



Sistemas de Computação

Paulo Santos

Representação da Informação

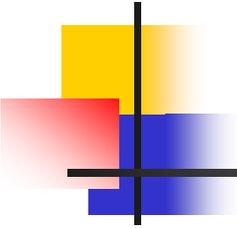


UNIÃO EUROPEIA

Fundo Social Europeu

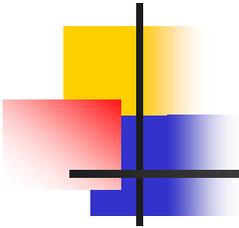
prime
Programa de Incentivos à
Modernização da Economia

IQF Instituto para a Qualidade
na Formação, I.P.



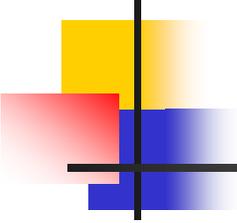
Aula 1

- Representação da informação
 - Unidade básica de informação - bit
 - Sistemas de numeração
 - Representação de números inteiros
 - Conversões entre bases numéricas
 - Operações aritméticas
 - Códigos alfanuméricos
 - ASCII
 - Unicode



Representação da Informação

- Nos computadores, a informação é representada por sinais eléctricos
 - Tensão alta – e.g. 3 a 5.5 V – **HIGH**
 - Tensão baixa – e.g. -0.5 a 2 V – **LOW**
- A estes níveis correspondem 2 valores lógicos
 - 1 (**Verdadeiro**), habitualmente associado a **HIGH**
 - 0 (**Falso**), habitualmente associado a **LOW**
- Cada dígito binário (0 ou 1) designa-se por **bit**
- 8 **bits** = 1 **Byte**

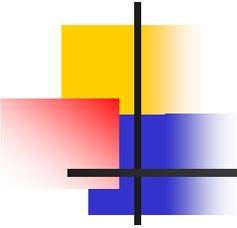


Representação da Informação

- O bit é a unidade básica de informação
- Muita informação \Rightarrow Medidas grandes
 - Kilo (K) – $1K = 2^{10} = 1024$
 - Mega (M) – $1M = 2^{20}$
 - Giga (G) – $1G = 2^{30}$
 - Tera (T) – $1T = 2^{40}$
 - ...

Exemplo:

$$2 \text{ MBytes} = 2 \times 2^{20} \text{ Bytes} = 2^{24} \text{ bits} = 16\,777\,216 \text{ bits}$$



Sistemas de Numeração

- Decimal (base 10)

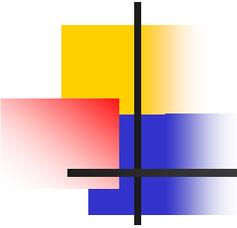
- 10 dígitos – 0 a 9

$$562.3 = 5 \times 10^2 + 6 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1}$$

- Binário (base 2)

- 2 dígitos – 0 e 1

$$\begin{aligned} 1010.01 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = \\ &= (10.25)_{10} \end{aligned}$$



Sistemas de Numeração

- Conversão base 10 \Rightarrow base 2
 - Subtrai-se sucessivamente a maior potência de 2 possível

Ex: $(41)_{10}$

$$41 - 32 = 9 \quad 2^5$$

$$9 - 8 = 1 \quad 2^3$$

$$1 - 1 = 0 \quad +2^0$$

41

$$(41)_{10} = (1\ 0\ 1\ 0\ 0\ 1)_2$$

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

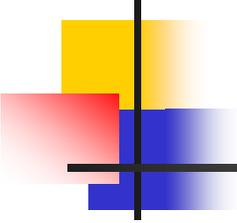
Sistemas de Numeração

- Conversão base 10 \Rightarrow base 2 (outro método)
 - Divide-se sucessivamente por 2 e anota-se o resto

Ex: $(41)_{10}$

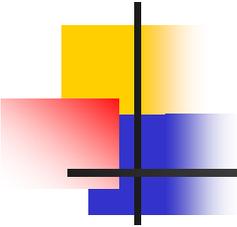
Divisão inteira	Resto	
$41 \div 2 = 20$	1	Bit menos significativo
$20 \div 2 = 10$	0	
$10 \div 2 = 5$	0	
$5 \div 2 = 2$	1	
$2 \div 2 = 1$	0	
$1 \div 2 = 0$	1	

$$(41)_{10} = (101001)_2$$



Sistemas de Numeração

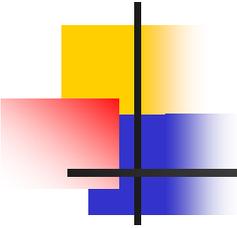
- Hexadecimal (base 16) e Octal (base 8)
 - Representação fácil de quantidades binárias
 - Octal – 8 dígitos – 0 a 7
 - Hexadecimal – 16 dígitos 0 a 9; A a F
 - Exemplos:
 - $(25)_8 = 2 \times 8^1 + 5 \times 8^0 = (21)_{10}$
 - $(B3)_{16} = 11 \times 16^1 + 3 \times 16^0 = (179)_{10}$



Sistemas de Numeração

- Inteiros de 0 a 15, em diferentes bases

Decimal	Binário	Octal	Hexadecimal
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F



Sistemas de Numeração

- Conversão base 2 \Rightarrow base 8

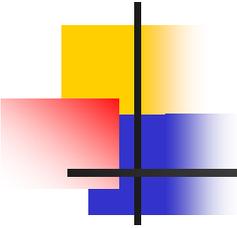
- Grupos de 3 bits

$$(101001)_2 = (51)_8 \quad \begin{array}{cc} \overbrace{5} & \overbrace{1} \\ 101 & 001 \end{array}$$

- Conversão base 2 \Rightarrow base 16

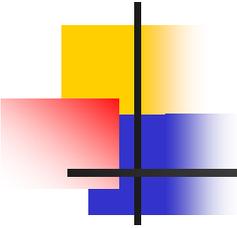
- Grupos de 4 bits

$$(101001)_2 = (29)_{16} \quad \begin{array}{cc} \overbrace{2} & \overbrace{9} \\ 0010 & 1001 \end{array}$$



Números Inteiros

- Gama de variação
 - Depende do hardware
 - 8 bits (byte): [0; 255] ou [-128; 127]
 - 16 bits: [0; 65535] ou [-32768; 32767]
 - 32 bits: [0; 4294967295] ou [-2147483648; 2147483647]
 - 64 bits



Operações Aritméticas

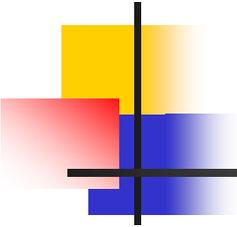
- Mesmas regras da base 10

- Adição

10110	transporte
10110	parcela 1
+10011	parcela 2
<hr/>	
101001	soma

- Subtração

0011	transporte
10110	diminuendo
-10011	diminuidor
<hr/>	
00011	diferença



Operações Aritméticas

- Multiplicação

1101	multiplicando
×101	multiplicador
<hr/>	
1101	
0000	
1101	
<hr/>	
1000001	produto

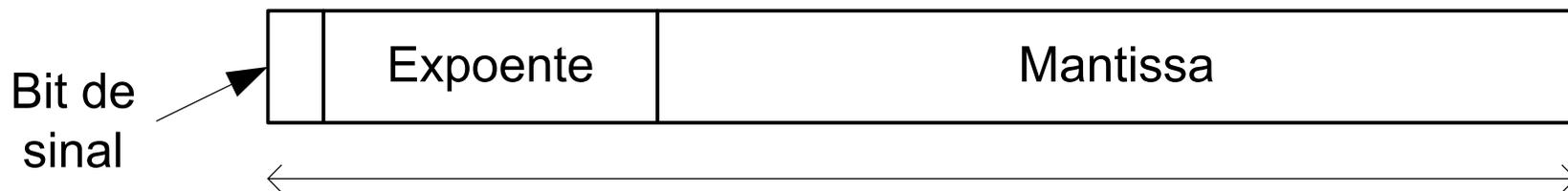
Números Reais

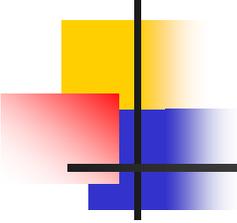
- Representados na forma

$$\pm 1.\textit{mantissa} \times 2^{\pm \textit{expoente}}$$

- Exemplo:

