



IC-UNICAMP

MC 602

Circuitos Lógicos e Organização de Computadores

IC/Unicamp

Prof Mario Côrtes

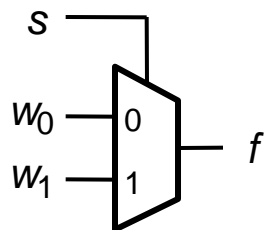
Capítulo 6

Circuitos Combinacionais Típicos

Tópicos

- Multiplexadores
- Decodificadores
- Codificadores binários e de prioridade
- Conversores de código
- Circuitos aritméticos de comparação
- Conversão bin-> 7 segmentos

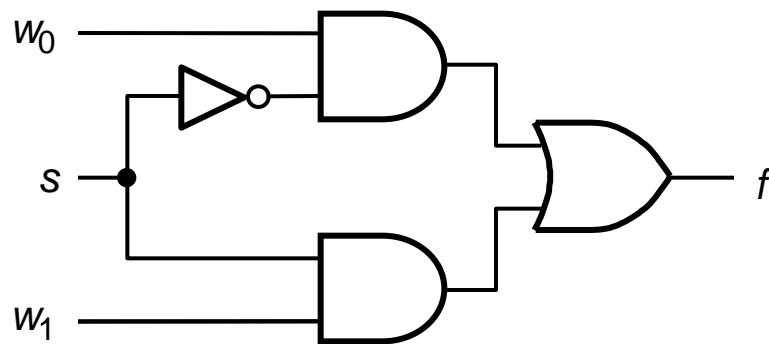
Mux 2:1



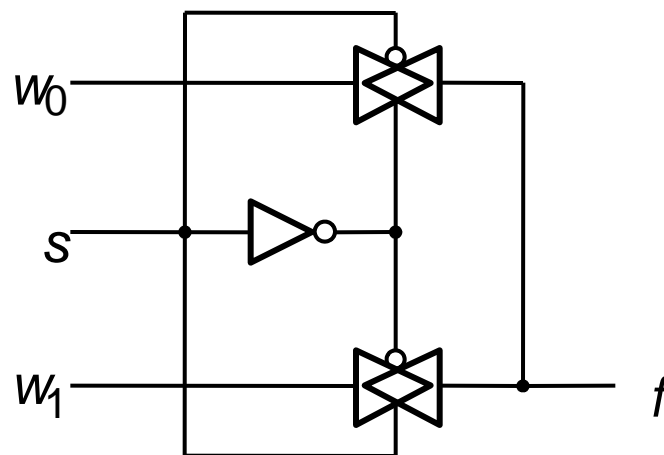
(a) Símbolo Gráfico

s	f
0	w_0
1	w_1

(b) Tabela verdade



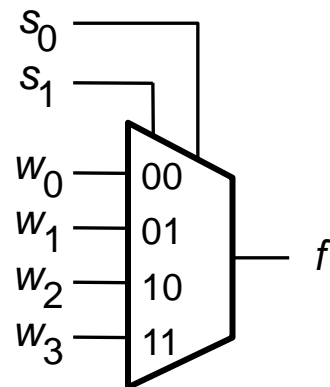
(c) Implementação em SOP



(d) Implementação com transmission gates

Mux 4:1

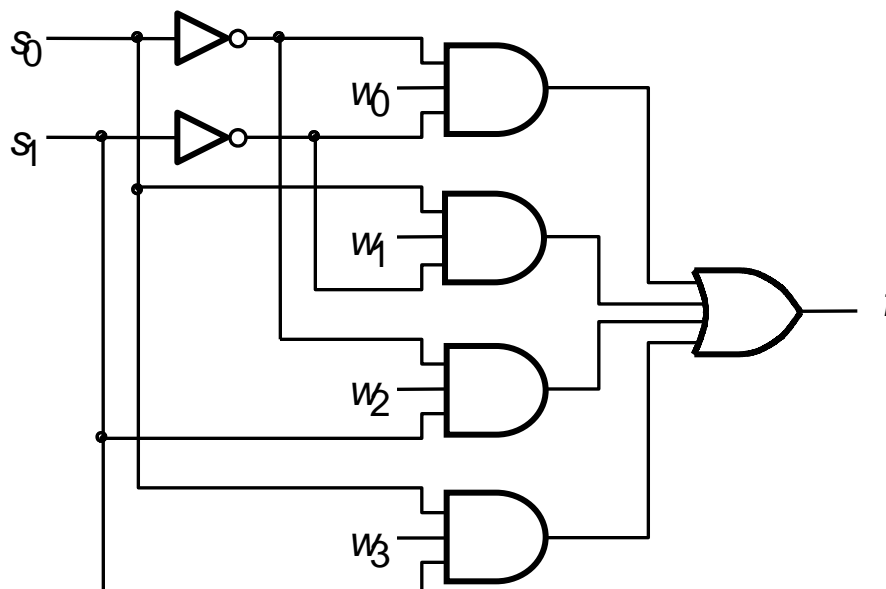
(a) Símbolo Gráfico



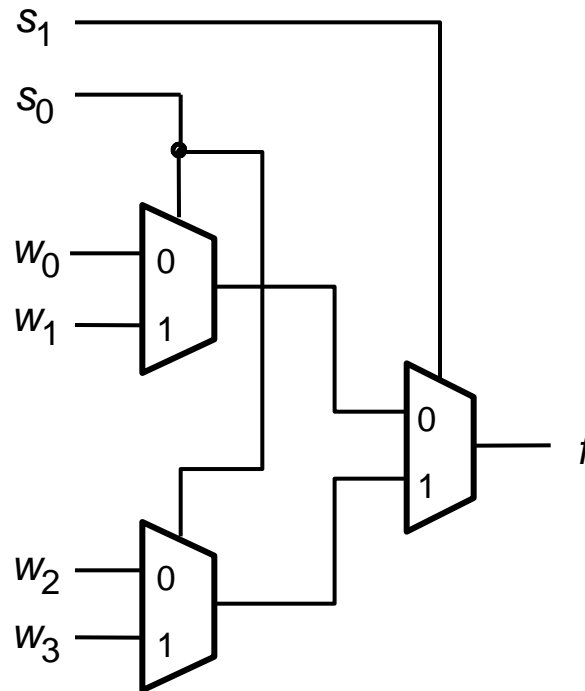
(b) Tabela verdade

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

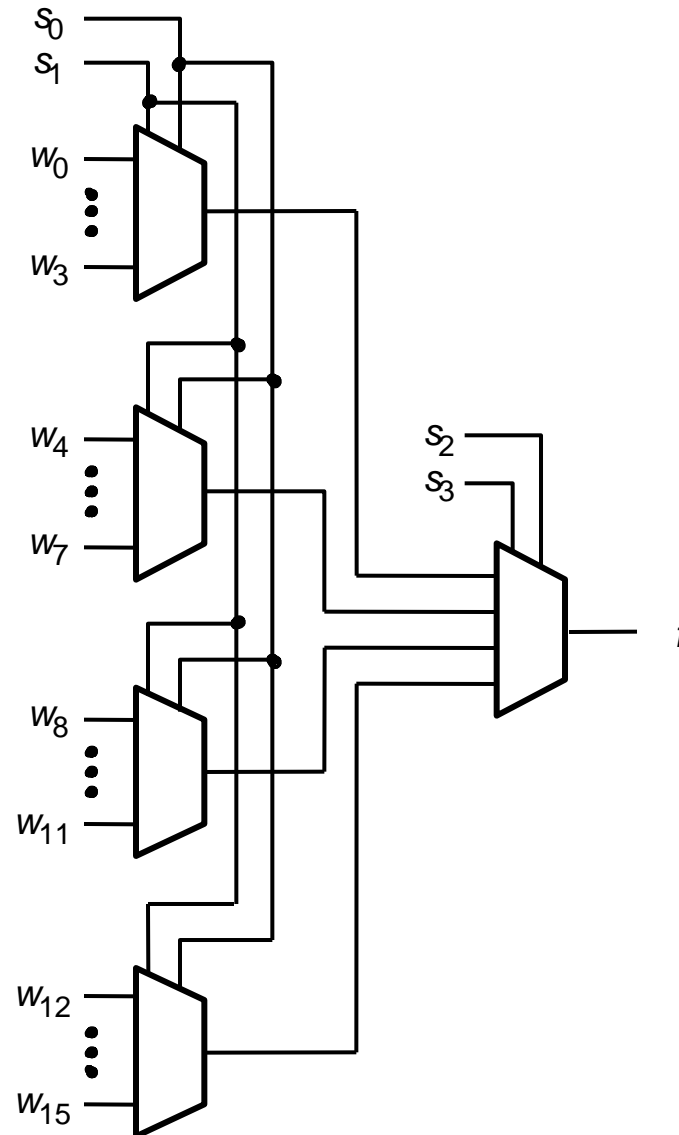
(c) Implementação em SOP



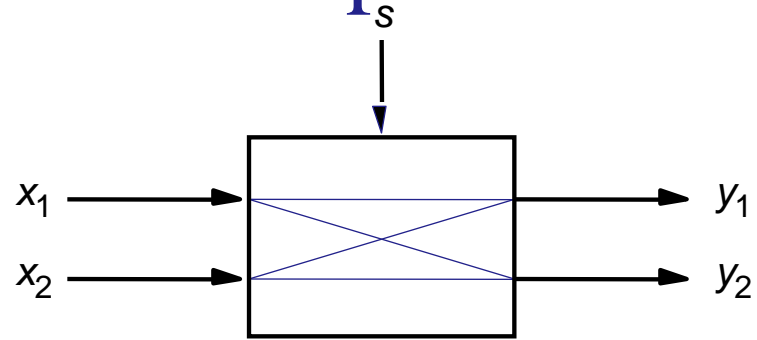
