_{10}$ $(2C)_{16}$

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

Cout=1
ov=0

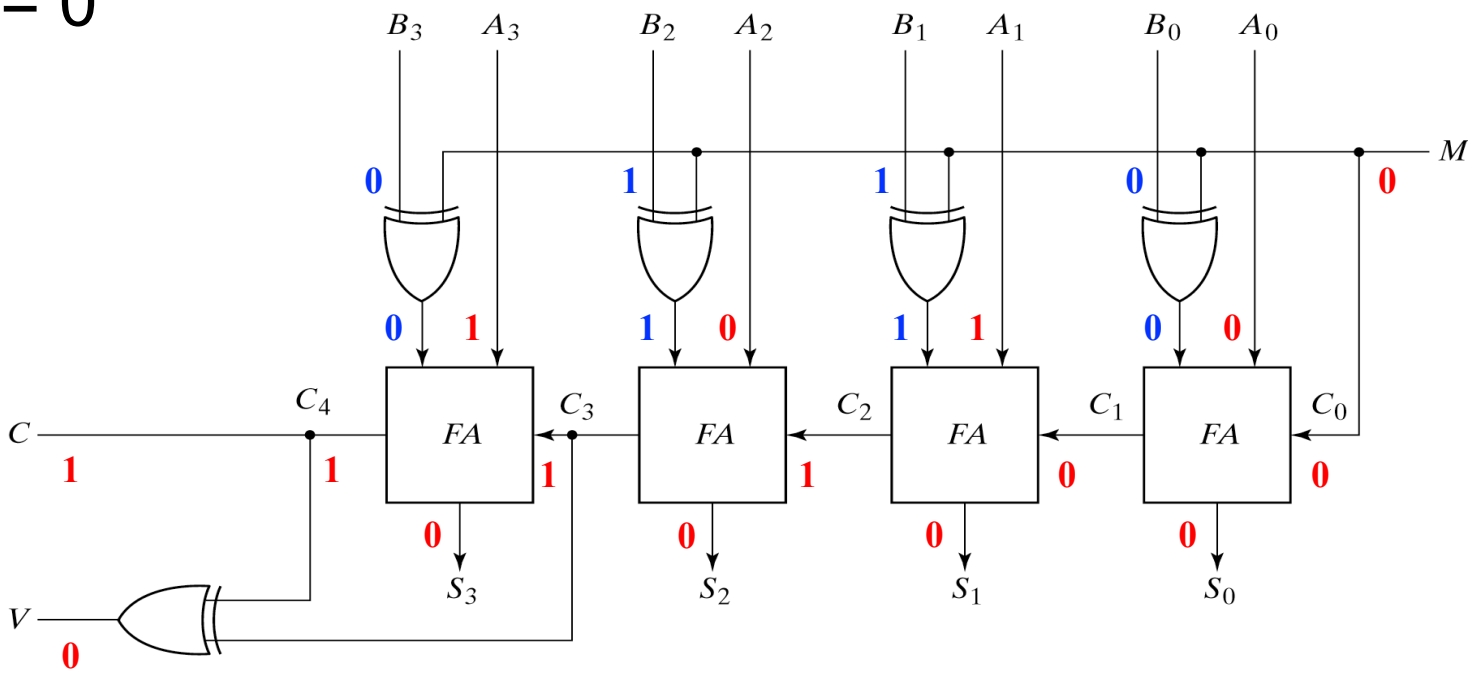


Soma correta em 2's mas incorreta em int. positivos

Exercícios (4/8)

Considere o **circuito SOMA/SUB** detalhado abaixo. Determine a saída ***soma*** (**S**), ***carry out*** (**C**) e ***overflow*** (**V**) para os estímulos fornecidos.

- ✓ A = A **(10 / -6)**
- ✓ B = 6
- ✓ Mode = 0



- ✓ **S = 0**
- ✓ **C = 1**
- ✓ **V = 0**

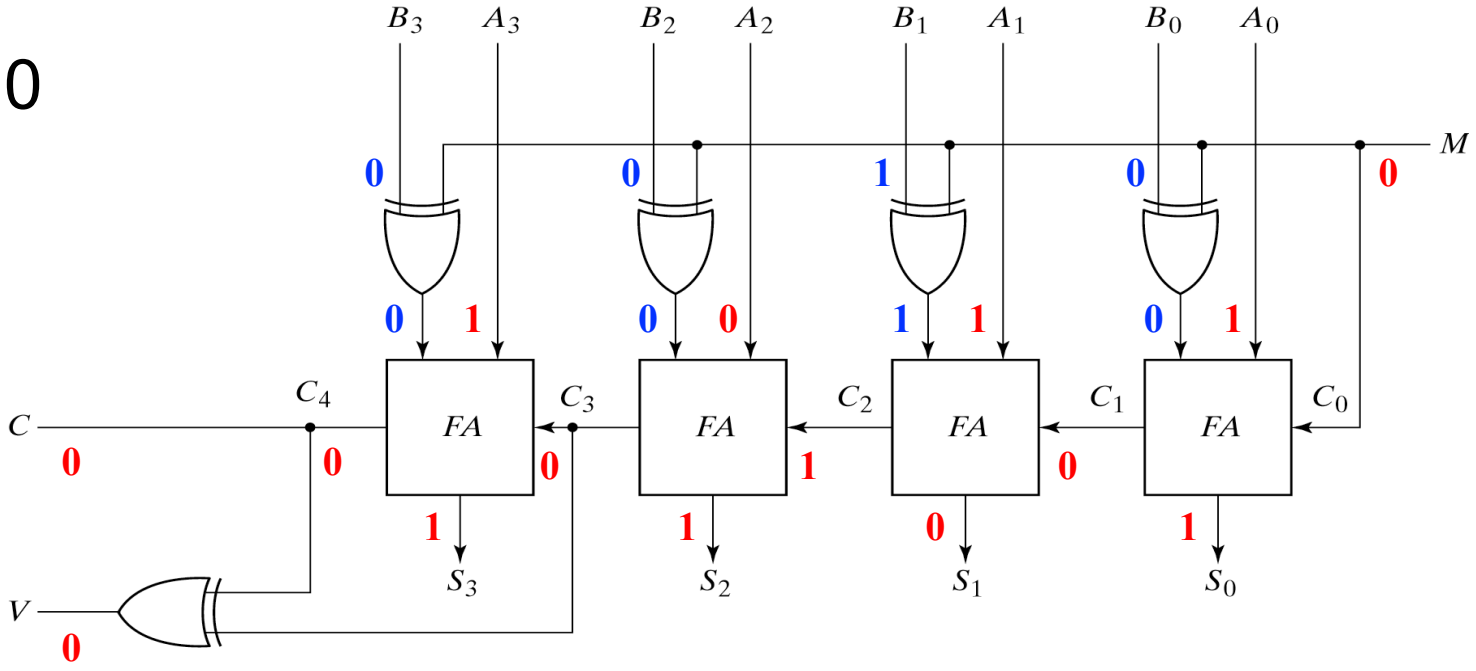
(0 / 0)

4 bits: $[0..15]_N$ e $[-8..7]_I$

Exercícios (5/8)

Considere o **circuito SOMA/SUB** detalhado abaixo. Determine a saída ***soma*** (**S**), ***carry out*** (**C**) e ***overflow*** (**V**) para os estímulos fornecidos.

- ✓ A = B **(11 / -5)**
- ✓ B = 2
- ✓ Mode = 0



- ✓ S = D
- ✓ C = 0
- ✓ V = 0

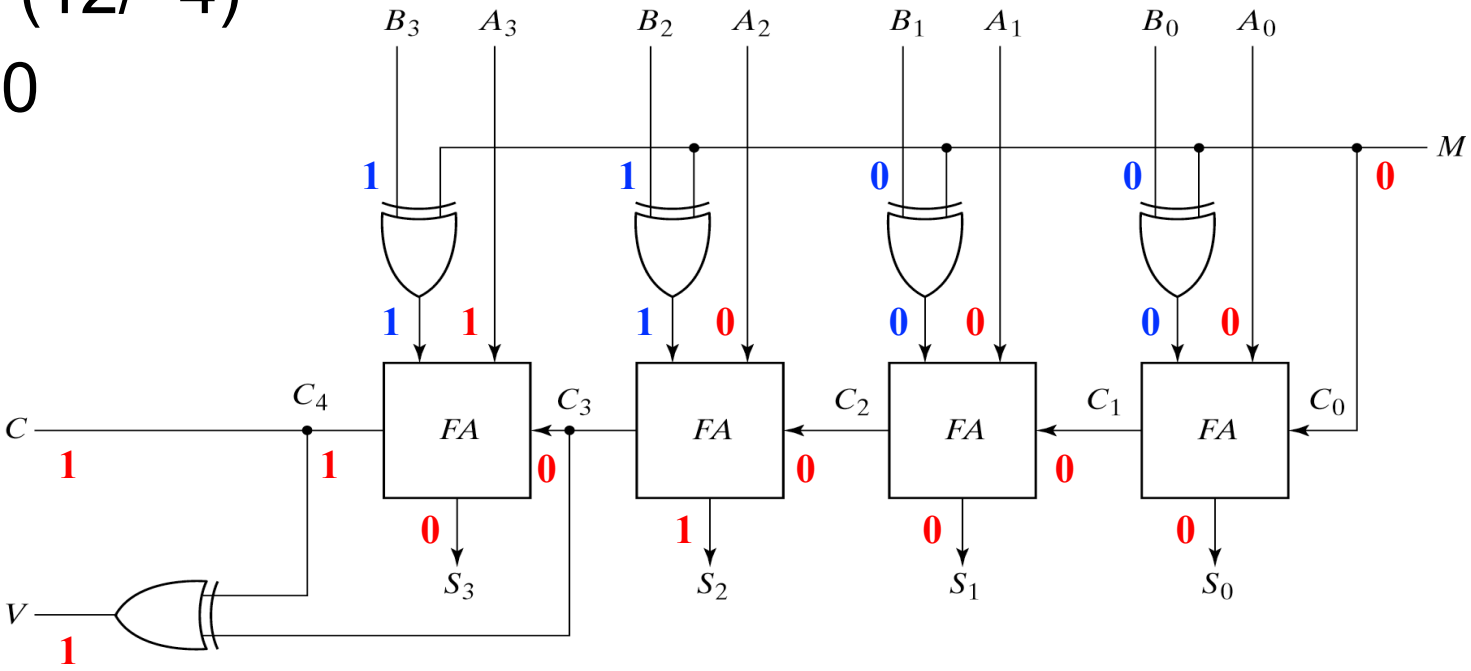
(13 / -3)

4 bits: $[0..15]_N$ e $[-8..7]_I$

Exercícios (6/8)

Considere o **circuito SOMA/SUB** detalhado abaixo. Determine a saída ***soma*** (**S**), ***carry out*** (**C**) e ***overflow*** (**V**) para os estímulos fornecidos.