- Números reais de precisão simples
(norma IEEE 754 – usada nos nossos computadores)





Códigos Alfanuméricos

- ASCII

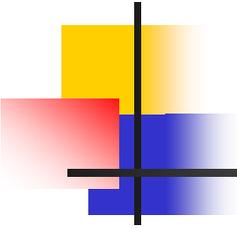
American Standard Code for Information Interchange

- 7 bits + 1 (opcional)
- Pequeno conjunto de caracteres
 - Caracteres de controlo
 - Sinais ortográficos
 - Algarismos
 - Letras maiúsculas e minúsculas (A...Z; a...z)
 - Sinais algébricos

Códigos Alfanuméricos

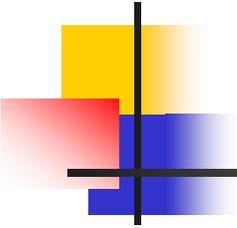
- Tabela de códigos ASCII

$B_4B_3B_2B_1$	$B_7B_6B_5$							
	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL



Códigos Alfanuméricos

- **UNICODE** (norma ISO/IEC 10646)
 - 32 bits
 - Grafismos de todo o mundo
 - Alfabeto latino, cirílico, grego, etc.
 - Caracteres chineses, japoneses, etc.
 - Engloba códigos que utilizam menos bits
 - UTF-8 (8 bits)
 - compatível com ASCII
 - UTF-16 (16 bits)
 - utilizado em muitos programas
 - permite a representação de caracteres como à, ã, é, ç, etc.



Sumário (Aulas 2 e 3)

- Lógica combinatória
 - Álgebra de *Boole*
 - Operações lógicas AND, OR e NOT
 - Portas lógicas
 - Propriedades da álgebra de *Boole*
 - Simplificação de funções lógicas
 - Mapas de *Karnaugh*
 - Forma normalizada – soma de produtos
 - Termos mínimos
 - Síntese de funções lógicas
 - Implicantes de uma função lógica



Sistemas de Computação

Paulo Santos

Álgebra de *Boole*

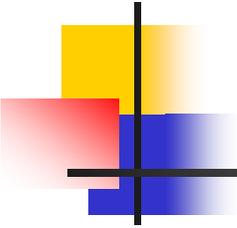


UNIÃO EUROPEIA

Fundo Social Europeu

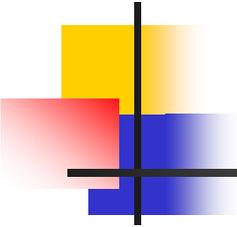
prime
Programa de Incentivos à
Modernização da Economia





Lógica Binária

- Recordando a aula anterior...
 - Valores lógicos
 - 0 (Falso)
 - 1 (Verdadeiro)
- Definem-se 3 operações básicas
 - **Conjunção** – “E”, “AND”, representada por ‘.’
 - **Disjunção** – “OU”, “OR”, representada por ‘+’
 - **Negação** – “NÃO”, “NOT”, representada pela barra horizontal sobre a variável ou por ‘~’



Lógica Binária

- Tabelas de verdade

AND		
X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

$$F = X \cdot Y$$

OR		
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

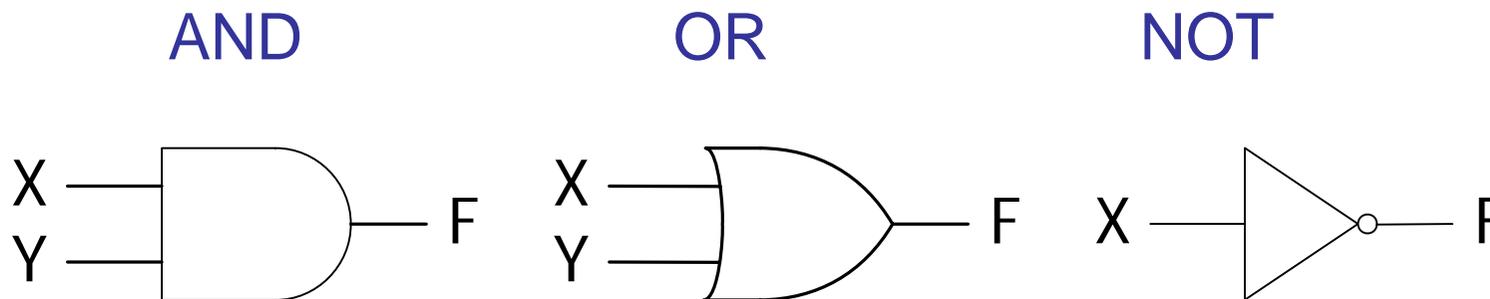
$$F = X + Y$$

NOT	
X	F
0	1
1	0

$$F = \overline{X}$$

Portas Lógicas

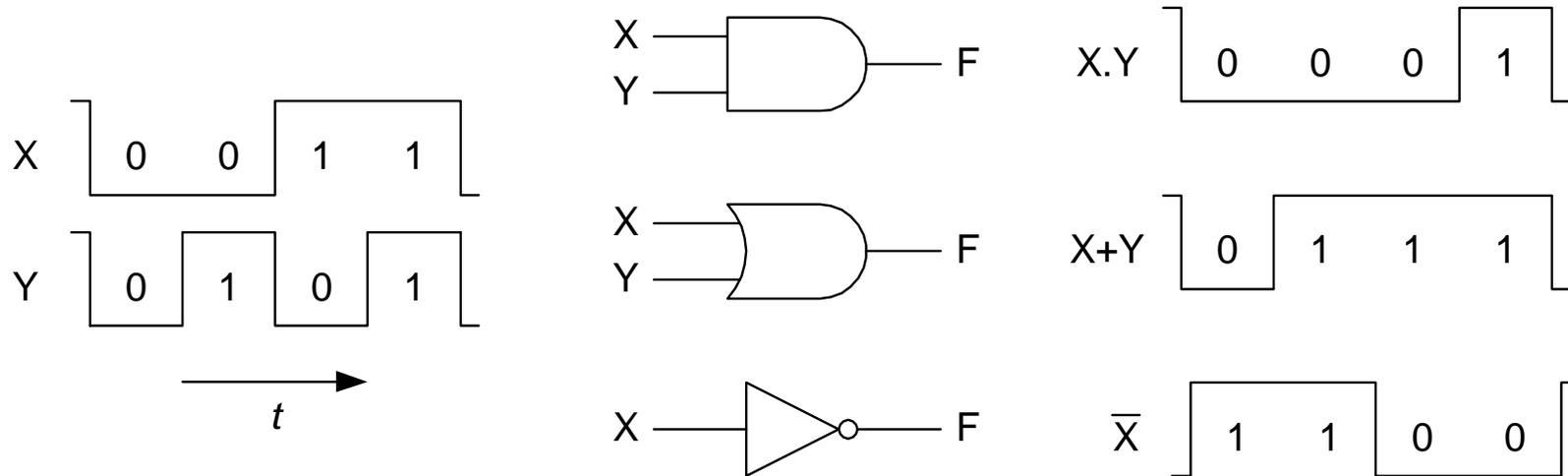
- Cada operação lógica tem um circuito eléctrico correspondente



- Estes componentes designam-se por **portas lógicas** (*logic gates*)