Mux 4:1 construído com Mux 2:1



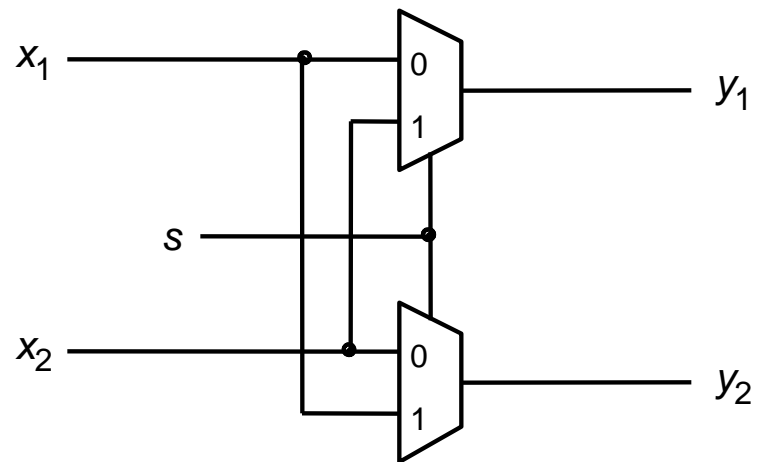
Mux 16:1 construído com Mux 4:1



Chave crossbar implementada c/ Mux



(a) Uma chave crossbar 2x2



(b) Implementação com Mux

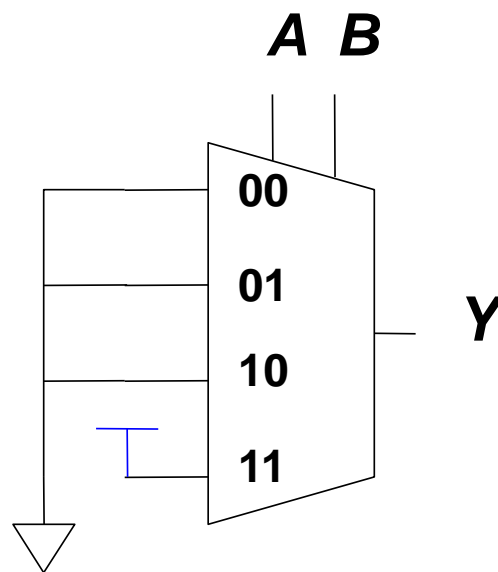


Lógica Usando Mux

- Usand o mux como uma **lookup table**

<i>A</i>	<i>B</i>	<i>Y</i>
0	0	0
0	1	0
1	0	0
1	1	1

$$Y = AB$$



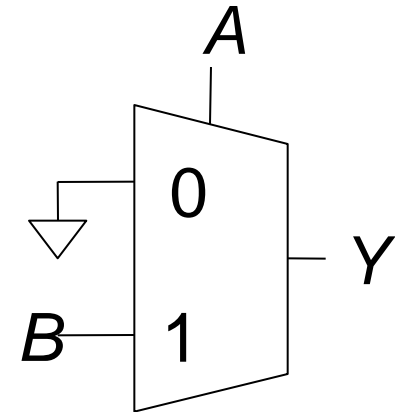
Implementando Funções com Mux

- Reduzindo o tamanho do Mux

$$Y = AB$$

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

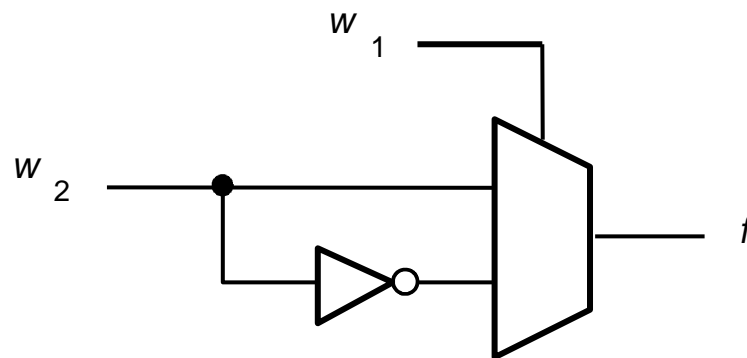
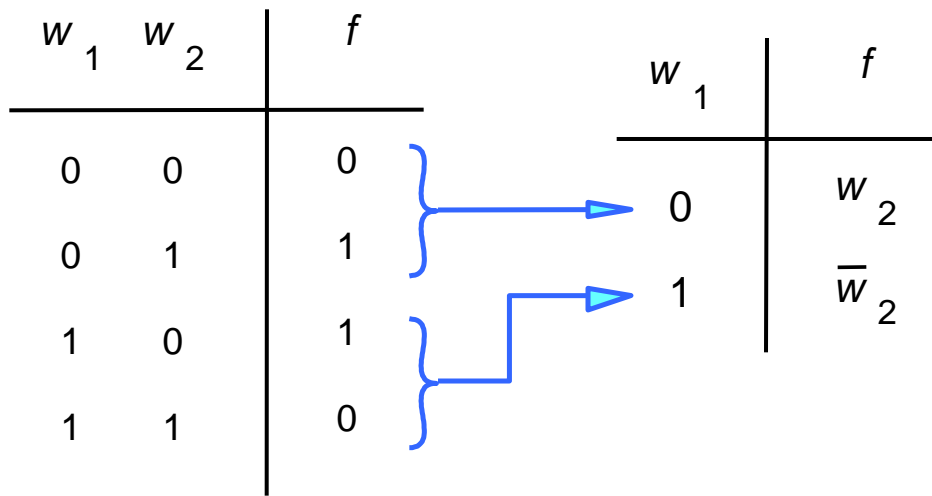
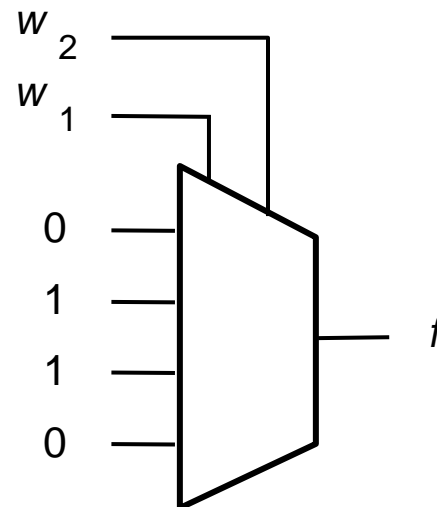
A	Y
0	0
1	B





Mux: implementação de funções lógicas

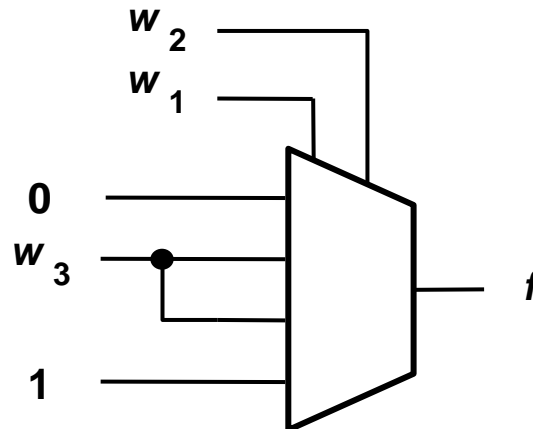
w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0



Exemplo

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

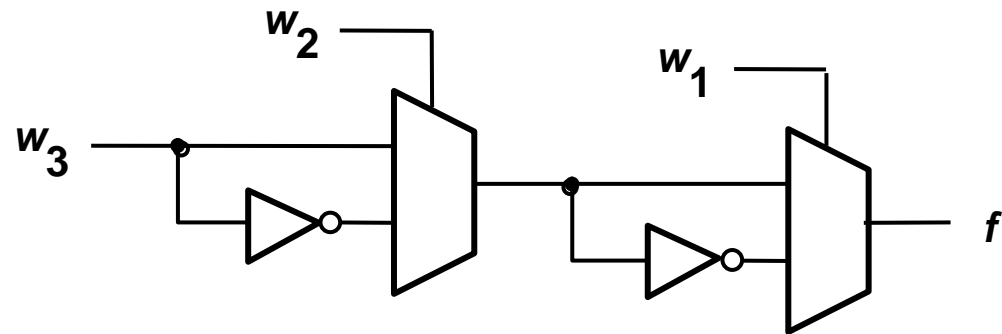