- ✓ A = 8 (8 / -8)
- ✓ B = C (12/ -4)
- ✓ Mode = 0



- ✓ S = 4
- ✓ C = 1
- ✓ V = 1

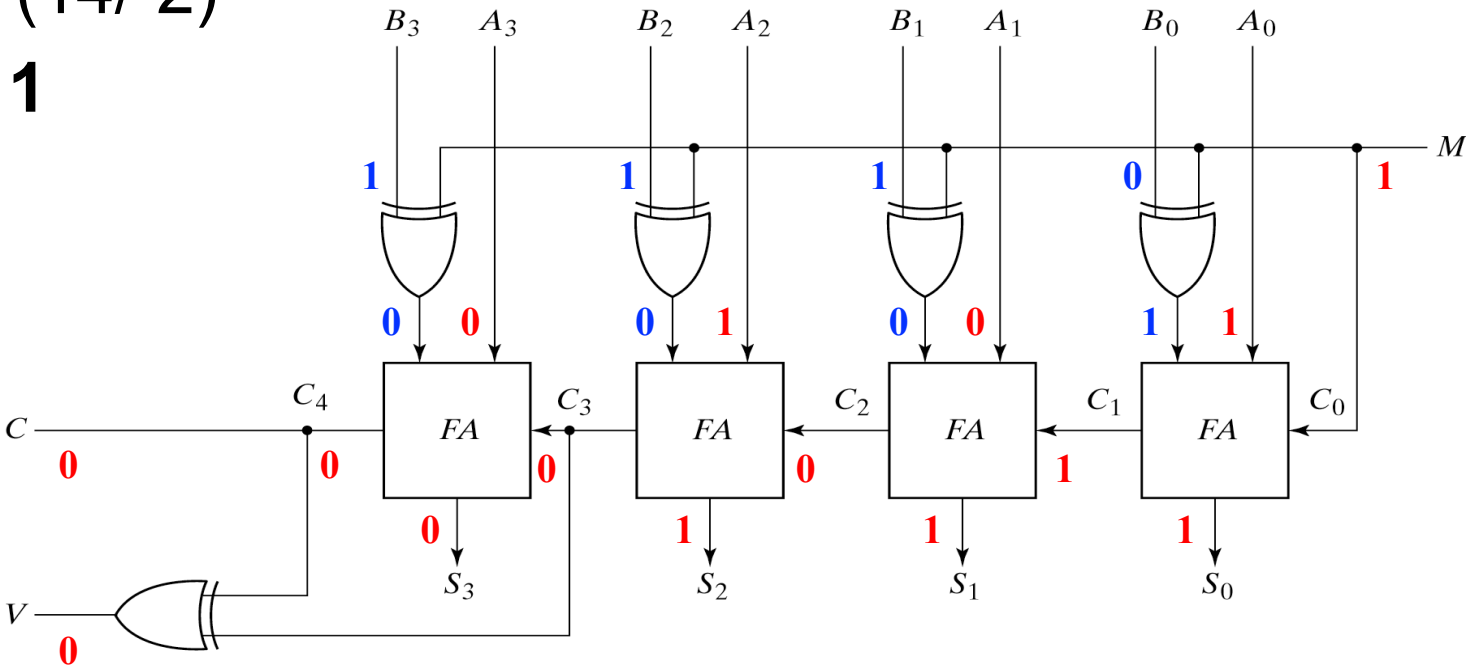
(4 / 4)

4 bits: $[0..15]_N$ e $[-8..7]_I$

Exercícios (7/8)

Considere o **circuito SOMA/SUB** detalhado abaixo. Determine a saída ***soma*** (**S**), ***carry out*** (**C**) e ***overflow*** (**V**) para os estímulos fornecidos.

- ✓ A = 5
- ✓ B = E (14/-2)
- ✓ **Mode = 1**



- ✓ S = 7
- ✓ C = 0 (inv)
- ✓ V = 0

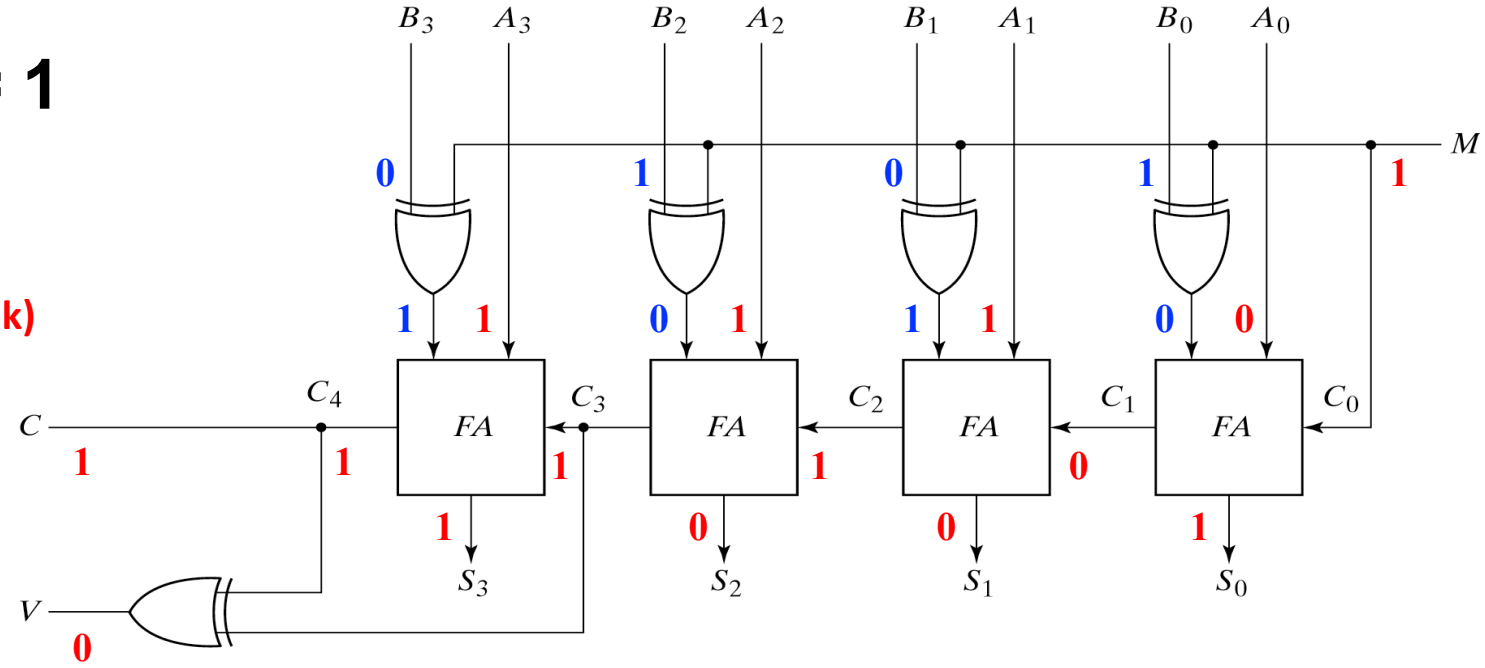
(7 / 7)

4 bits: $[0..15]_N$ e $[-8..7]_I$

Exercícios (8/8)

Considere o **circuito SOMA/SUB** detalhado abaixo. Determine a saída ***soma*** (**S**), ***carry out*** (**C**) e ***overflow*** (**V**) para os estímulos fornecidos.

- ✓ A = E (14, -2)
- ✓ B = 5
- ✓ **Mode = 1**

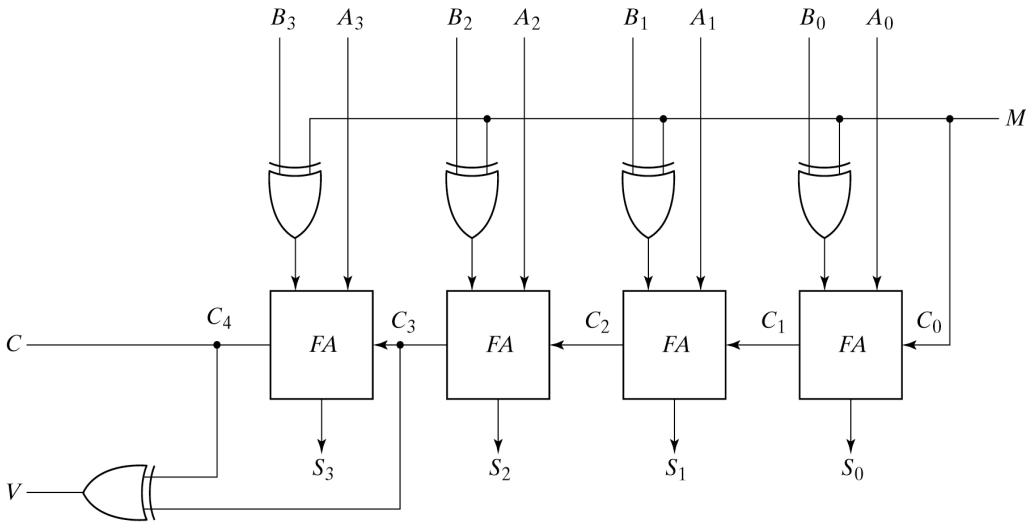


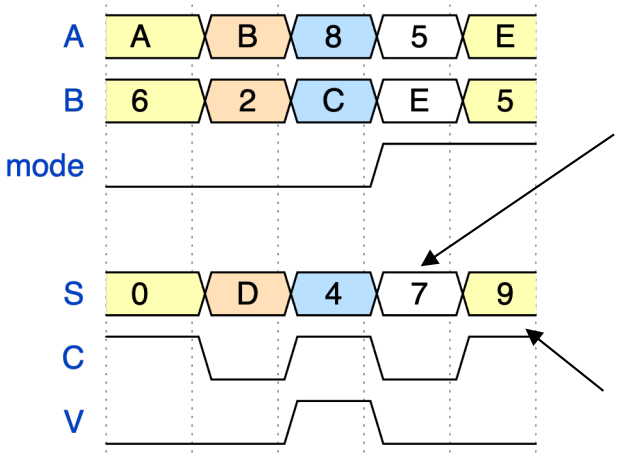
- ✓ S = 9
- ✓ C = 1 (inv – ok)
- ✓ V = 0