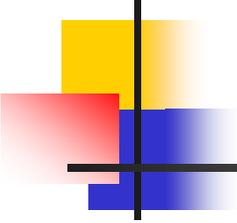
Portas Lógicas

■ Evolução temporal



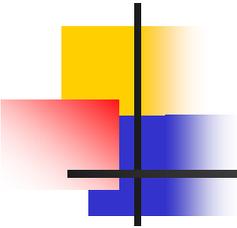
Na realidade existe um atraso temporal entre variações à entrada e consequente variação na saída

(veremos esse facto mais à frente)



Álgebra de *Boole*

- Representação de funções lógicas por equações
 - Exemplo: $F = \bar{X} + YZ$
- A partir da função lógica obtém-se:
 - **Tabela de verdade** – valores lógicos da função para todas as combinações de entradas possíveis
 - **Esquema do circuito** – com as portas lógicas e respectivas ligações



Álgebra de *Boole*

- Tabela de verdade

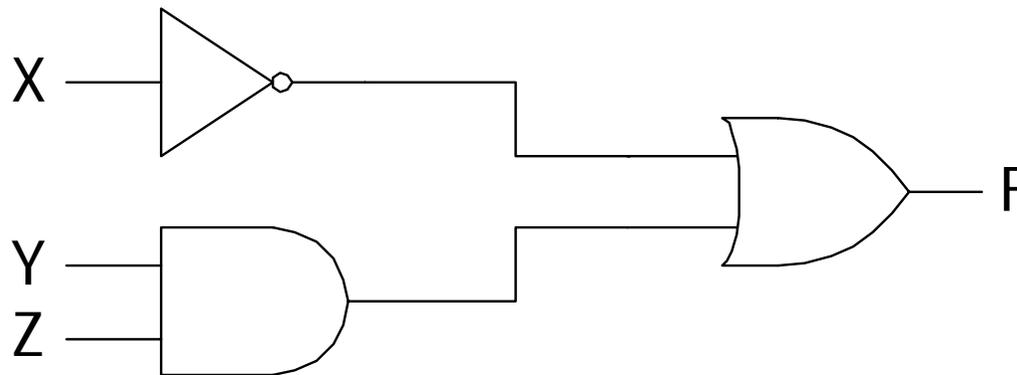
$$F = \bar{X} + YZ$$

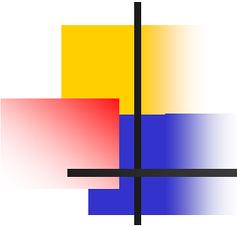
X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Álgebra de *Boole*

- Esquema do circuito

$$F = \bar{X} + YZ$$



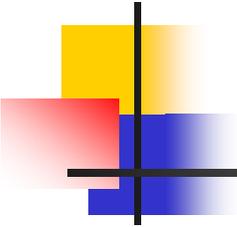


Álgebra de *Boole*

■ Propriedades

■ AND

- $X \cdot 1 = X$ (elemento neutro)
- $X \cdot 0 = 0$ (elemento absorvente)
- $X \cdot X = X$
- $X \cdot \bar{X} = 0$
- $X \cdot Y = Y \cdot X$ (comutativa)
- $X(YZ) = (XY)Z$ (associativa)
- $X + (YZ) = (X + Y) \cdot (X + Z)$ (distributiva)

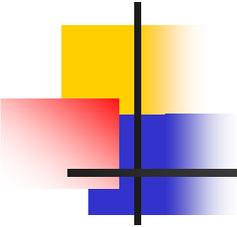


Álgebra de *Boole*

- Propriedades

- OR

- $X+0 = X$ (elemento neutro)
 - $X+1 = 1$ (elemento absorvente)
 - $X+X = X$
 - $X+\bar{X} = 1$
 - $X+Y = Y+X$ (comutativa)
 - $X+(Y+Z) = (X+Y)+Z$ (associativa)
 - $X.(Y+Z) = (XY) + (XZ)$ (distributiva)



Álgebra de *Boole*

- NOT

$$\overline{\overline{X}} = X$$

- Leis de DeMorgan

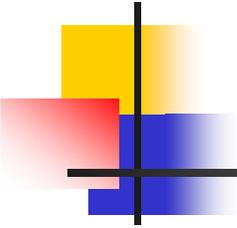
$$\overline{X+Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

- Teorema do consenso

$$XY + \overline{X}Z + YZ = XY + \overline{X}Z$$

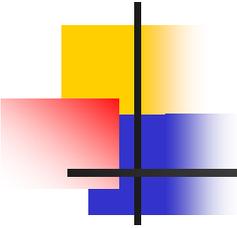
$$(X+Y)(\overline{X}+Z)(Y+Z) = (X+Y)(\overline{X}+Z)$$



Álgebra de *Boole*

- Aplicação das propriedades
 - Simplificação de equações
 - Exemplo

$$\begin{aligned}F &= XY\bar{Z} + XYZ + X = XY(\bar{Z} + Z) + X \\ &= XY \cdot 1 + X \\ &= XY + X \\ &= XY + X \cdot 1 \\ &= X(Y + 1) \\ &= X \cdot 1 \\ &= X\end{aligned}$$



Álgebra de *Boole*

- Outro exemplo
 - Simplificar a expressão

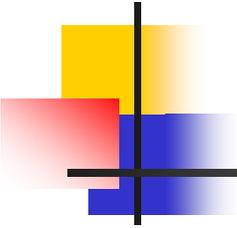
$$\begin{aligned} F &= X + \bar{X}Y + X(\bar{X} + \bar{Y}) = X + \bar{X}Y + X\bar{X} + X\bar{Y} \\ &= X + \bar{X}Y + 0 + X\bar{Y} \\ &= X + \bar{X}Y + X\bar{Y} \\ &= (X + \bar{X})(X + Y) + X\bar{Y} \\ &= 1 \cdot (X + Y) + X\bar{Y} \\ &= X + Y + X\bar{Y} \\ &= X(1 + \bar{Y}) + Y \\ &= X + Y \end{aligned}$$



Sistemas de Computação

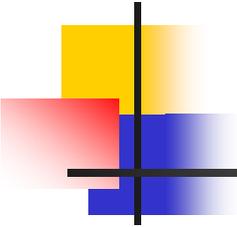
Paulo Santos

Mapas de *Karnaugh*



Formas Normalizadas

- Cada função lógica pode ser escrita de diversas maneiras diferentes
- É habitual escrever a função utilizando uma **forma normalizada**
 - Simplificação mais fácil
 - Implementação mais eficiente do circuito
- Uma das formas normalizadas consiste em escrever a função como uma **soma de produtos**
 - é sempre possível, qualquer que seja a função lógica



Formas Normalizadas

- Exemplo:

- Considera a função $F(X,Y,Z)$ definida pela tabela de verdade

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

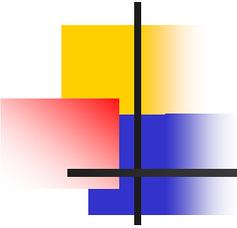
A partir da tabela de verdade pode-se escrever F como:

$$F = \bar{X}\bar{Y}\bar{Z} + \bar{X}YZ + XYZ$$

Simplificando:

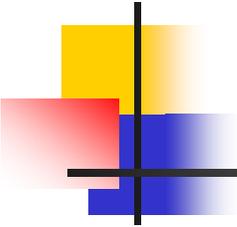
$$F = \bar{X}Y + YZ$$

Em ambos os casos F está escrita como uma **soma de produtos**



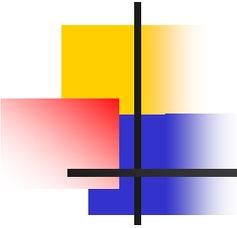
Formas Normalizadas

- **Termo mínimo**
 - produto lógico em que todas as variáveis da função aparecem exactamente uma vez (complementadas ou não)
- A cada **termo mínimo** corresponde uma dada combinação de entradas
 - por exemplo,
 $\bar{X}YZ$ corresponde a $X=0$; $Y=1$ e $Z=1$
- n variáveis $\Rightarrow 2^n$ **termos mínimos**