w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1



XOR3 implementada com 2 MUX 2:1

w_1	w_2	w_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$w_2 \oplus w_3$ (rows 2-5)
 $\overline{w_2 \oplus w_3}$ (rows 6-9)





Teorema de expansão de Shannon

- Qualquer função booleana pode ser escrita:

$$f(w_1, \dots, w_k, \dots, w_n) = \overline{w_k} \cdot f(w_1, \dots, 0, \dots, w_n) + w_k \cdot f(w_1, \dots, 1, \dots, w_n)$$

- Prova direta, substituindo $w_k = 0$ e depois $= 1$
- Função também pode ser expandida em 4 termos

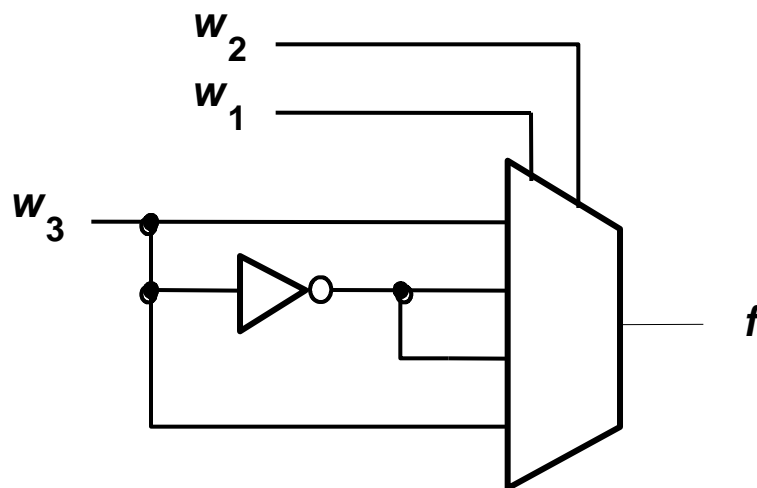
$$f(w_1, \dots, w_j, \dots, w_k, \dots, w_n) = \overline{w_j} \cdot \overline{w_k} \cdot f(w_1, \dots, 0, \dots, 0, \dots, w_n) + \overline{w_j} \cdot w_k \cdot f(w_1, \dots, 0, \dots, 1, \dots, w_n) + w_j \cdot \overline{w_k} \cdot f(w_1, \dots, 1, \dots, 0, \dots, w_n) + w_j \cdot w_k \cdot f(w_1, \dots, 1, \dots, 1, \dots, w_n)$$