(9 / -7)

4 bits: $[0..15]_N$ e $[-8..7]_I$

RESUMO DESTES 5 EXERCÍCIOS



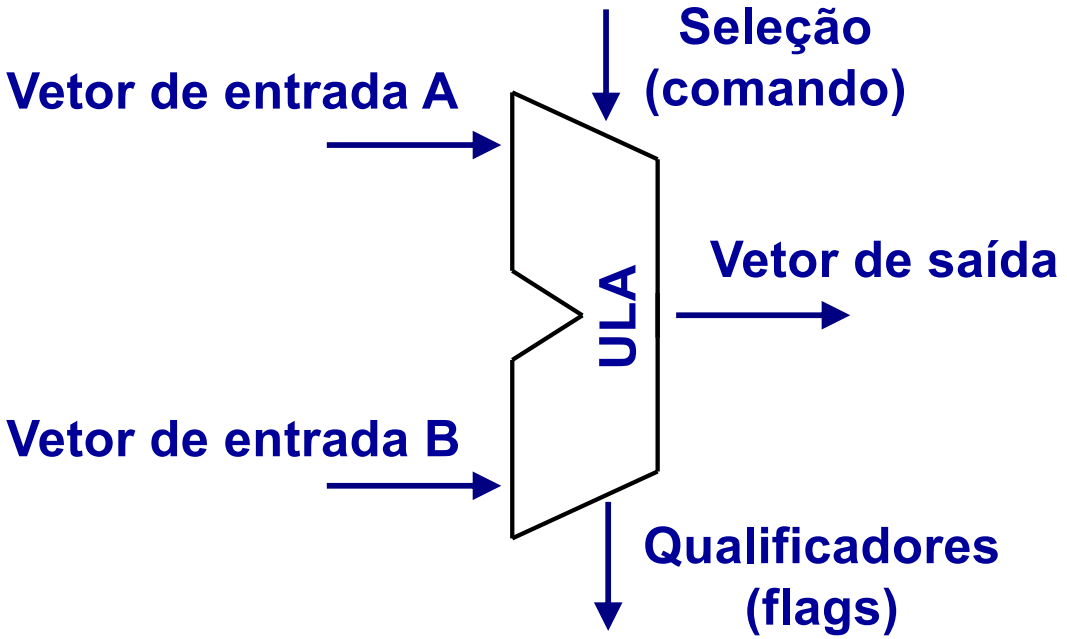


5-(-2): 5+2 = 7 (sem ov)
5-14 (ok, cout=0 indicando que há erro - não há negativo nos Naturais)

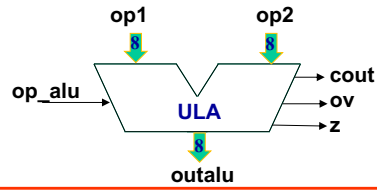
-2-5 = -7 (1001) (sem ov)
14-5 = 9 (ok, cout=1 pois cout é invertido na subtração)

(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:

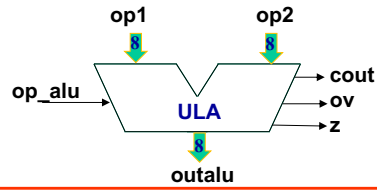


ULA – Funções Lógicas



- **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

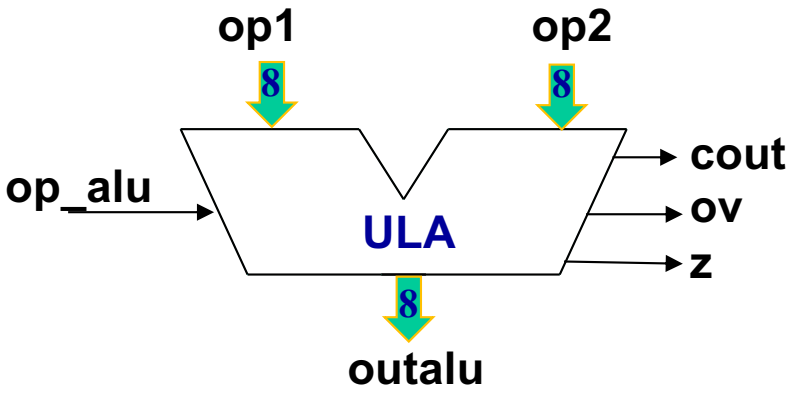


- **Dentre as funcionalidades aritméticas mais comuns estão:**
  - Soma das entradas
  - Subtração das entradas
  - Deslocamento de uma das entrada
  - Rotação de uma das entradas
  - Complemento de dois de uma das entradas
  - E variações das funcionalidades acima utilizando a flag C/OV
- **A seleção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

ULA - comandos

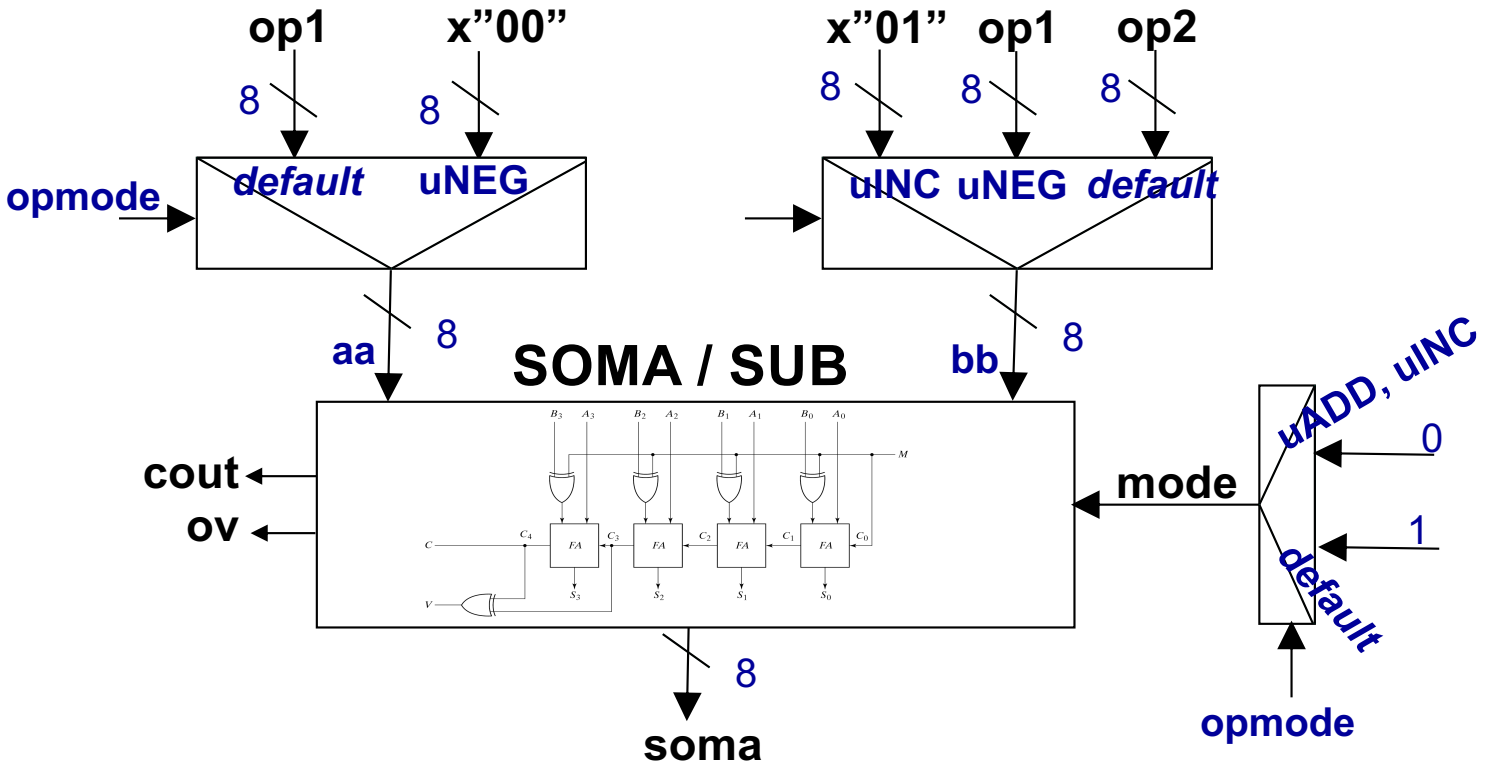
– **Exemplo operações:**

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



Operações Aritméticas: uso do “soma sub”