Termos Mínicos

X	Y	Z	Termo mínimo	Símbolo
0	0	0	$\bar{X} \bar{Y} \bar{Z}$	m_0
0	0	1	$\bar{X} \bar{Y} Z$	m_1
0	1	0	$\bar{X} Y \bar{Z}$	m_2
0	1	1	$\bar{X} Y Z$	m_3
1	0	0	$X \bar{Y} \bar{Z}$	m_4
1	0	1	$X \bar{Y} Z$	m_5
1	1	0	$X Y \bar{Z}$	m_6
1	1	1	$X Y Z$	m_7



Termos Mínimos

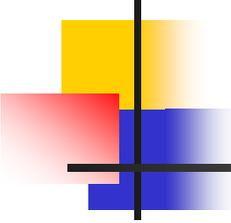
- Uma função pode ser definida como uma soma lógica de termos mínimos

- Exemplo

$$F(X, Y, Z) = \sum (m_2, m_3, m_4, m_7)$$

É o mesmo que

$$F = \overline{X}Y\overline{Z} + \overline{X}YZ + X\overline{Y}\overline{Z} + XYZ$$

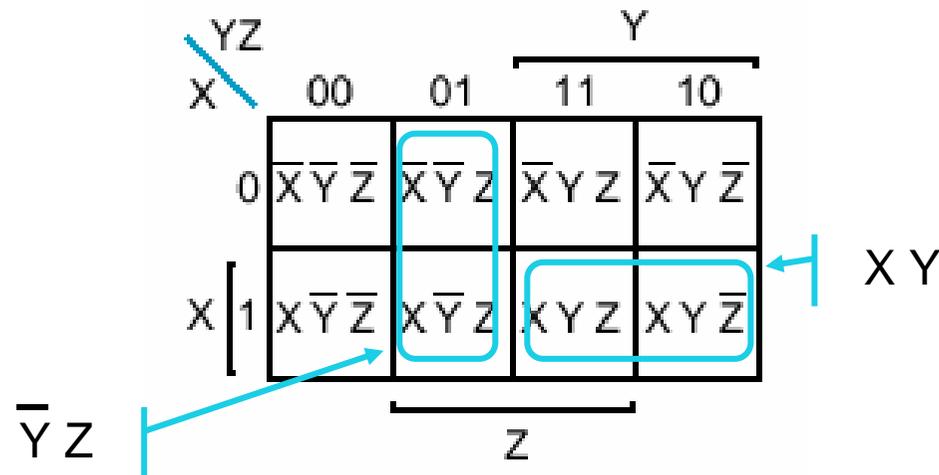


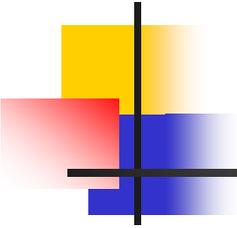
Mapas de *Karnaugh*

- Ferramenta útil para simplificação de funções e projecto de circuitos
- Geralmente utilizados quando se pretende obter uma função lógica a partir da tabela de verdade
- Representam-se no mapa os termos mínimos da função (com valor 1 ou 0)
- Obtém-se as funções lógicas através do agrupamento de termos mínimos com valor '1'

Mapas de *Karnaugh*

- Os termos mínimos são organizados de modo a que apenas mude um bit entre quadrados adjacentes (código binário reflectido)



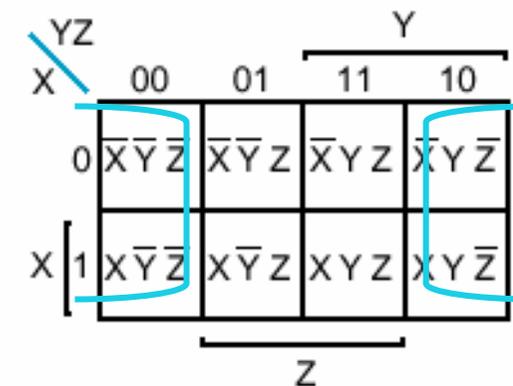
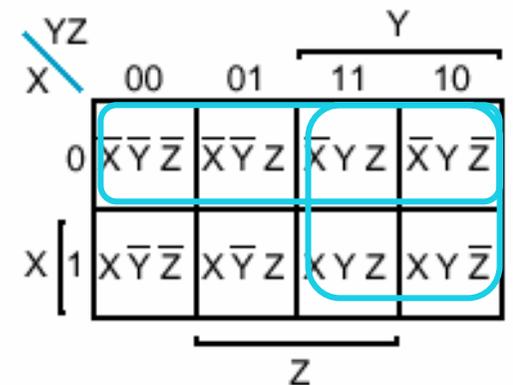
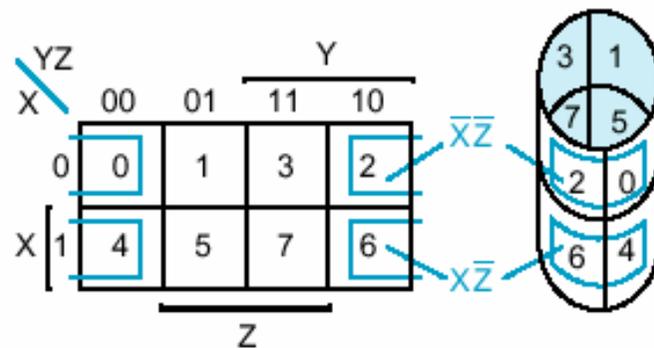


Mapas de *Karnaugh*

- Como agrupar os termos mínimos ?
 - As regras básicas são:
 - Só se podem agrupar termos mínimos adjacentes em que a função assume o valor lógico 1
 - Os agrupamentos são feitos segundo rectângulos ou quadrados sobre o mapa de *Karnaugh*
 - O número de termos mínimos dentro de cada agrupamento tem que ser uma potência de 2

Mapas de *Karnaugh*

- Exemplos de agrupamentos de termos mínimos

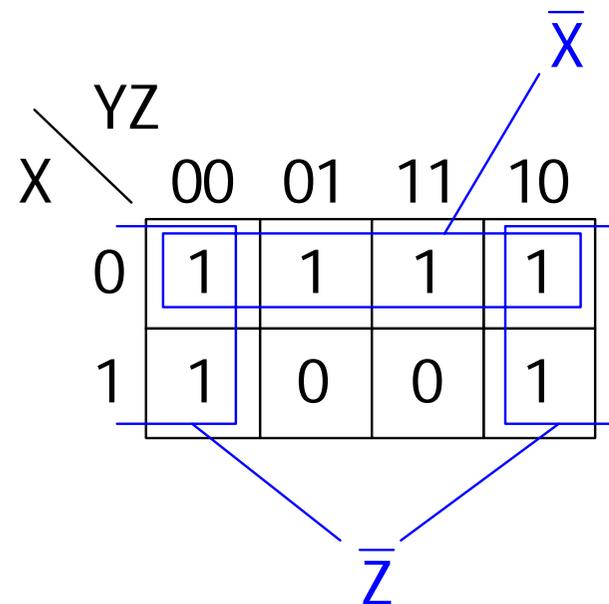


Mapas de *Karnaugh*

- Exemplo de simplificação de função

$$F = (X + Y) \cdot \bar{Z} + \bar{X}$$

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



A função simplifica-se para

$$F = \bar{X} + \bar{Z}$$

Mapas de *Karnaugh*

- Exemplo de obtenção dos termos produto

ZY		XW			
		00	01	11	10
XY	00	1	0	1	1
	01	0	0	1	1
	11	0	1	1	1
	10	1	0	1	1

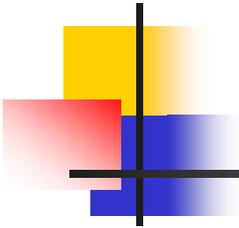
Diagram illustrating the Karnaugh map for the function F . The map is a 4x4 grid with variables X, Y, Z, W . The rows are labeled XY (00, 01, 11, 10) and the columns are labeled ZW (00, 01, 11, 10). The map shows the following values:

- Row 00: 1, 0, 1, 1
- Row 01: 0, 0, 1, 1
- Row 11: 0, 1, 1, 1
- Row 10: 1, 0, 1, 1

Groupings are shown with colored boxes and lines:

- A blue box highlights the cells (00, 11), (01, 11), (11, 11), and (10, 11), labeled Z .
- A red box highlights the cells (11, 01) and (11, 11), labeled $X Y W$.
- Green lines highlight the cells (00, 00), (10, 00), (00, 10), and (10, 10), labeled $\overline{Y} \overline{W}$.

$$F = XYW + \overline{Y} \overline{W} + Z$$



Mapas de *Karnaugh*

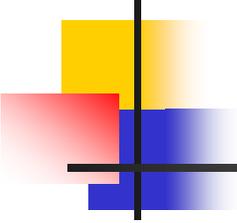
- Implicantes
 - Designa-se **implicante** de uma função a qualquer termo produto que assume o valor lógico 1 para todos os termos mínimos que o compõem
 - Se a simplificação de uma variável num implicante o transformar num termo produto não implicante, então esse implicante é um **implicante primo**

Mapas de *Karnaugh*

- Exemplos de implicantes e implicantes primos

X \ YZ	00	01	11	10
0	1	0	1	1
1	0	0	1	1

- Implicantes: $Y; \bar{X}\bar{Z}; \bar{X}Y; YZ; XY; \bar{X}\bar{Y}\bar{Z};$ etc.
- Implicantes primos: $Y; \bar{X}\bar{Z}$



Mapas de *Karnaugh*

- Implicantes primos
 - Se um termo mínimo estiver incluído apenas num implicante primo então esse implicante primo é **essencial**
 - Os implicantes primos a que a condição anterior não se aplica são implicantes primos **não-essenciais**

Mapas de *Karnaugh*

- Exemplos de implicantes primos essenciais e não essenciais

		ZW			
	XY	00	01	11	10
00		0	0	1	1
01		0	0	1	1
11		1	1	1	0
10		0	0	0	0

— Implicantes primos essenciais

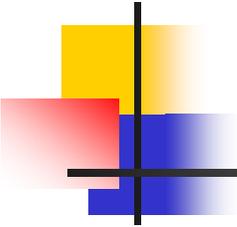
— Implicantes primos não-essenciais

Mapas de *Karnaugh*

- Quando uma determinada combinação de variáveis nunca ocorre, é possível definir a função como **não totalmente especificada**

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

- Os termos mínimos **não especificados** podem ser agrupados ao não, consoante dê mais jeito



Sumário (Aulas 2 e 3)

- Lógica combinatória
 - Álgebra de *Boole*
 - Operações lógicas AND, OR e NOT
 - Portas lógicas
 - Propriedades da álgebra de *Boole*
 - Simplificação de funções lógicas
 - Mapas de *Karnaugh*
 - Forma normalizada – soma de produtos
 - Termos mínimos
 - Síntese de funções lógicas
 - Implicantes de uma função lógica



Sistemas de Computação

Paulo Santos

Álgebra de *Boole*

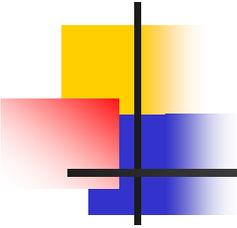


UNIÃO EUROPEIA

Fundo Social Europeu

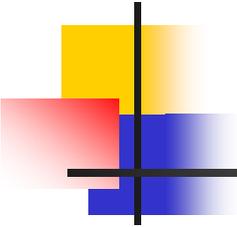
prime
Programa de Incentivos à
Modernização da Economia





Lógica Binária

- Recordando a aula anterior...
 - Valores lógicos
 - 0 (Falso)
 - 1 (Verdadeiro)
- Definem-se 3 operações básicas
 - **Conjunção** – “E”, “AND”, representada por ‘.’
 - **Disjunção** – “OU”, “OR”, representada por ‘+’
 - **Negação** – “NÃO”, “NOT”, representada pela barra horizontal sobre a variável ou por ‘~’



Lógica Binária

- Tabelas de verdade

AND

X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

$$F = X \cdot Y$$

OR

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

$$F = X + Y$$

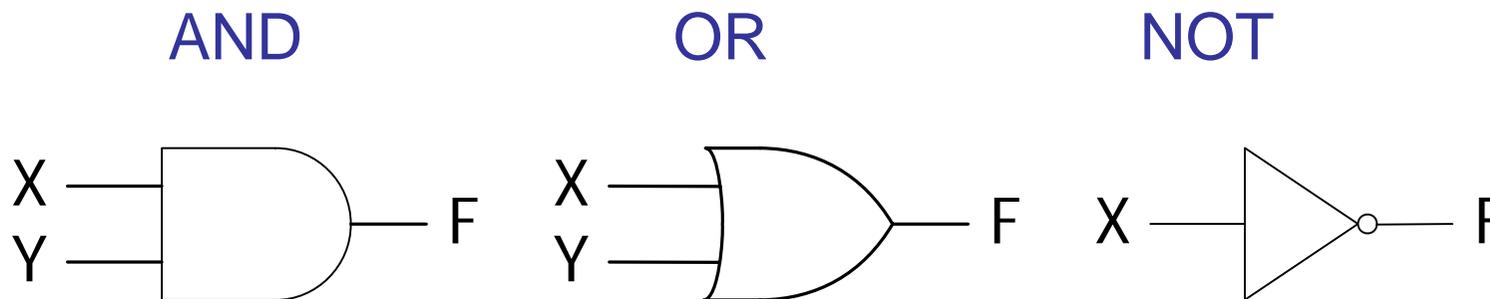
NOT

X	F
0	1
1	0

$$F = \bar{X}$$

Portas Lógicas

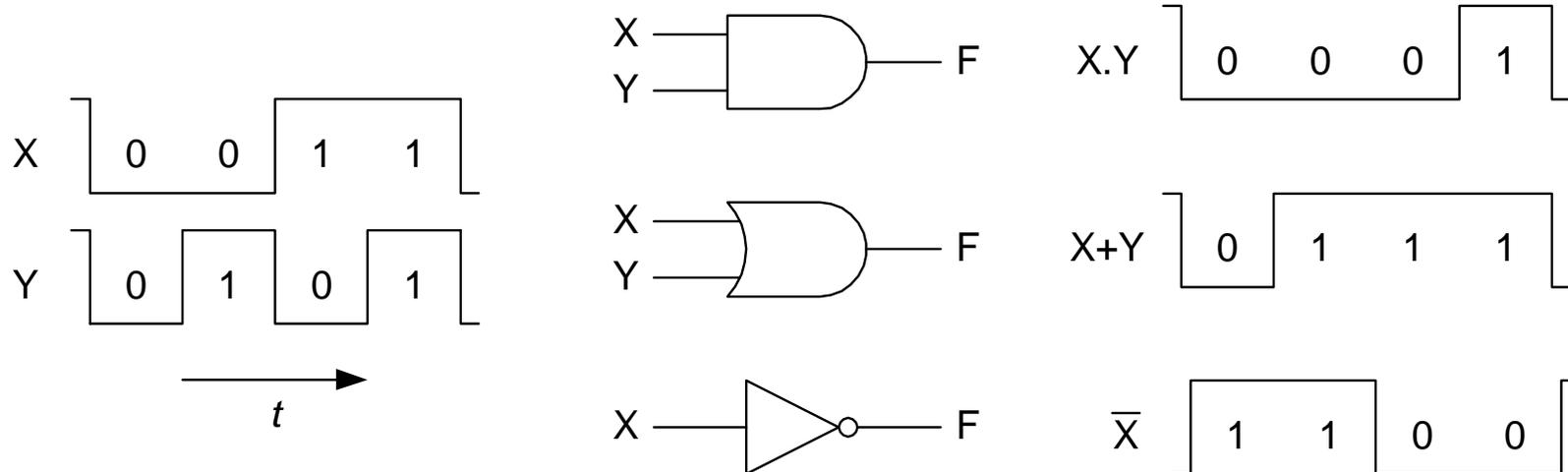
- Cada operação lógica tem um circuito eléctrico correspondente