- Exercício: aplicar o teorema nas soluções anteriormente feitas intuitivamente



XOR3 implementada com 1 MUX 4:1

w_1	w_2	w_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$f(w_1, w_2, w_3) =$$

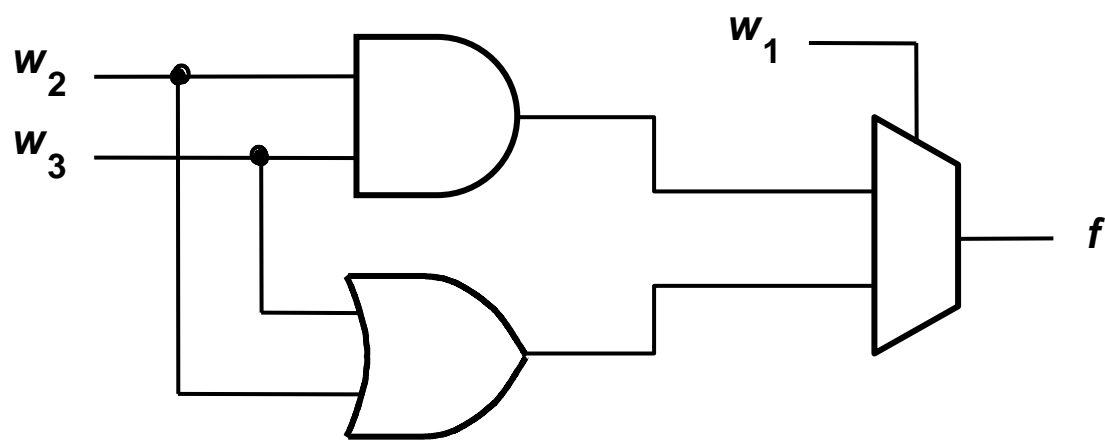
$$\overline{w_1} \cdot \overline{w_2} \cdot \underbrace{f(0, 0, w_3)}_{w_3} + \overline{w_1} \cdot w_2 \cdot \underbrace{f(0, 1, w_3)}_{\overline{w_3}} + w_1 \cdot \overline{w_2} \cdot \underbrace{f(1, 0, w_3)}_{\overline{w_3}} + w_1 \cdot w_2 \cdot \underbrace{f(1, 1, w_3)}_{w_3}$$



Exemplo: maioria de uns

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	$w_2 w_3$
1	$w_2 + w_3$



- Expandir em w_1 e posteriormente em $w_1 w_2$

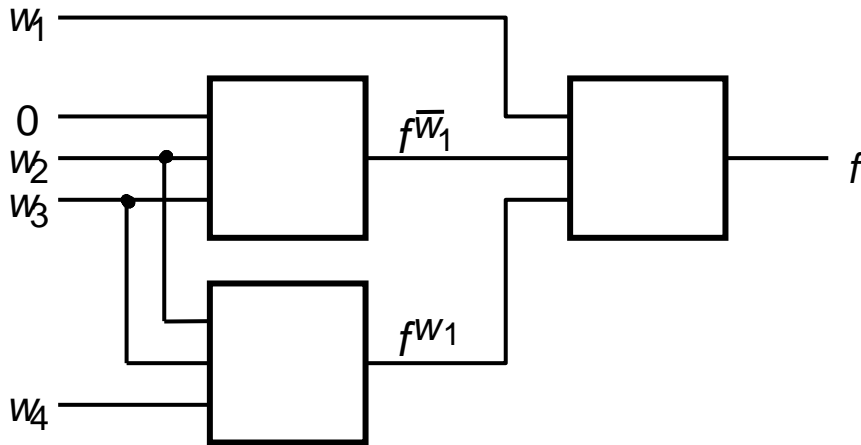


Fazer outros exemplos

- Exemplo 6.5: $f = \overline{w_1} \cdot w_3 + w_2 \cdot \overline{w_3}$
 - expandir em w_1 , w_2 e w_3 e procurar menor custo
- Exemplo 6.6: $f = \overline{w_1} \cdot \overline{w_3} + w_1 \cdot w_2 + w_2 \cdot w_3$
 - a) expandir em w_1
 - b) implementar com MUX 4:1 (expandir w_1 e w_2)
 - comparar os custos de a e b
- Exemplo 6.7: $f = w_1 \cdot w_2 + w_1 \cdot w_3 + w_2 \cdot w_3$
 - circuito maioria
 - implementar somente com MUXs 2:1
 - comparar custo com slide anterior

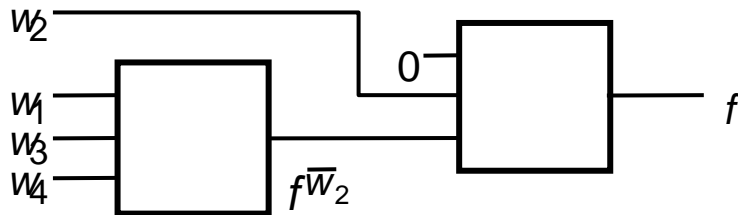
Implementação com LUTs

$$f = \overline{w_2} \cdot w_3 + \overline{w_1} \cdot w_2 \cdot \overline{w_3} + w_2 \cdot \overline{w_3} \cdot w_4 + \overline{w_1} \cdot \overline{w_2} \cdot \overline{w_4}$$