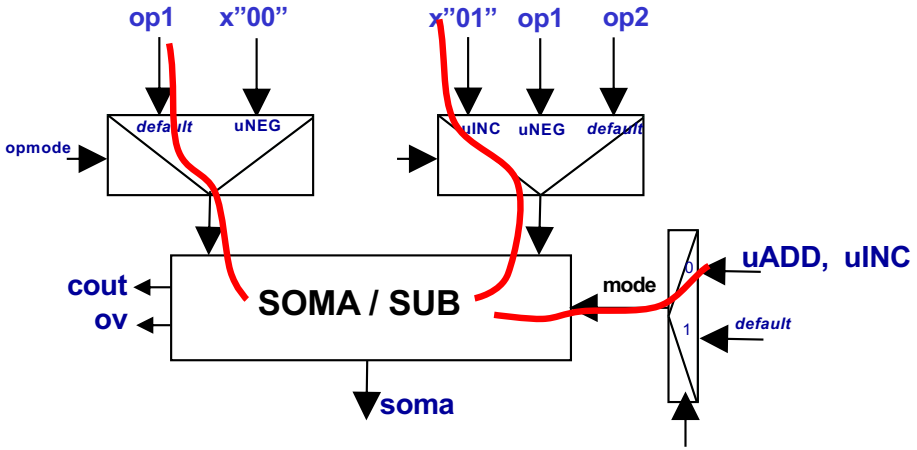
Exemplo para
4 operações
aritméticas



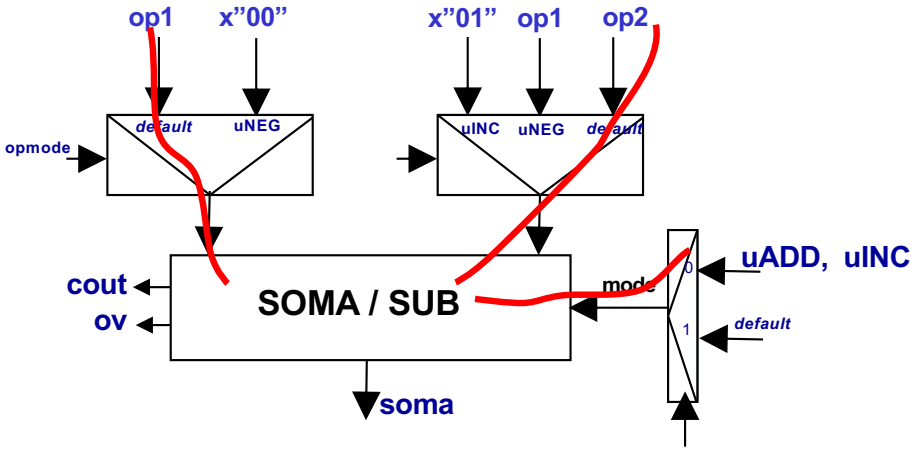
opmode	Ação
uINC	op1 + 1 + 0
uADD	op1+ op2 + 0
uSUB	op1+ not(op2) + 1
uNEG (2’s comp)	0 + not(op1) + 1

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

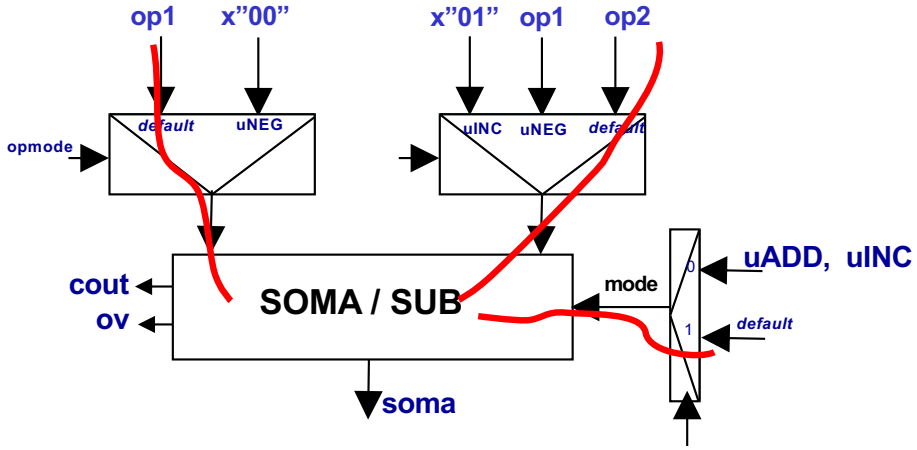
Instanciação do somador – operações



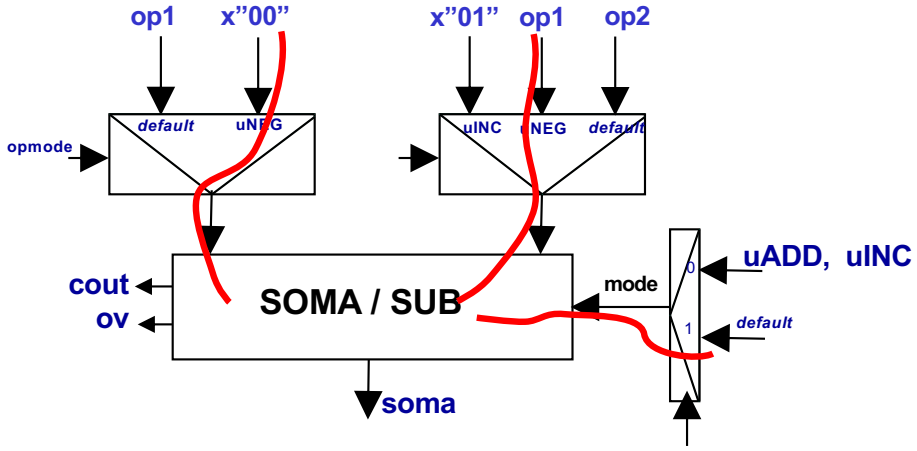
Inc: op1 + 1 + 0



ADD: op1 + op2 + 0



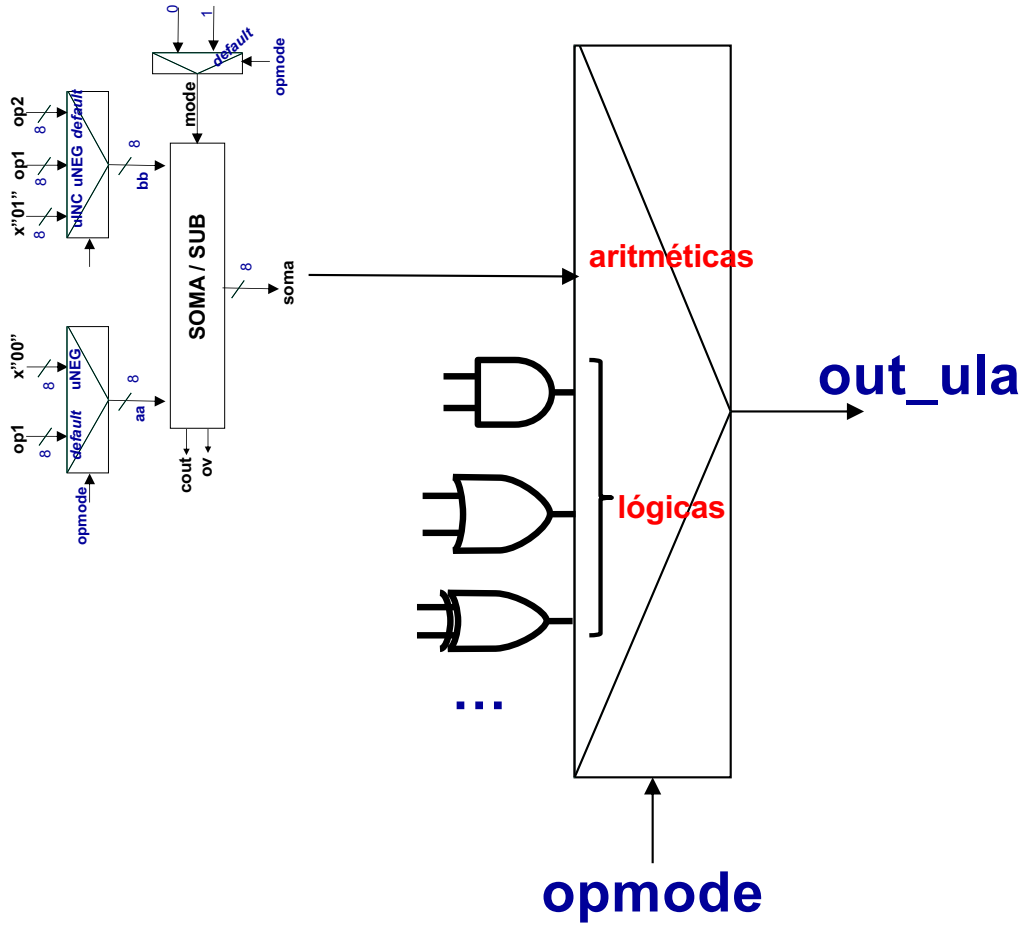
Sub: op1 + not(op2) + 1



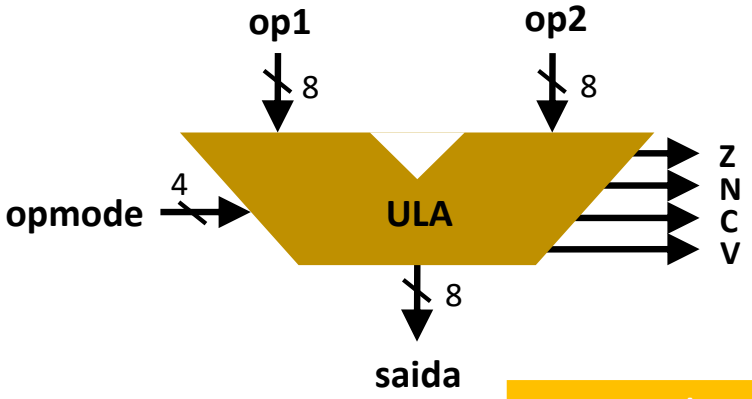
NEG (2's com): 0 + not(op1) + 1

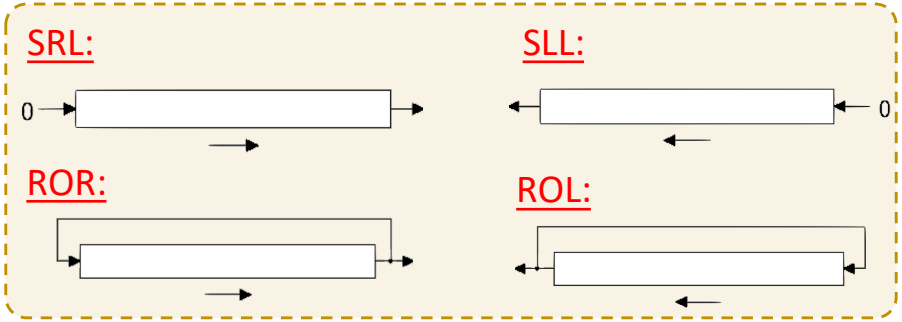
Como integrar com as operações lógicas?