- Estes componentes designam-se por **portas lógicas** (*logic gates*)

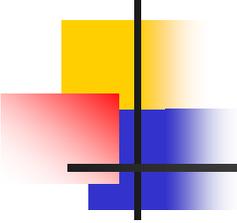
Portas Lógicas

■ Evolução temporal



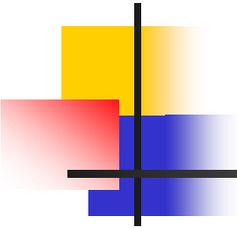
Na realidade existe um atraso temporal entre variações à entrada e consequente variação na saída

(veremos esse facto mais à frente)



Álgebra de *Boole*

- Representação de funções lógicas por equações
 - Exemplo: $F = \bar{X} + YZ$
- A partir da função lógica obtém-se:
 - **Tabela de verdade** – valores lógicos da função para todas as combinações de entradas possíveis
 - **Esquema do circuito** – com as portas lógicas e respectivas ligações



Álgebra de *Boole*

- Tabela de verdade

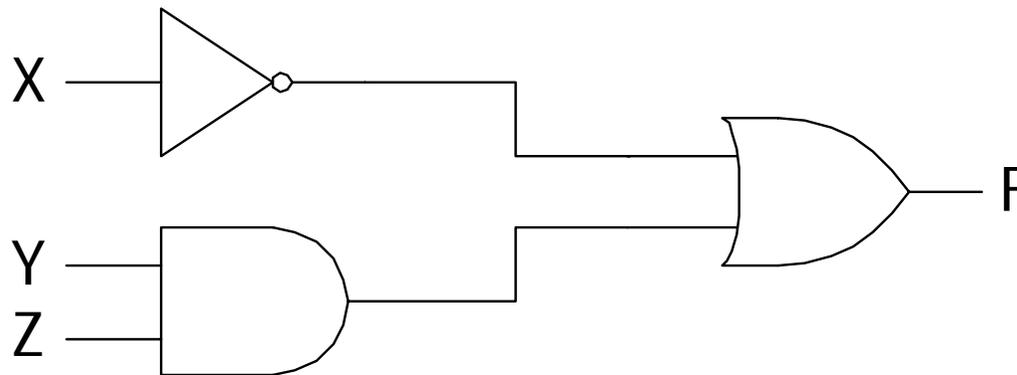
$$F = \bar{X} + YZ$$

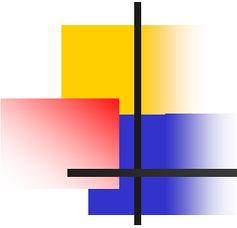
X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Álgebra de *Boole*

- Esquema do circuito

$$F = \bar{X} + YZ$$



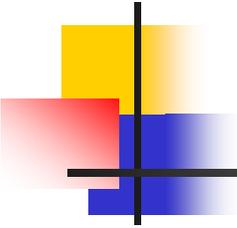


Álgebra de *Boole*

- Propriedades

- AND

- $X \cdot 1 = X$ (elemento neutro)
 - $X \cdot 0 = 0$ (elemento absorvente)
 - $X \cdot X = X$
 - $X \cdot \bar{X} = 0$
 - $X \cdot Y = Y \cdot X$ (comutativa)
 - $X(YZ) = (XY)Z$ (associativa)
 - $X + (YZ) = (X + Y) \cdot (X + Z)$ (distributiva)

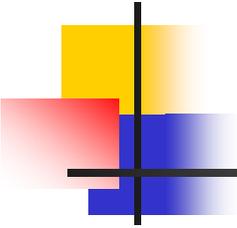


Álgebra de *Boole*

- Propriedades

- OR

- $X+0 = X$ (elemento neutro)
 - $X+1 = 1$ (elemento absorvente)
 - $X+X = X$
 - $X+\bar{X} = 1$
 - $X+Y = Y+X$ (comutativa)
 - $X+(Y+Z) = (X+Y)+Z$ (associativa)
 - $X.(Y+Z) = (XY) + (XZ)$ (distributiva)



Álgebra de *Boole*

- NOT

$$\overline{\overline{X}} = X$$

- Leis de DeMorgan

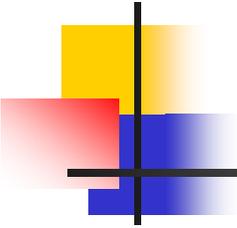
$$\overline{X+Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

- Teorema do consenso

$$XY + \overline{X}Z + YZ = XY + \overline{X}Z$$

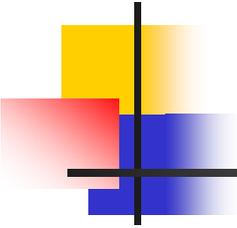
$$(X+Y)(\overline{X}+Z)(Y+Z) = (X+Y)(\overline{X}+Z)$$



Álgebra de *Boole*

- Aplicação das propriedades
 - Simplificação de equações
 - Exemplo

$$\begin{aligned}F &= XY\bar{Z} + XYZ + X = XY(\bar{Z} + Z) + X \\ &= XY \cdot 1 + X \\ &= XY + X \\ &= XY + X \cdot 1 \\ &= X(Y + 1) \\ &= X \cdot 1 \\ &= X\end{aligned}$$



Álgebra de *Boole*

- Outro exemplo
 - Simplificar a expressão

$$\begin{aligned}F &= X + \bar{X}Y + X(\bar{X} + \bar{Y}) = X + \bar{X}Y + X\bar{X} + X\bar{Y} \\ &= X + \bar{X}Y + 0 + X\bar{Y} \\ &= X + \bar{X}Y + X\bar{Y} \\ &= (X + \bar{X})(X + Y) + X\bar{Y} \\ &= 1 \cdot (X + Y) + X\bar{Y} \\ &= X + Y + X\bar{Y} \\ &= X(1 + \bar{Y}) + Y \\ &= X + Y\end{aligned}$$



Sistemas de Computação

Paulo Santos

Mapas de *Karnaugh*

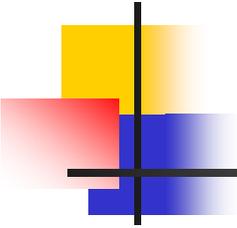


UNIÃO EUROPEIA

Fundo Social Europeu

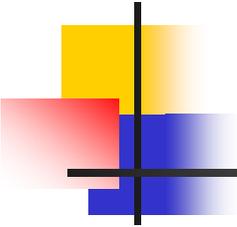
prime
Programa de Incentivos à
Modernização da Economia





Formas Normalizadas

- Cada função lógica pode ser escrita de diversas maneiras diferentes
- É habitual escrever a função utilizando uma **forma normalizada**
 - Simplificação mais fácil
 - Implementação mais eficiente do circuito
- Uma das formas normalizadas consiste em escrever a função como uma **soma de produtos**
 - é sempre possível, qualquer que seja a função lógica



Formas Normalizadas

- Exemplo:

- Considera a função $F(X,Y,Z)$ definida pela tabela de verdade

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

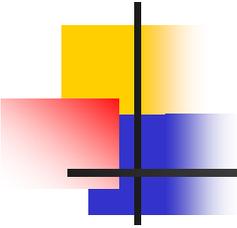
A partir da tabela de verdade pode-se escrever F como:

$$F = \bar{X}\bar{Y}\bar{Z} + \bar{X}YZ + XYZ$$

Simplificando:

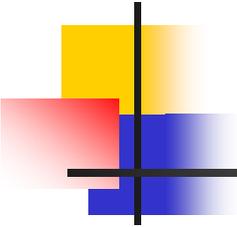
$$F = \bar{X}Y + YZ$$

Em ambos os casos F está escrita como uma **soma de produtos**



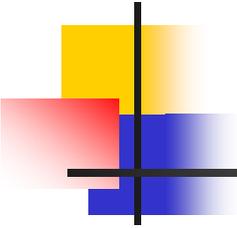
Formas Normalizadas

- **Termo mínimo**
 - produto lógico em que todas as variáveis da função aparecem exactamente uma vez (complementadas ou não)
- A cada **termo mínimo** corresponde uma dada combinação de entradas
 - por exemplo,
 $\bar{X}YZ$ corresponde a $X=0$; $Y=1$ e $Z=1$
- n variáveis $\Rightarrow 2^n$ **termos mínimos**



Termos Mínicos

X	Y	Z	Termo mínimo	Símbolo
0	0	0	$\bar{X} \bar{Y} \bar{Z}$	m_0
0	0	1	$\bar{X} \bar{Y} Z$	m_1
0	1	0	$\bar{X} Y \bar{Z}$	m_2
0	1	1	$\bar{X} Y Z$	m_3
1	0	0	$X \bar{Y} \bar{Z}$	m_4
1	0	1	$X \bar{Y} Z$	m_5
1	1	0	$X Y \bar{Z}$	m_6
1	1	1	$X Y Z$	m_7



Termos Mínimos

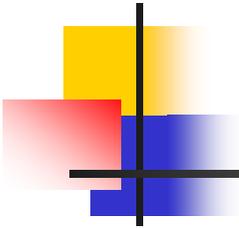
- Uma função pode ser definida como uma soma lógica de termos mínimos

- Exemplo

$$F(X, Y, Z) = \sum (m_2, m_3, m_4, m_7)$$

É o mesmo que

$$F = \overline{X}Y\overline{Z} + \overline{X}YZ + X\overline{Y}\overline{Z} + XYZ$$

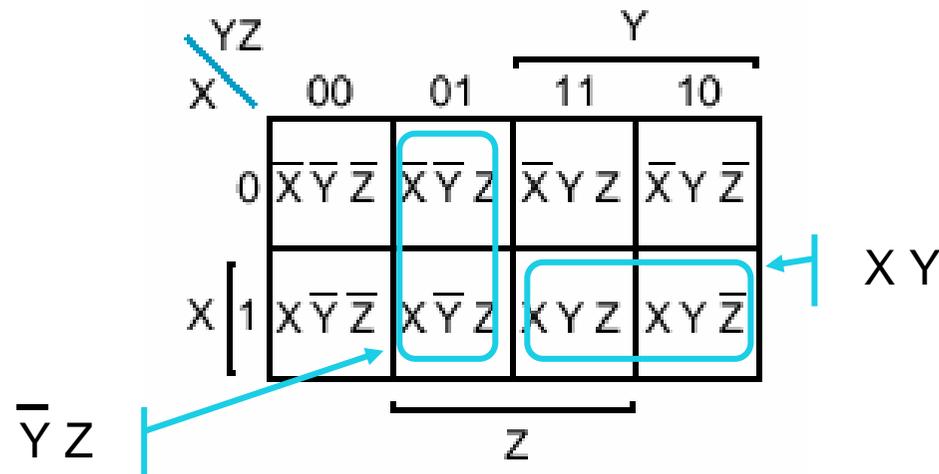


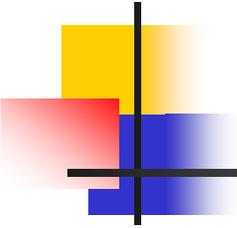
Mapas de *Karnaugh*

- Ferramenta útil para simplificação de funções e projecto de circuitos
- Geralmente utilizados quando se pretende obter uma função lógica a partir da tabela de verdade
- Representam-se no mapa os termos mínimos da função (com valor 1 ou 0)
- Obtém-se as funções lógicas através do agrupamento de termos mínimos com valor '1'

Mapas de *Karnaugh*

- Os termos mínimos são organizados de modo a que apenas mude um bit entre quadrados adjacentes (código binário reflectido)



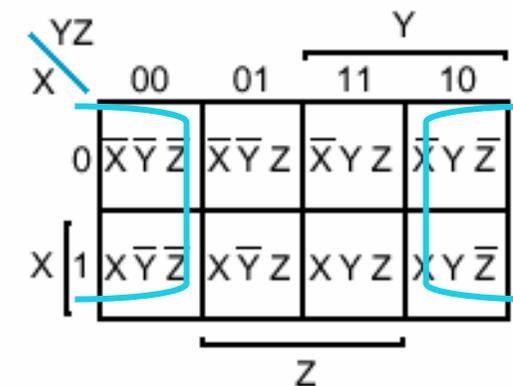
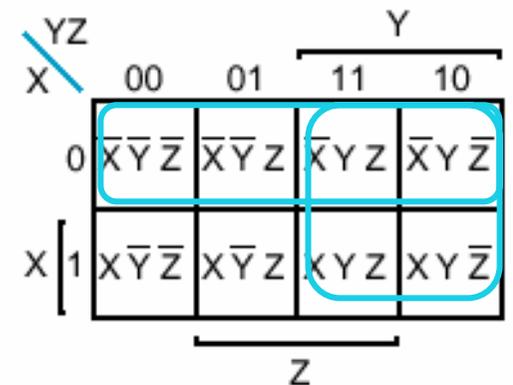
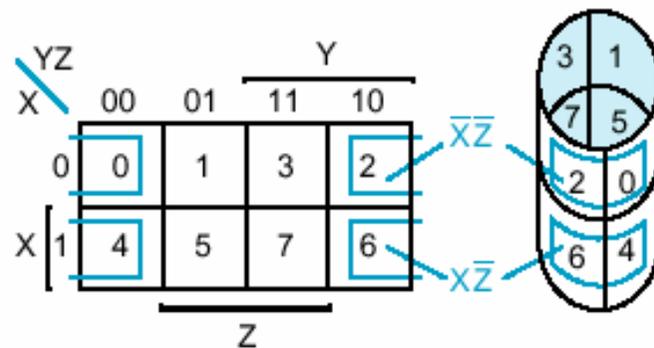


Mapas de *Karnaugh*

- Como agrupar os termos mínimos ?
 - As regras básicas são:
 - Só se podem agrupar termos mínimos adjacentes em que a função assume o valor lógico 1
 - Os agrupamentos são feitos segundo rectângulos ou quadrados sobre o mapa de *Karnaugh*
 - O número de termos mínimos dentro de cada agrupamento tem que ser uma potência de 2

Mapas de *Karnaugh*

- Exemplos de agrupamentos de termos mínimos

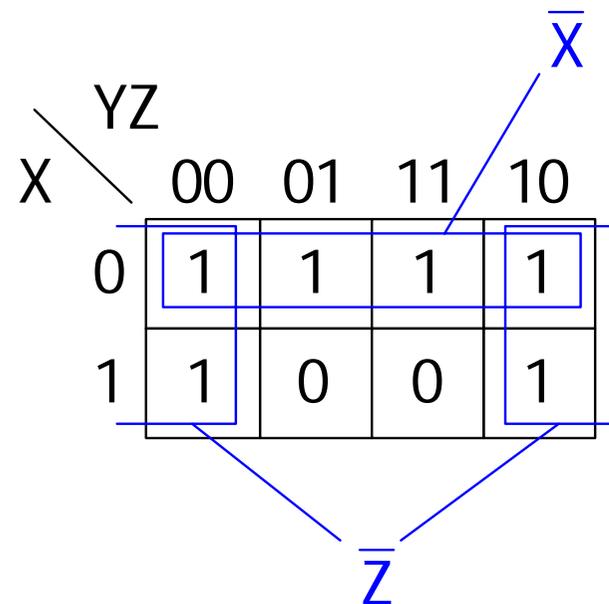


Mapas de *Karnaugh*

- Exemplo de simplificação de função

$$F = (X + Y) \cdot \bar{Z} + \bar{X}$$

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



A função simplifica-se para

$$F = \bar{X} + \bar{Z}$$

Mapas de *Karnaugh*

- Exemplo de obtenção dos termos produto

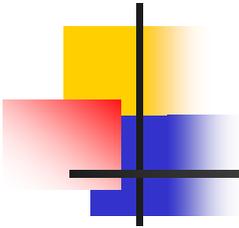
		ZW			
		00	01	11	10
XY	00	1	0	1	1
	01	0	0	1	1
	11	0	1	1	1
	10	1	0	1	1