(a) Usando três 3-LUTs

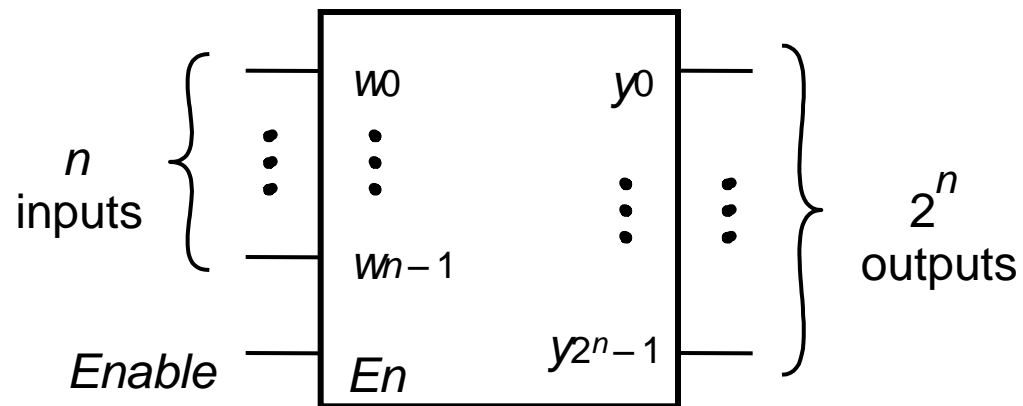
expansão apenas em w_1



(b) Usando duas 3-LUTs

expansão apenas em w_2

Decodificadores

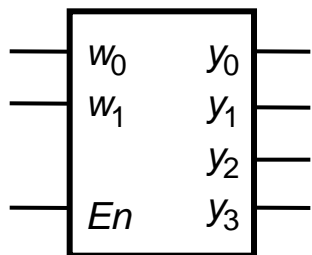


Decodificador n -to- 2^n

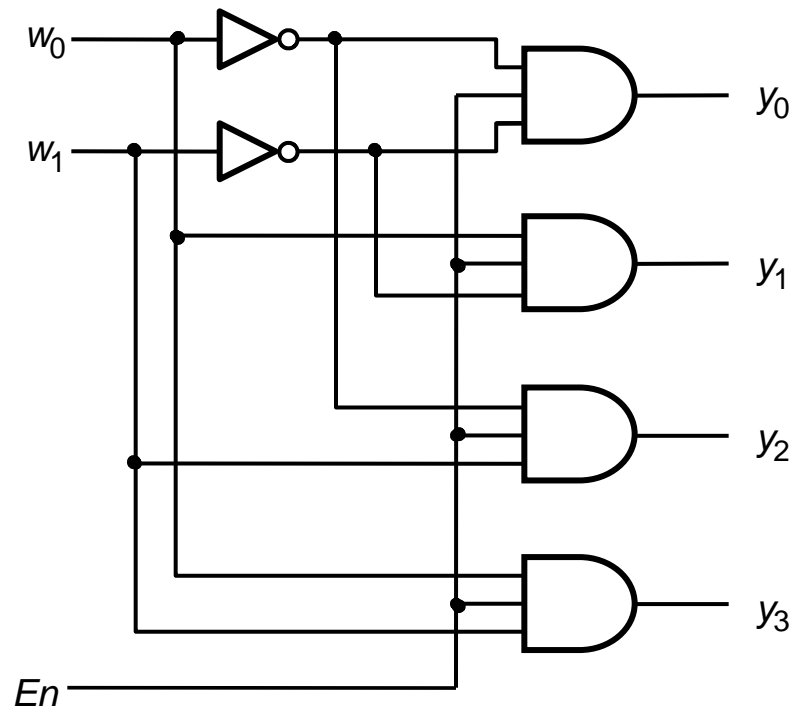
Decod. 2:4