Utilização de multiplexadores



Exemplo de operações realizadas pela ULA





opmode	Ação	Operação
0000	saida ← op1 + op2	Soma
0001	saida ← op1 - op2	Subtração
0010	saida ← op1 + 1	Incremento
0011	saida ← op1 - 1	Decremento
0100	saida ← NOT (op1)	Complemento
0101	saida ← op1 . op2	AND
0110	saida ← op1 + op2	OR
0111	saida ← op1 ⊕ op2	XOR
1000	saida ← SRL (op1)	Deslocamento lógico à direita
1001	saida ← SLL (op1)	Deslocamento lógico à esquerda
1010	saida ← ROR (op1)	Rotação à direita
1011	saida ← ROL (op1)	Rotação à esquerda
11--	saida ← 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

A+B → C= 01xx(A) / D= 0001(B) **ou** C=0001(B) e D=01xx(A) **/** Cin =0

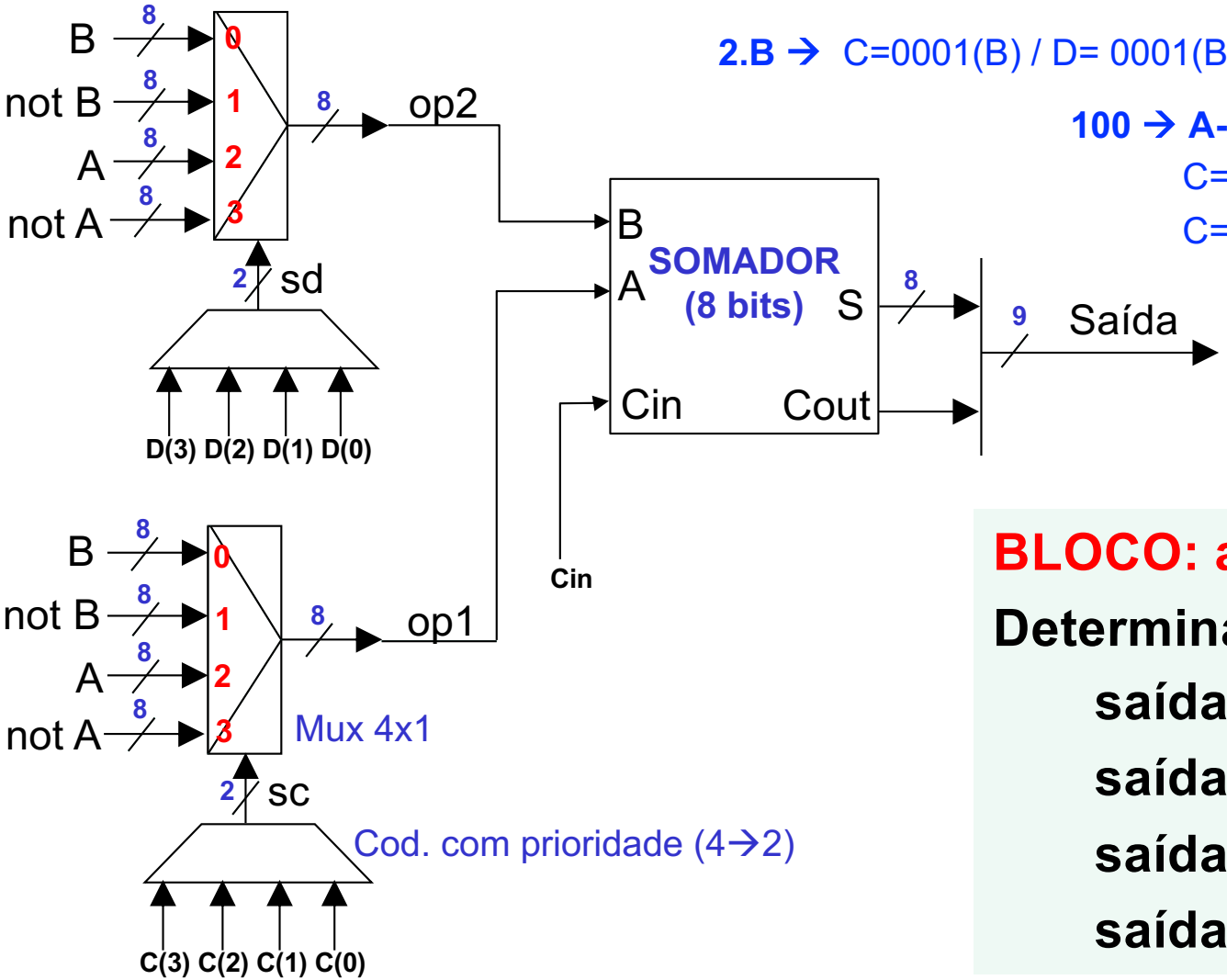
A-B → C= 01xx(A) / D= 001x(not B) / Cin =**1**

2.B → C=0001(B) / D= 0001(B) / Cin=**0**

100 → A-A ou B-B

C= 01xx(A) / D= 1xxx(not A) **/** Cin=1

C=0001(B) / D= 001x(not B) **/** Cin =1



BLOCO: apenas somador

Determinar C / D / Cin para:

- **saída ← A + B**
- **saída ← A - B**
- **saída ← 2.B**
- **saída ← 100_{16}**

Exercícios ULA (1/4)

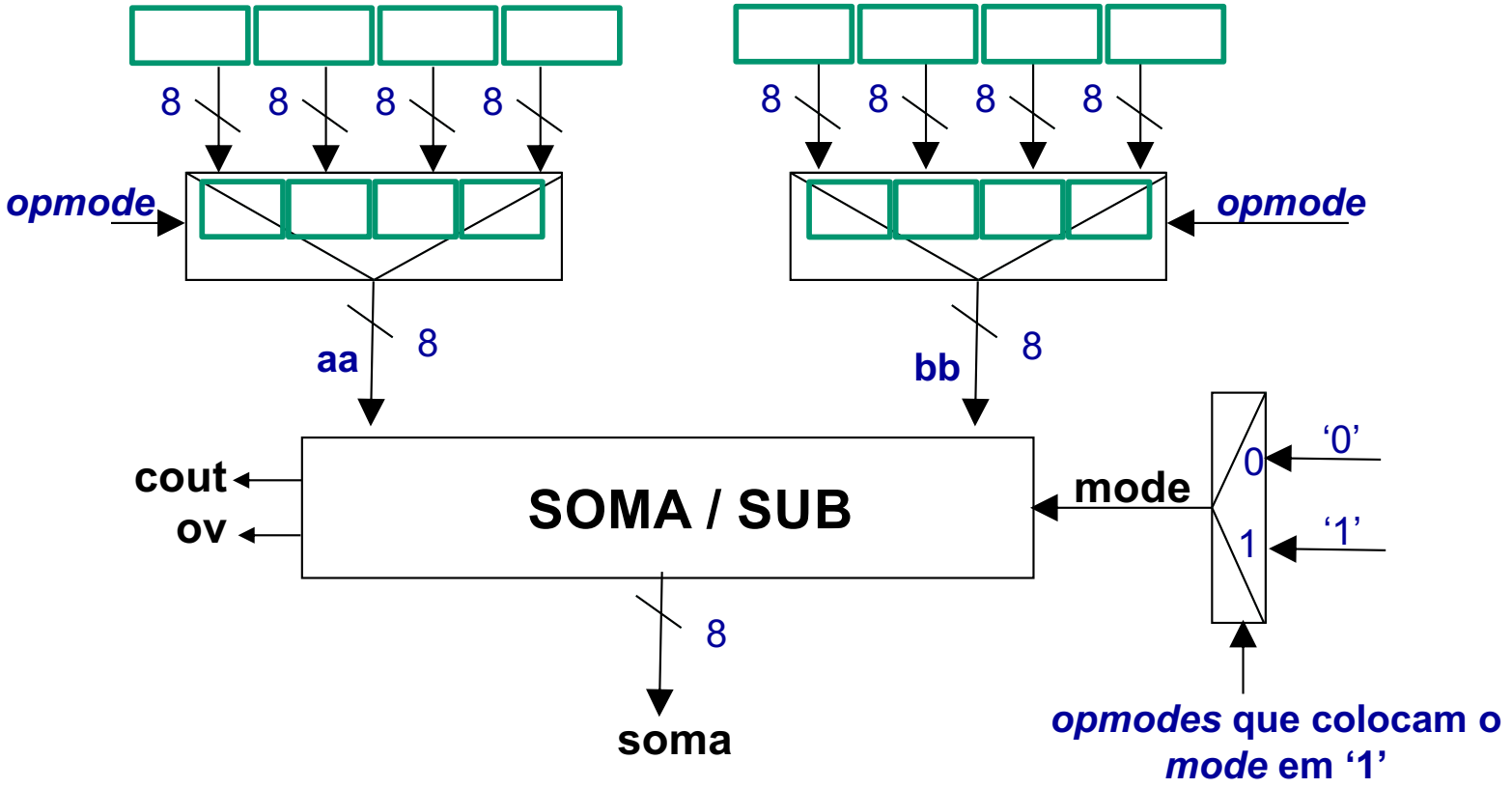
Projete a parte aritmética de uma ULA que realize 8 funções:

***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

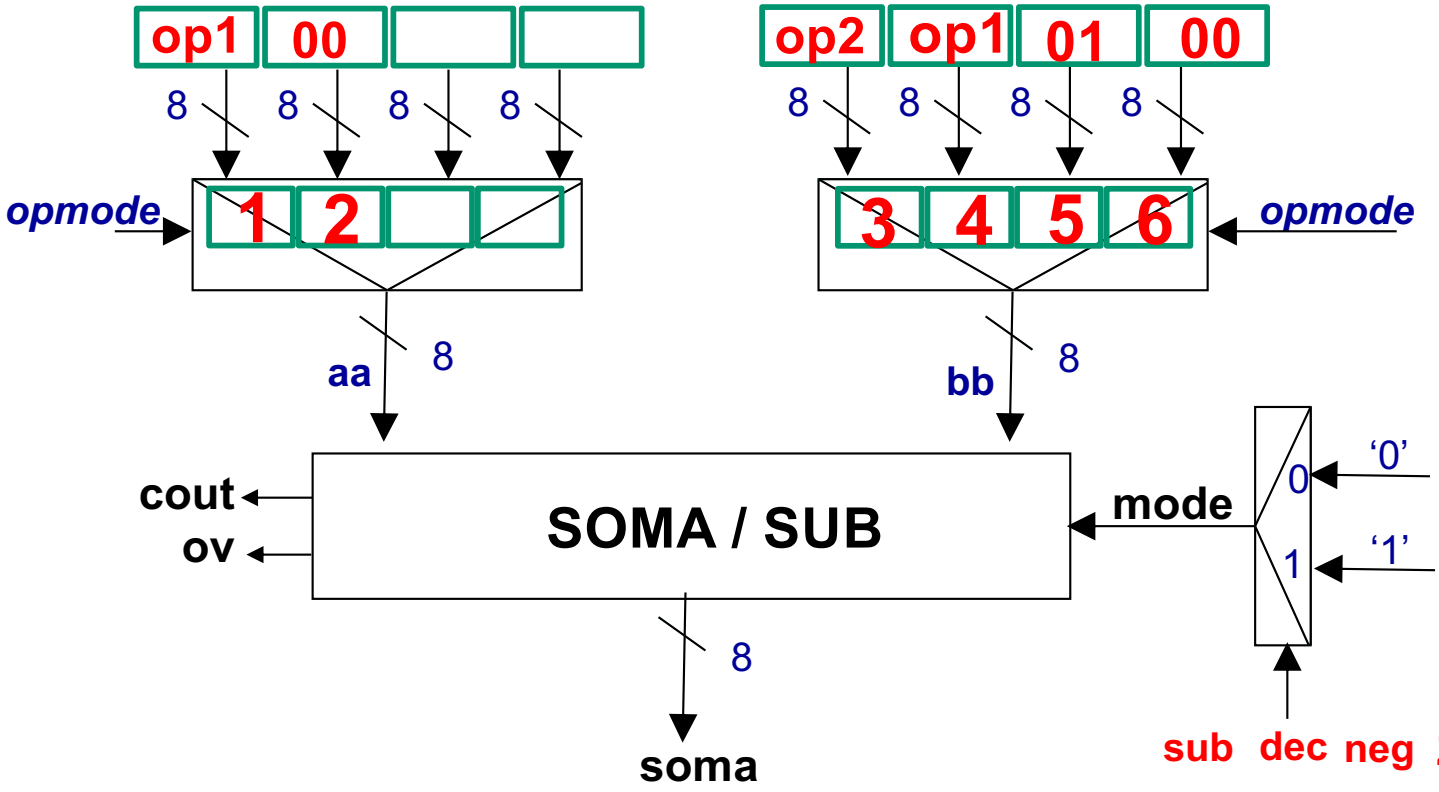
2: neg

3: sum sub

4: mu2 neg Z

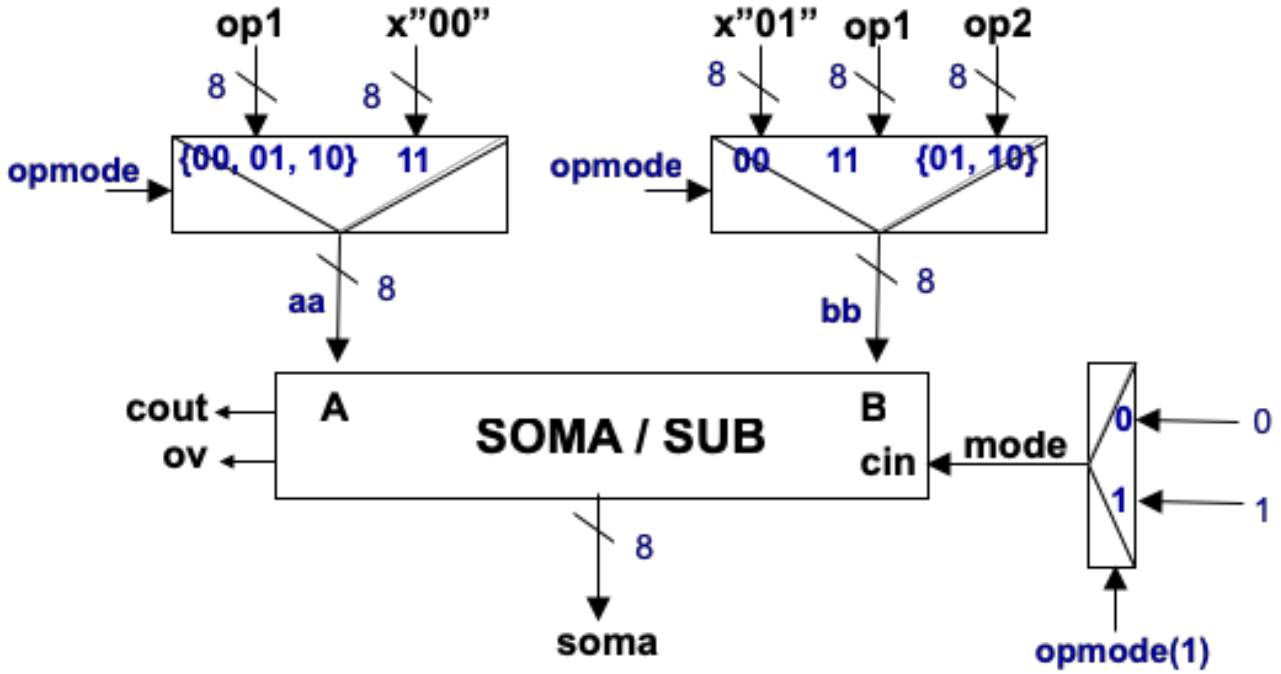
5: inc dec

6: P1



Exercícios ULA (2/4)

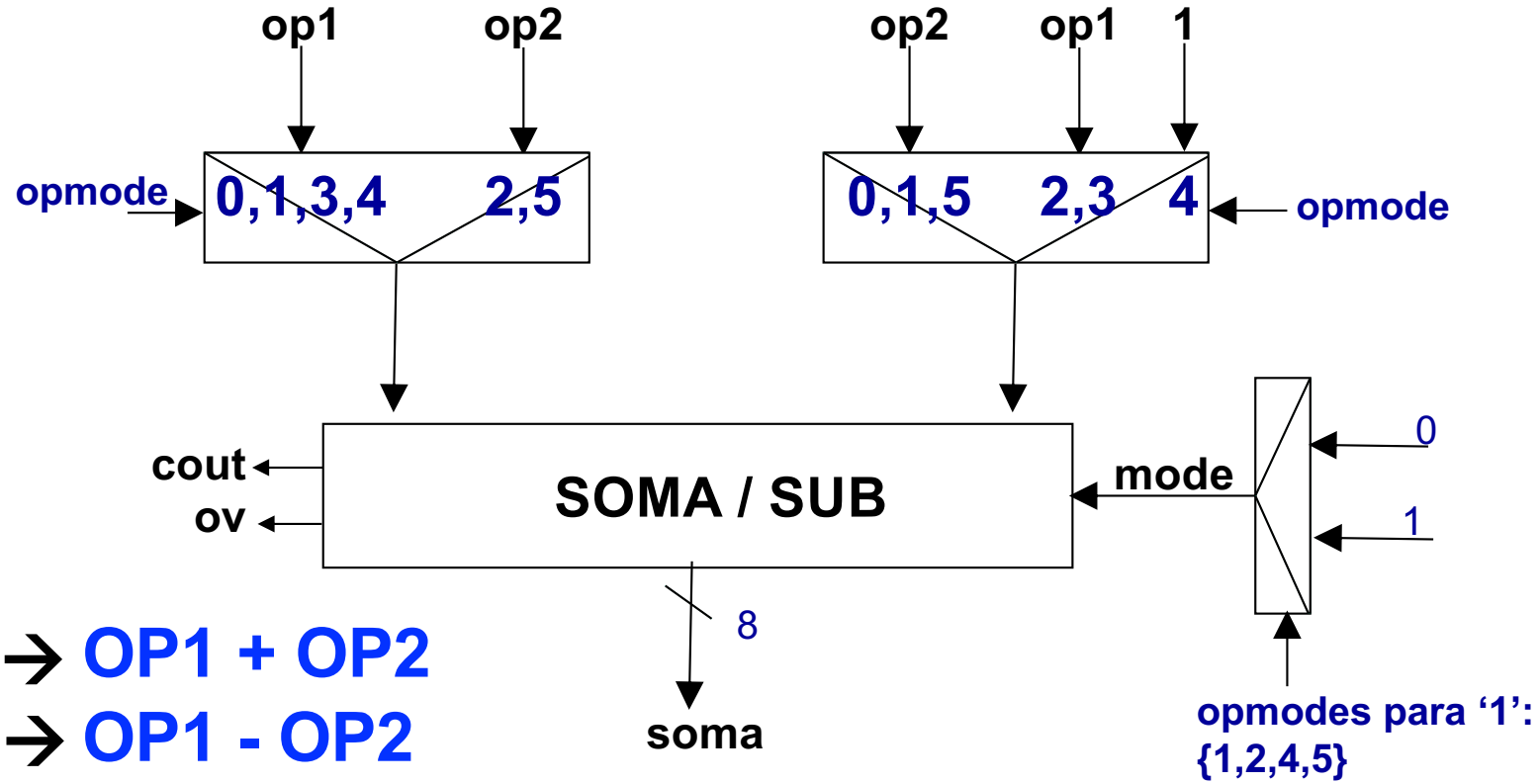
2. **O circuito somador/somador é parte integrante de uma ULA. Para um *opmode* de 2 bits, determine quais operações circuito soma/sub realiza, conforme a tabela abaixo.**



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

Exercícios ULA (3/4)

3) Indique as 6 operações executas pelo módulo soma/sub



0 → OP1 + OP2
1 → OP1 - OP2
2 → OP2 - OP1
3 → 2*OP1
4 → OP1- 1
5 → zero

Exercícios ULA (4/4)

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"									

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

```
   1 1 1 1
   0011 0100
+  0111 1110
-------------
(0) 1011 0010
```

```
          1
   0011 0100
+  1000 0001
-------------
          1
-------------
   1011 0110
```

```
          1 1
   1100 1011
+         1
-------------
   1100 1100
```