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Tabwela verdade

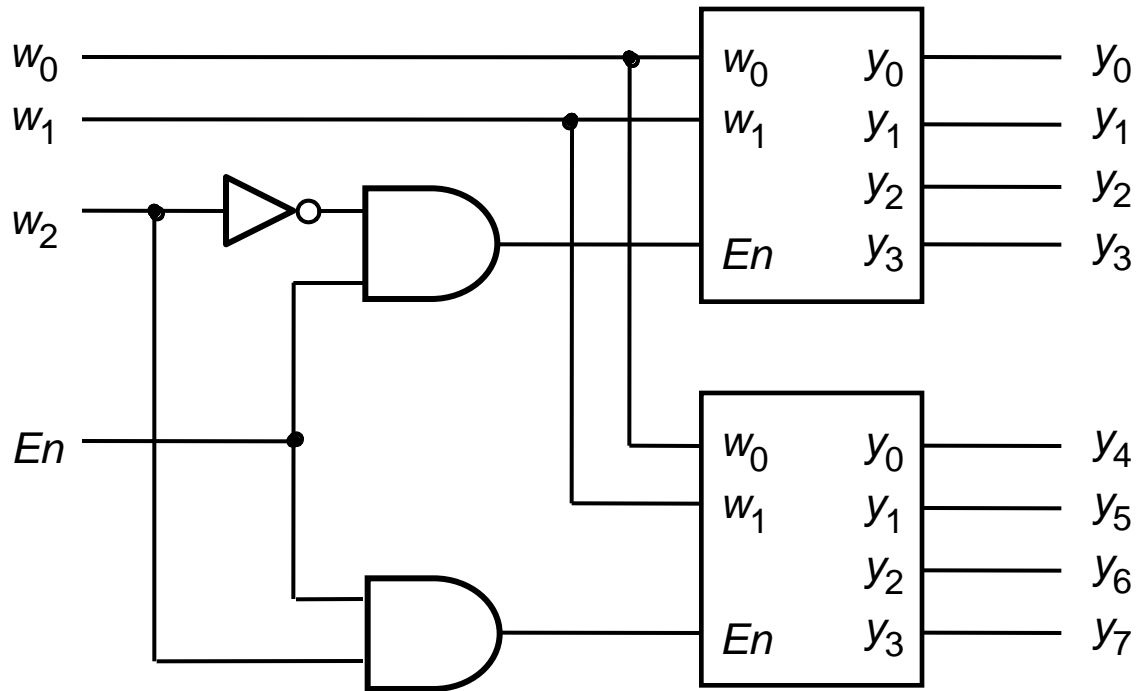


(b) Símbolo gráfico

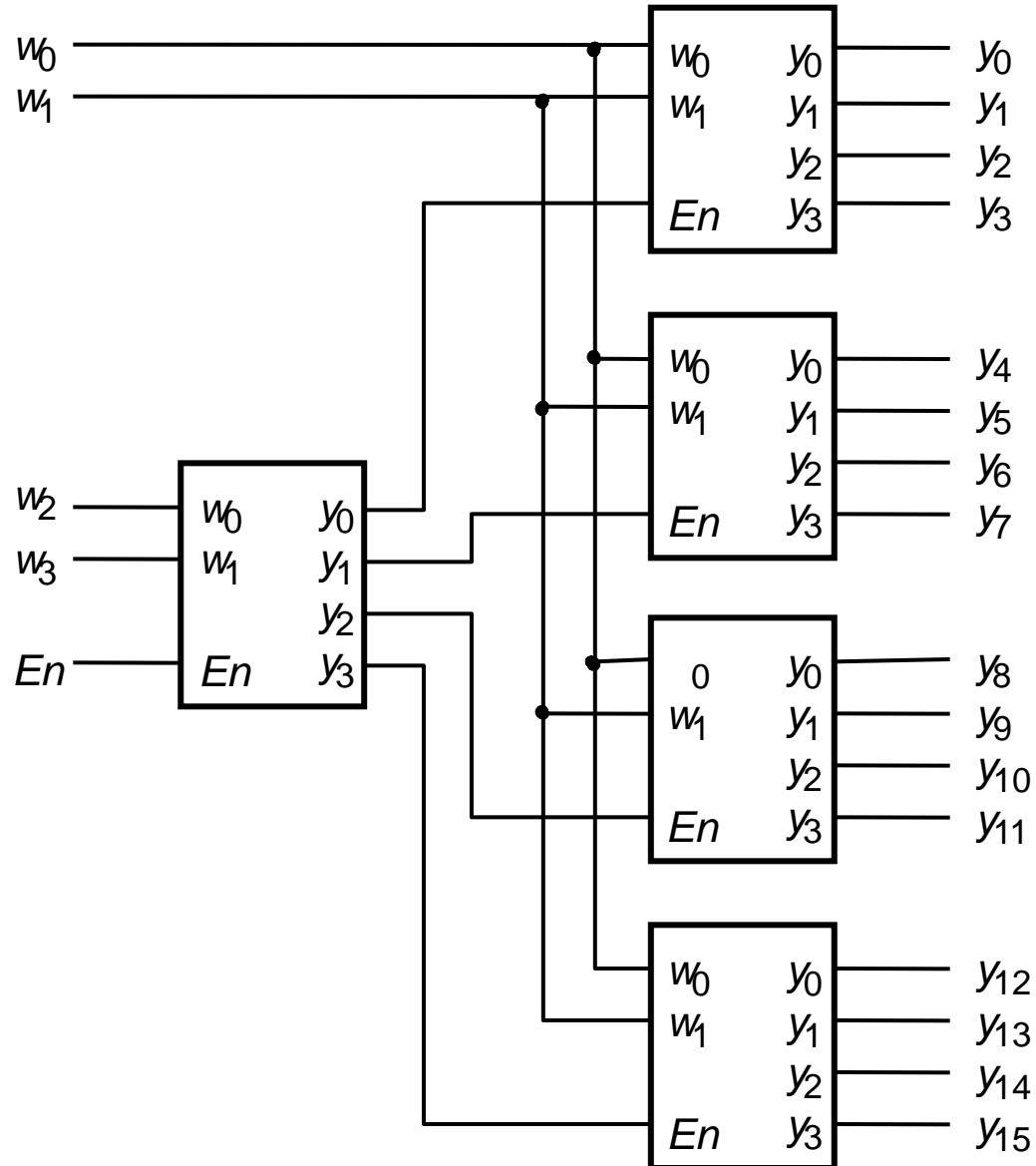


(c) Circuito lógico

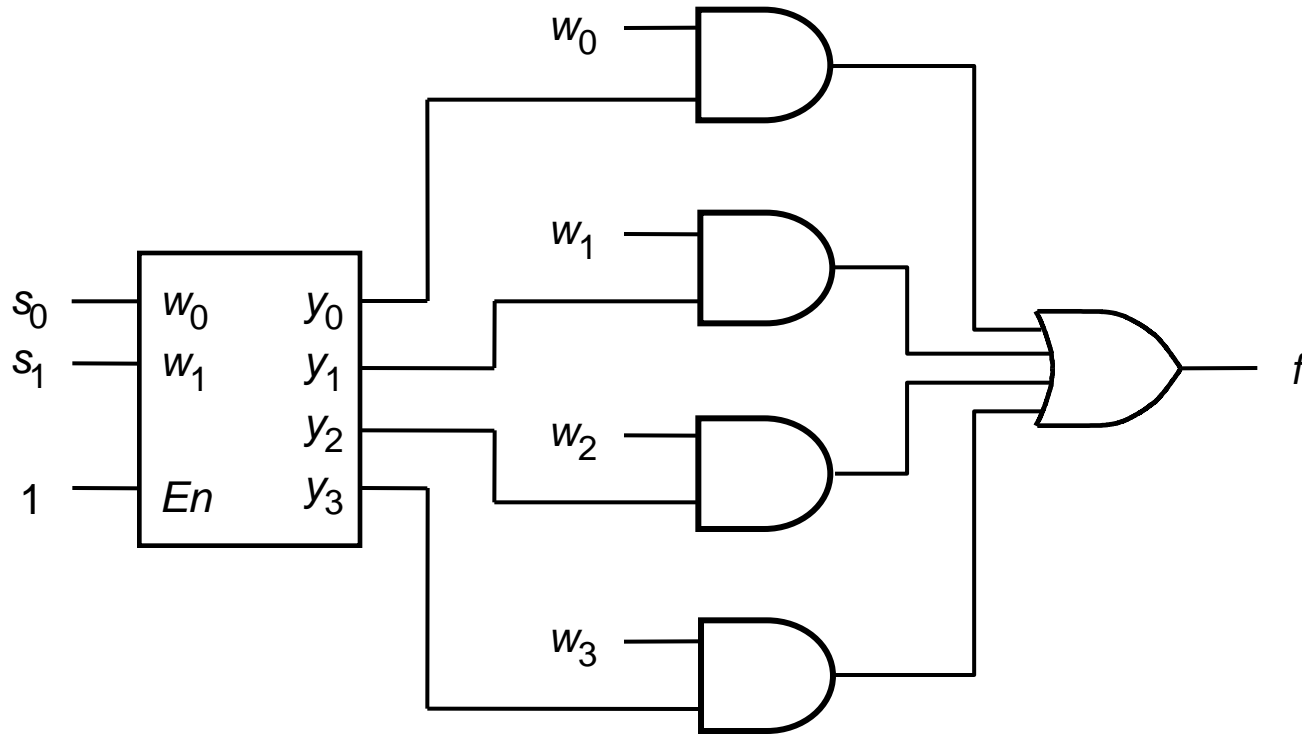
Um Decod. 3:8 usando decod. 2:4



Decod. 4:16 usando árvore de decod.

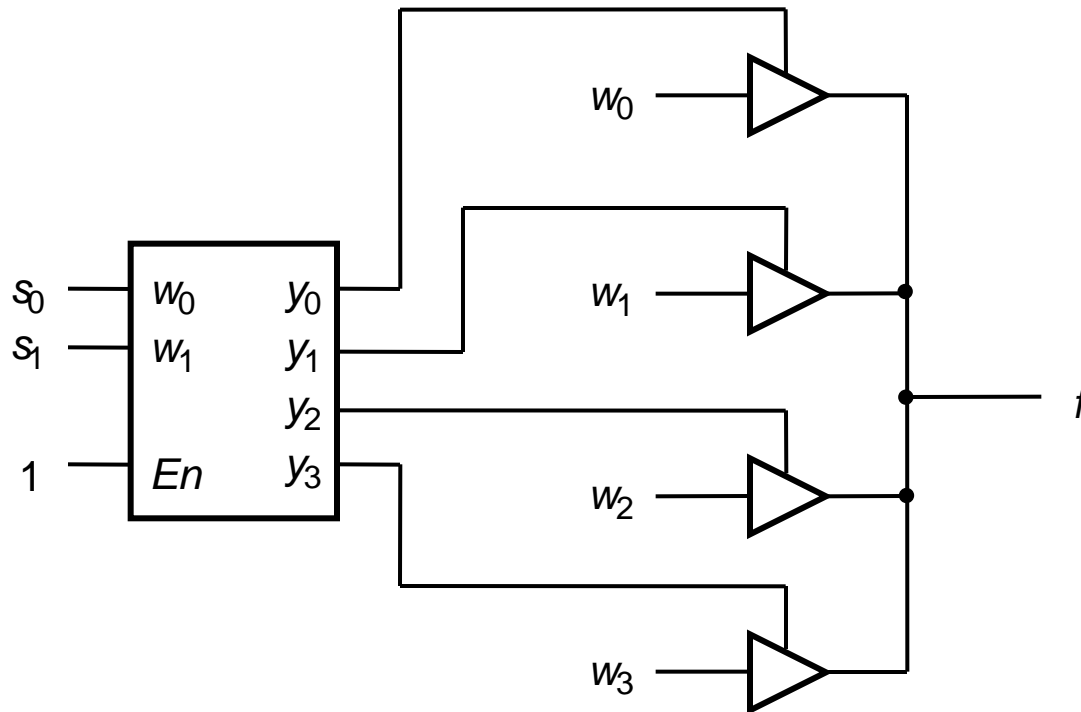


Mux implementado a partir de decoder



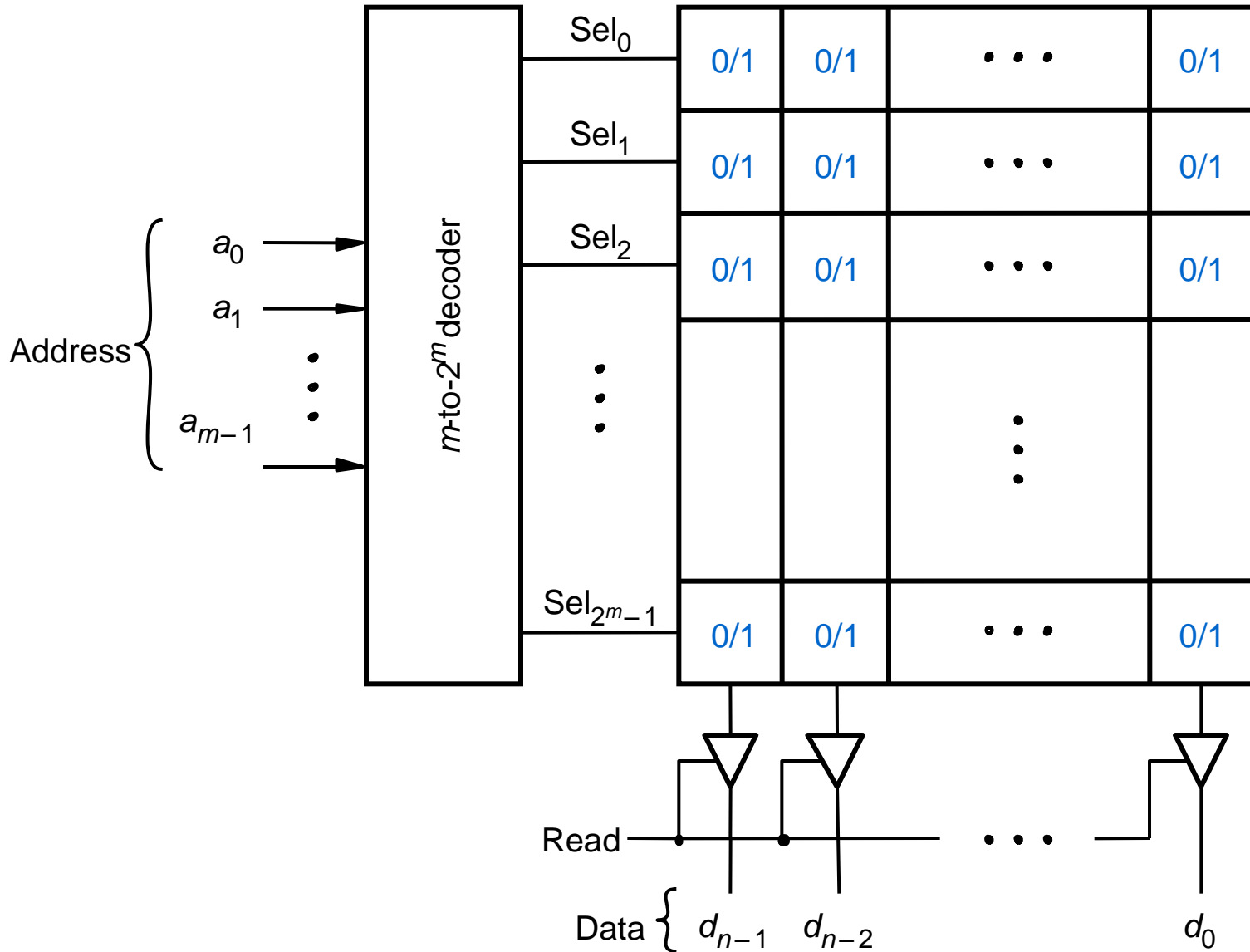
exemplo 6.9

Mux implementado c/ decoder e buffers tri-state

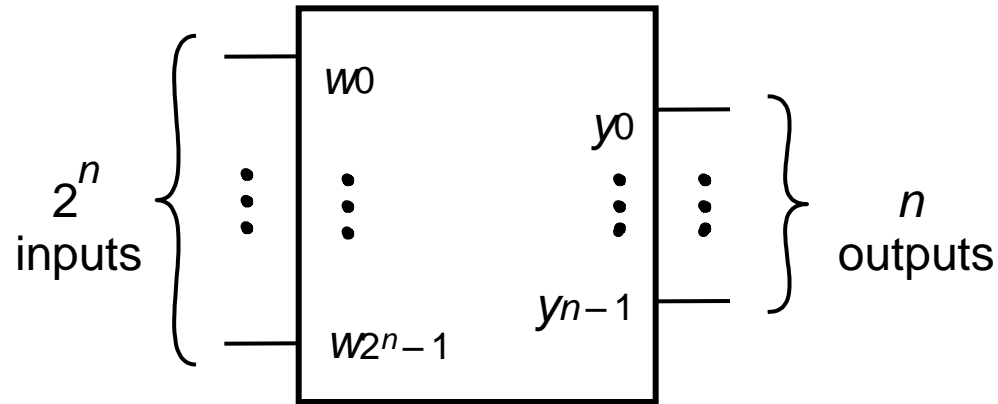


exemplo 6.10

Demultiplexador/Decoder (expl 6.11)