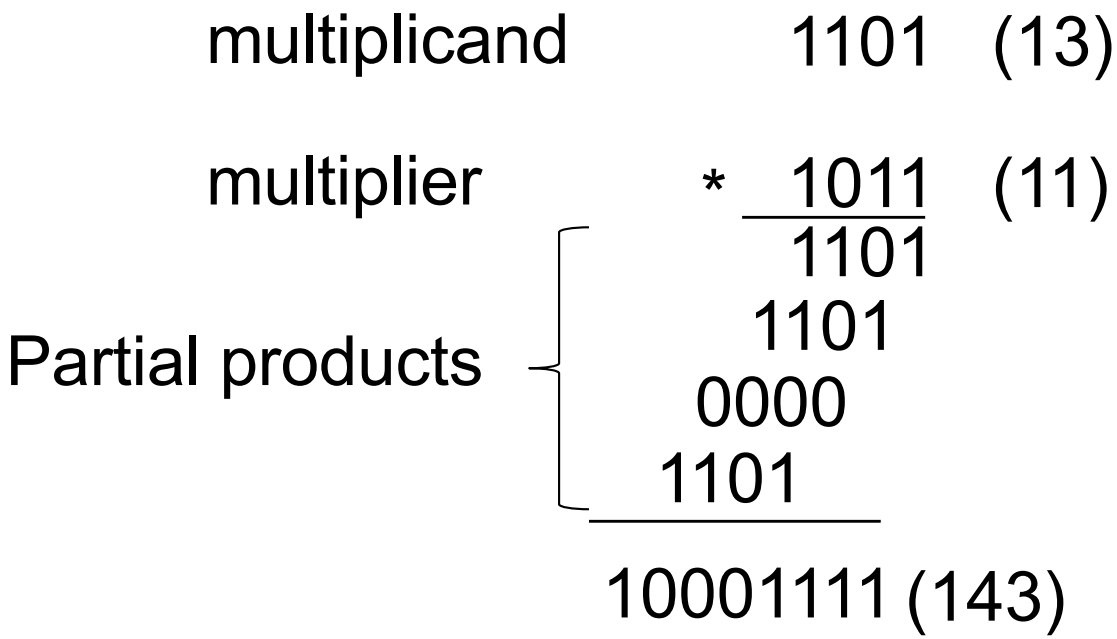
Recapitulando os blocos vistos

1. (De)Codificador
2. (De)multiplexador
3. Comparador
4. Gerador de paridade e verificação de paridade
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)

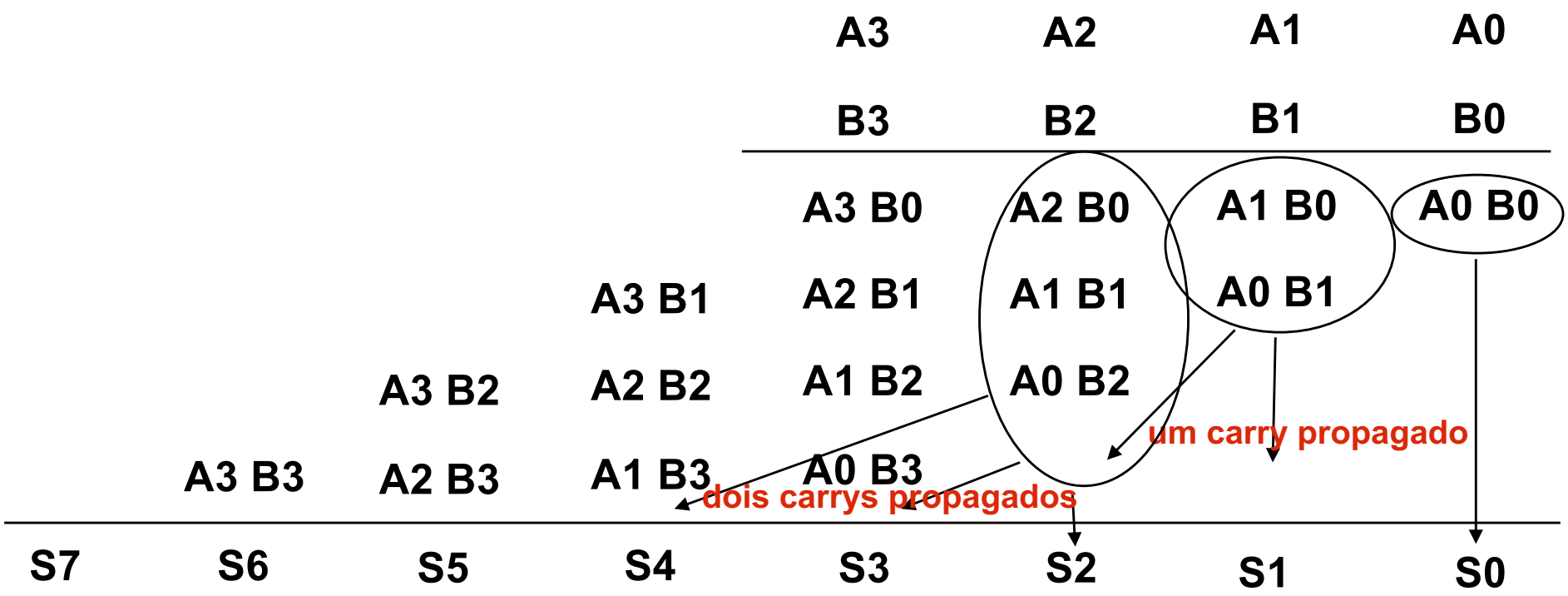
(9) MULTIPLICADOR

- **Dois números de *n* bits quando multiplicados, geram *2n* bits**

- **Exemplo:**

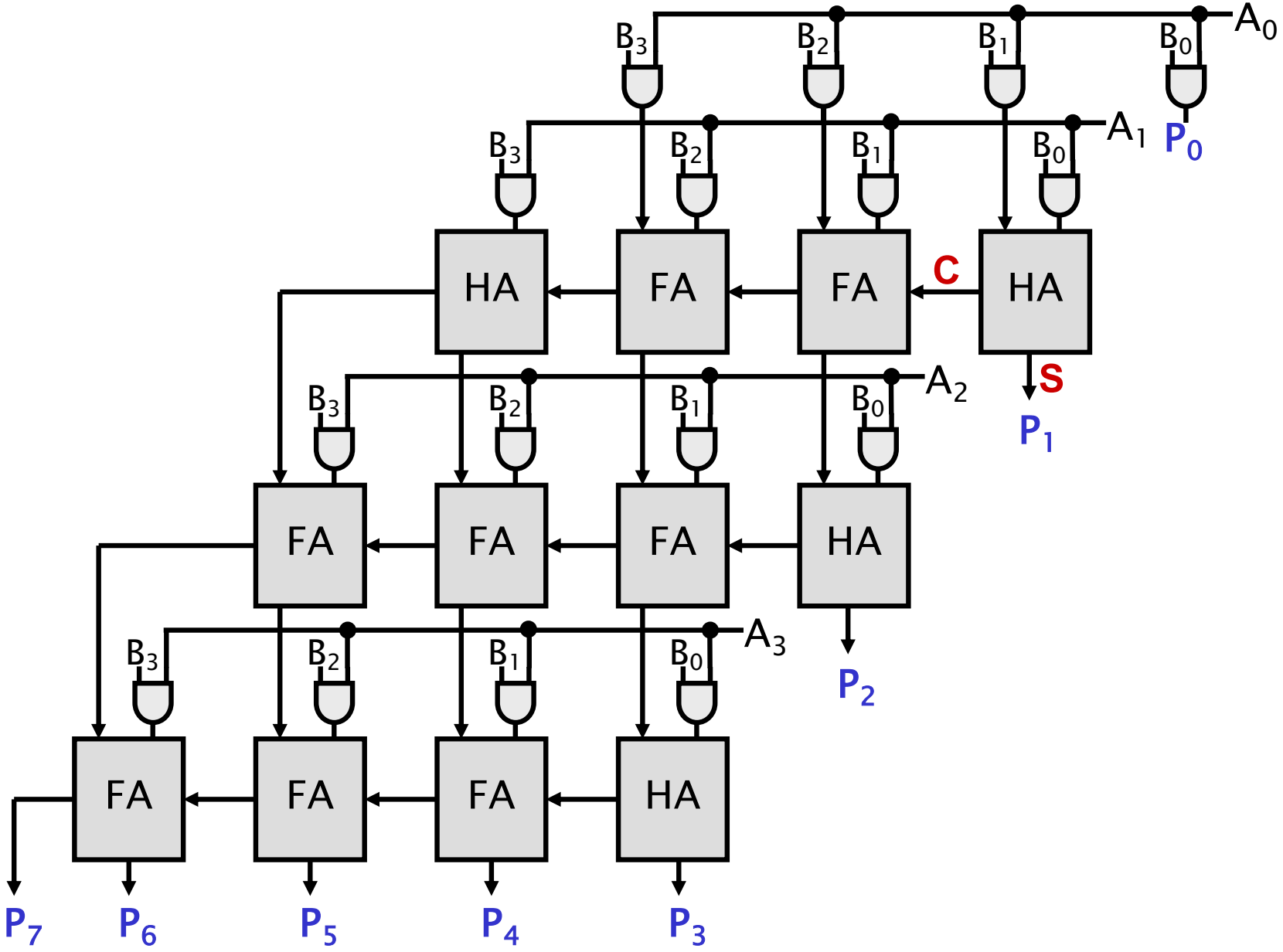


MULTIPLICAÇÃO



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

MULTIPLICAÇÃO – arquitetura ”array”



MULTIPLICAÇÃO – arquitetura ”array”