Codificadores

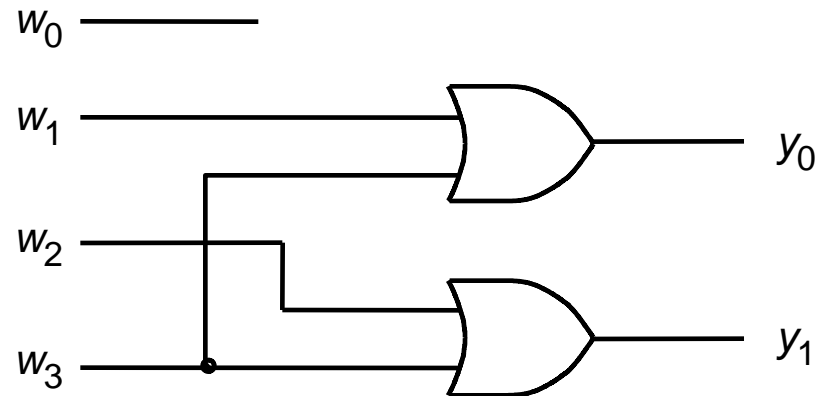


Codificador 2ⁿ-to-n

Codificador 4:2

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Tabela verdade



(b) Circuito

Codificador de prioridade

"pedidos"				"grants"				"codificação"		
w_3	w_2	w_1	w_0	g_3	g_2	g_1	g_0	y_1	y_0	z
0	0	0	0	0	0	0	0	d	d	0
0	0	0	1	0	0	0	1	0	0	1
0	0	1	x	0	0	1	0	0	1	1
0	1	x	x	0	1	0	0	1	0	1
1	x	x	x	1	0	0	0	1	1	1

- Função $y=f(w)$ e $z=f(w)$ poderia ser sintetizada como no cap.4. Mas é mais simples observar que:

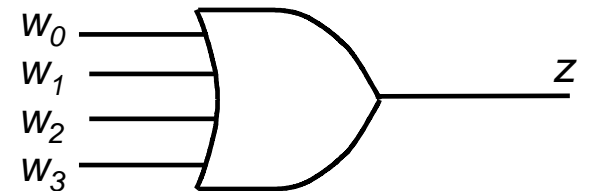
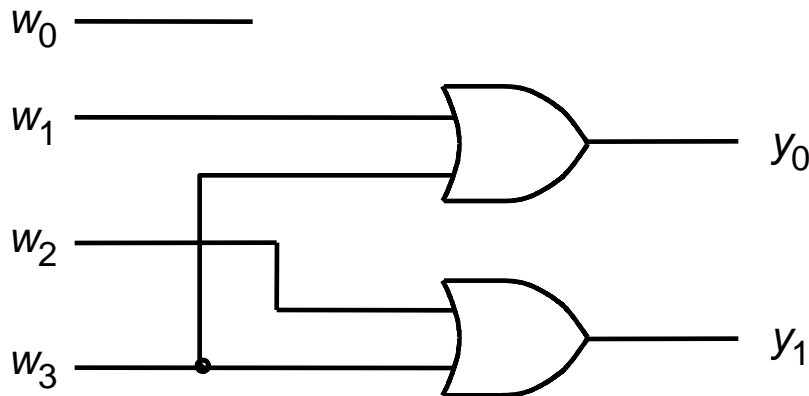
$$g_3 = w_3 \quad g_2 = \overline{w_3} \cdot w_2 \quad g_1 = \overline{w_3} \cdot \overline{w_2} \cdot w_1 \quad g_0 = \overline{w_3} \cdot \overline{w_2} \cdot \overline{w_1} \cdot w_0$$

- y_1 e y_0 podem ser gerados com um codificador 4:2 (próximo slide)

Implementação do codificador prioridade

confrontar
com slides
anteriores

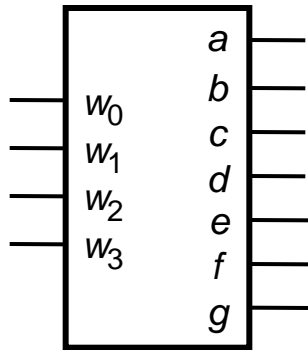
W_3	W_2	W_1	W_0	Y_1	Y_0	Z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



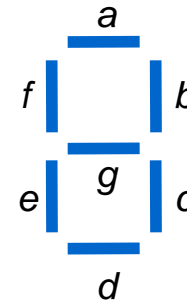
Conversores de código

- Transformam código de entrada (m bits) em códigos de saída (n bits)
 - conversor binário \rightarrow 7 segmentos
 - conversor binário \rightarrow BCD

Conversor Bin \rightarrow BCD (7 segmentos)



(a) Conversor de código



(b) Display de 7 segmentos

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Tabela verdade

Implementação convencional

Segment a

	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	0	0

Segment f

	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	1	1	0	0
10	1	0	0	0

Segment g

	00	01	11	10
00	1	0	1	0
01	1	0	0	0
11	0	1	0	0
10	0	0	0	0

Segment e

	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	1	0	0
10	0	0	0	0

Segment d

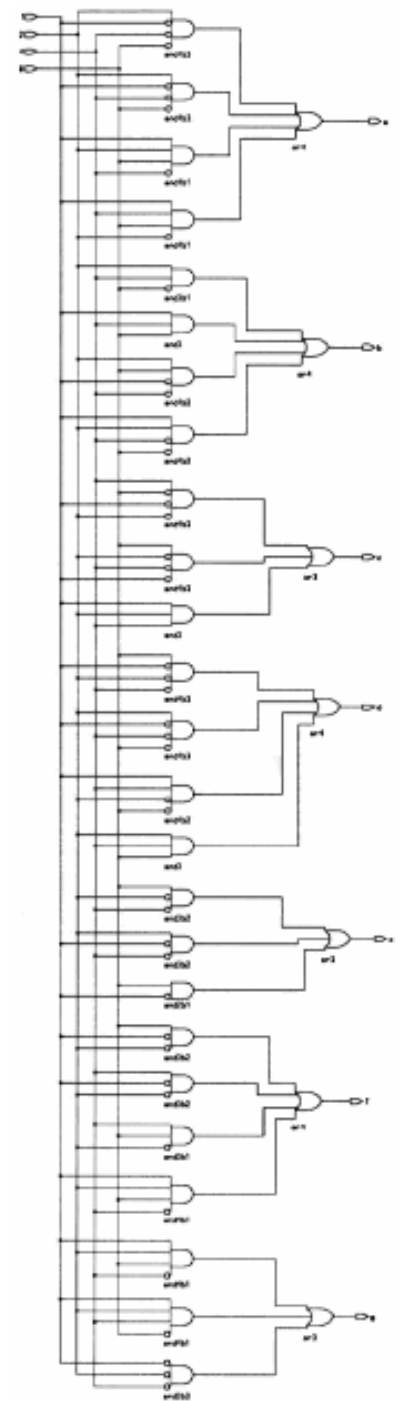
	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	1	1	0
10	0	0	0	1

Segment b

	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	0	0	1	1
10	0	1	1	0

Segment c

	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	1	0
10	1	0	1	0



Comparador de 4 bits