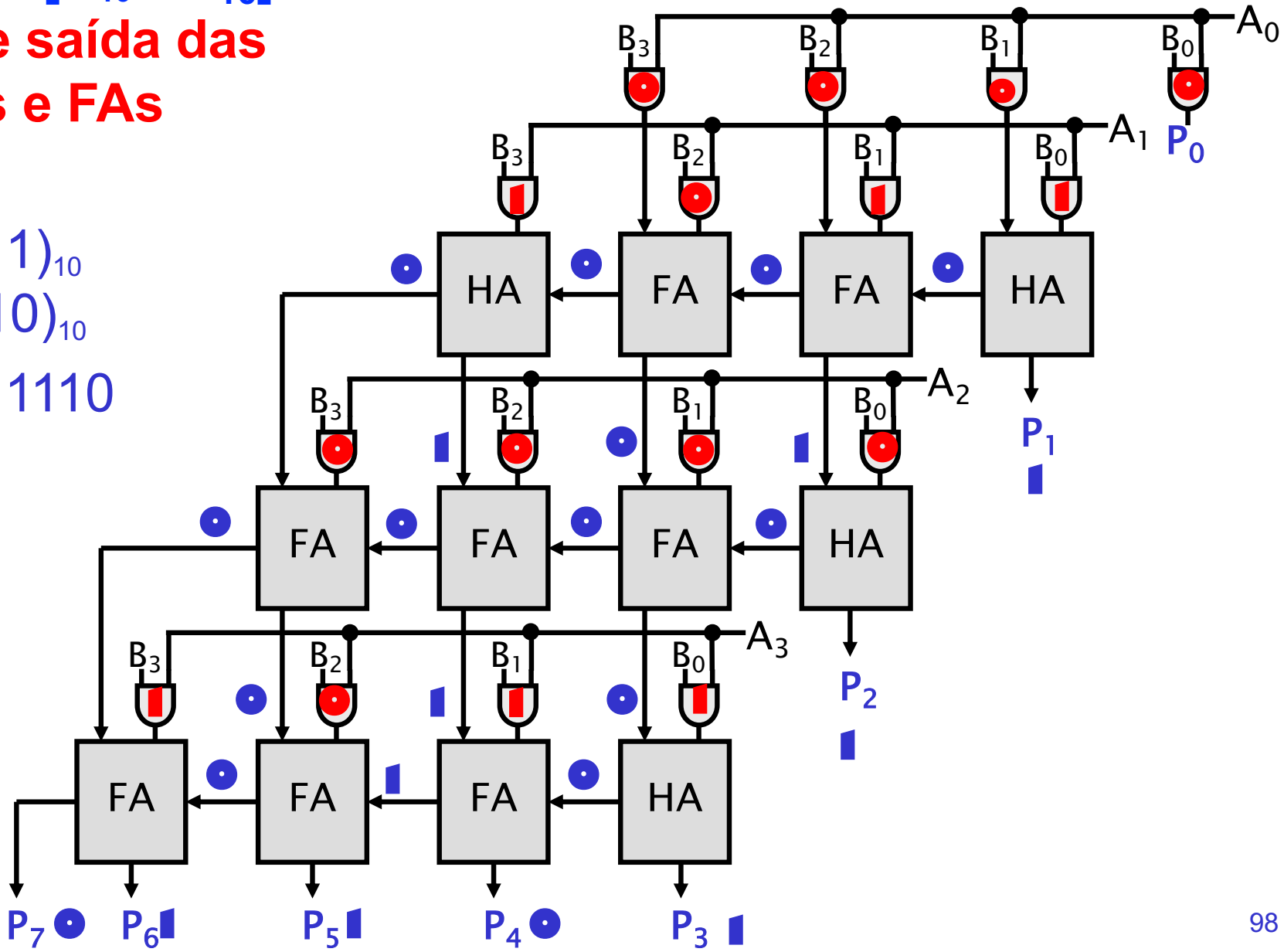
Multiplique [B_{16} * A_{16}] escrevendo os valores de saída das ands, HAs e FAs

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

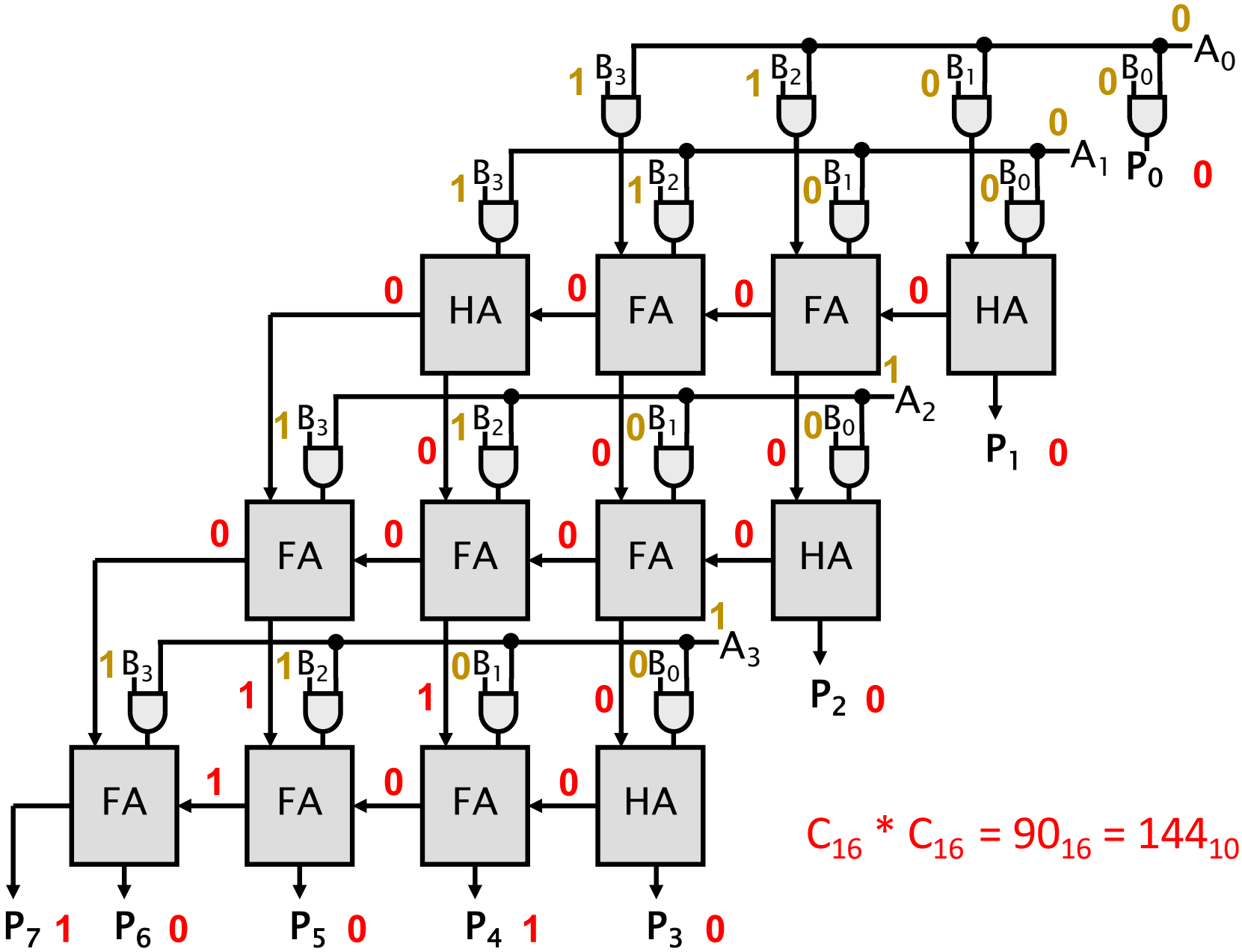
$A = 1010\ (10)_{10}$

$6E \rightarrow 0110\ 1110$

$(110)_{10}$

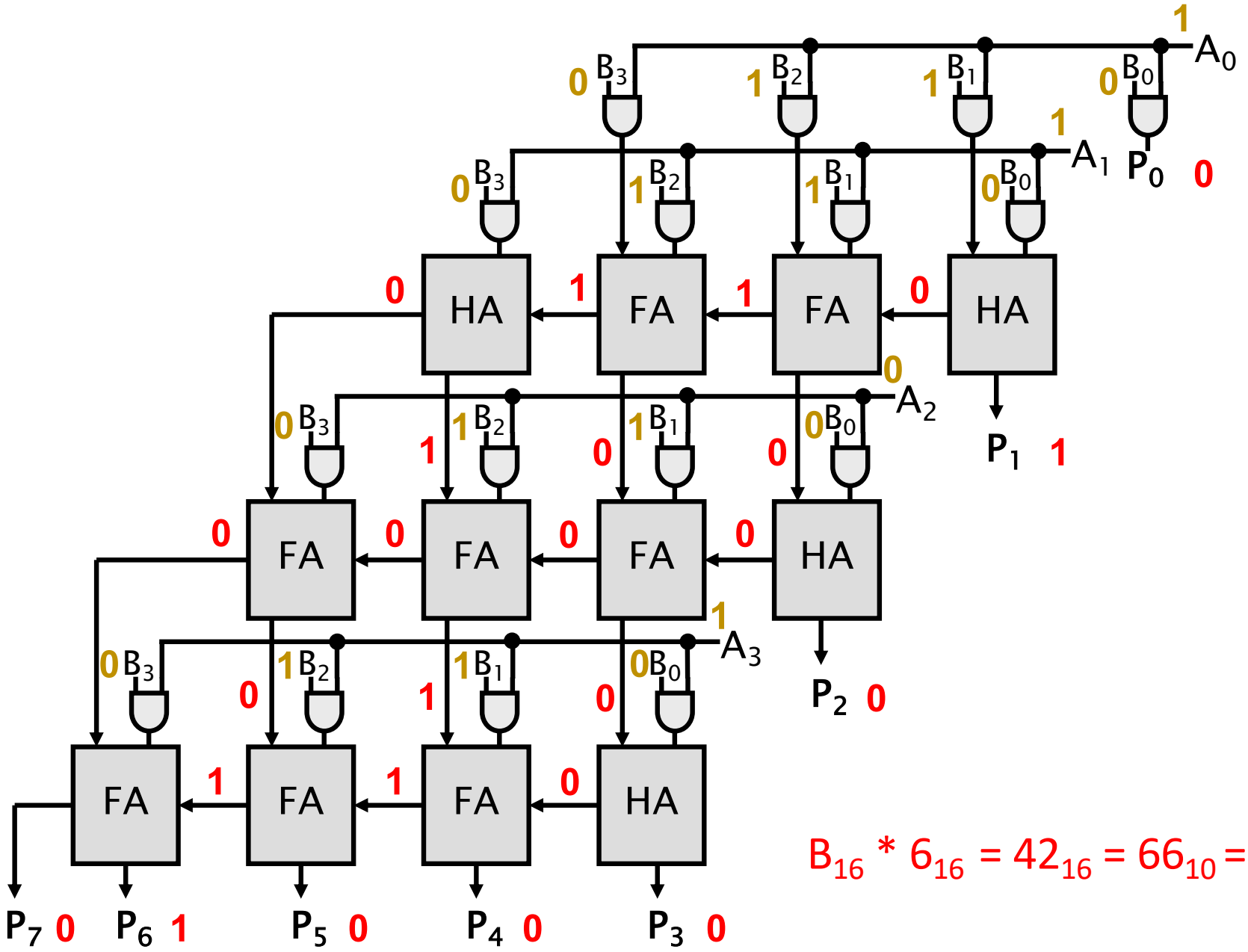


Multiplique [C_{16} * C_{16}] escrevendo os valores de saída das ands, HAs e FAs:



$C_{16} * C_{16} = 90_{16} = 144_{10} = 10010000_2$

Multiplique [B_{16} * 6_{16}] escrevendo os valores de saída das ands, HAs e FAs:



$B_{16} * 6_{16} = 42_{16} = 66_{10} = 01000010_2$