Diagram illustrating the Karnaugh map for the function F . The map is a 4x4 grid with rows labeled XY (00, 01, 11, 10) and columns labeled ZW (00, 01, 11, 10). The values in the cells are: (00,00)=1, (00,11)=1, (00,10)=1, (01,11)=1, (01,10)=1, (11,01)=1, (11,11)=1, (11,10)=1, (10,00)=1, (10,11)=1, (10,10)=1.

Groupings are shown with colored boxes and lines:

- A red box highlights the cells (11,01) and (11,11), labeled XYW .
- A blue box highlights the cells (00,11), (01,11), (11,11), and (10,11), labeled Z .
- Green lines highlight the cells (00,00), (10,00), (00,10), and (10,10), labeled $\overline{Y}\overline{W}$.

$$F = XYW + \overline{Y}\overline{W} + Z$$



Mapas de *Karnaugh*

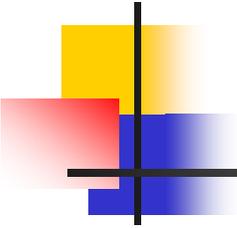
- Implicantes
 - Designa-se **implicante** de uma função a qualquer termo produto que assume o valor lógico 1 para todos os termos mínimos que o compõem
 - Se a simplificação de uma variável num implicante o transformar num termo produto não implicante, então esse implicante é um **implicante primo**

Mapas de *Karnaugh*

- Exemplos de implicantes e implicantes primos

X \ YZ	00	01	11	10
0	1	0	1	1
1	0	0	1	1

- Implicantes: $Y; \bar{X}\bar{Z}; \bar{X}Y; YZ; XY; \bar{X}\bar{Y}\bar{Z};$ etc.
- Implicantes primos: $Y; \bar{X}\bar{Z}$



Mapas de *Karnaugh*

- Implicantes primos
 - Se um termo mínimo estiver incluído apenas num implicante primo então esse implicante primo é **essencial**
 - Os implicantes primos a que a condição anterior não se aplica são implicantes primos **não-essenciais**

Mapas de *Karnaugh*

- Exemplos de implicantes primos essenciais e não essenciais

		ZW			
		00	01	11	10
XY	00	0	0	1	1
	01	0	0	1	1
	11	1	1	1	0
	10	0	0	0	0

— Implicantes primos essenciais

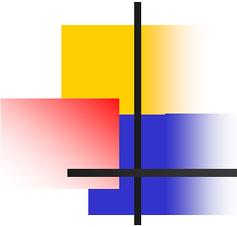
— Implicantes primos não-essenciais

Mapas de *Karnaugh*

- Quando uma determinada combinação de variáveis nunca ocorre, é possível definir a função como **não totalmente especificada**

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

- Os termos mínimos **não especificados** podem ser agrupados ao não, consoante dê mais jeito



Sumário (Aulas 4 e 5)

- Circuitos lógicos combinatórios
 - Definição de circuito combinatório
 - Procedimentos de projecto
 - Exemplos de circuitos combinatórios
 - Tecnologias de fabrico
 - Tempos de propagação



Sistemas de Computação

Paulo Santos

Circuitos Lógicos Combinatórios



UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

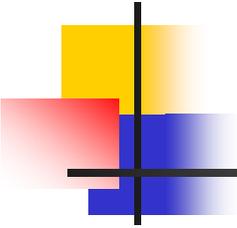
IQF
Instituto para a Qualidade
na Formação, I.P.

Circuitos Combinatórios

- Definição

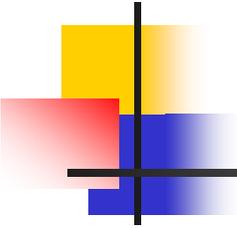
- Um circuito lógico diz-se **combinatório** se, em cada instante temporal, os valores das saídas dependem apenas dos valores das entradas





Procedimentos de Projecto

1. Determinar o n° de entradas, o n° de saídas e atribuir-lhes designações
2. Escrever a tabela de verdade que relaciona as entradas com as saídas
3. Obter funções simplificadas para cada saída
 - Mapas de *Karnaugh*, álgebra de *Boole*
4. Desenhar o esquema do circuito
5. Verificar o funcionamento do circuito



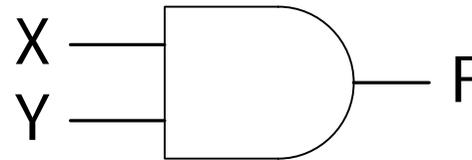
Ferramentas de Projecto

- Ferramentas de CAD
(*Computer Aided Design*)
 - Bibliotecas de funções e símbolos
 - Ferramentas de síntese
 - Tabelas de verdade
 - Linguagens de descrição de hardware (e.g. VHDL)
 - Editores esquemáticos
 - Simulação
 - Nível funcional
 - Nível lógico
 - Nível eléctrico

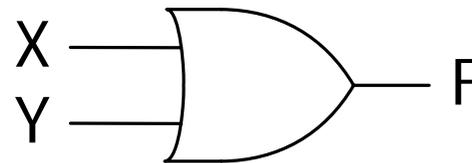
Portas Lógicas

- Portas lógicas estudadas até aqui...

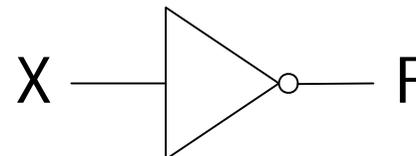
- AND



- OR



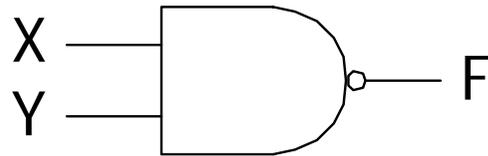
- NOT (*Inverter*)



Portas Lógicas

■ Outras portas lógicas

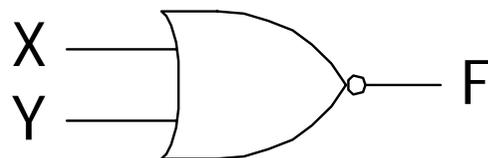
■ NAND



$$F = \overline{X \cdot Y}$$

X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

■ NOR

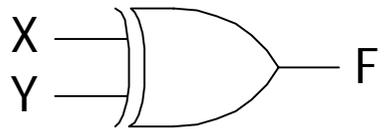


$$F = \overline{X + Y}$$

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

Portas Lógicas

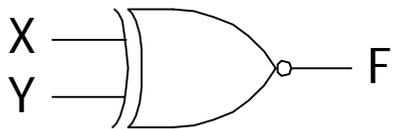
■ XOR (ou-exclusivo)



$$F = X \oplus Y = \bar{X} \cdot Y + X \cdot \bar{Y}$$

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

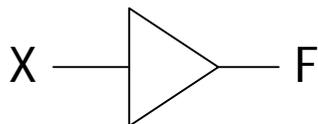
■ XNOR (equivalência)



$$F = \overline{X \oplus Y} = X \cdot Y + \bar{X} \cdot \bar{Y}$$

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

■ Buffer

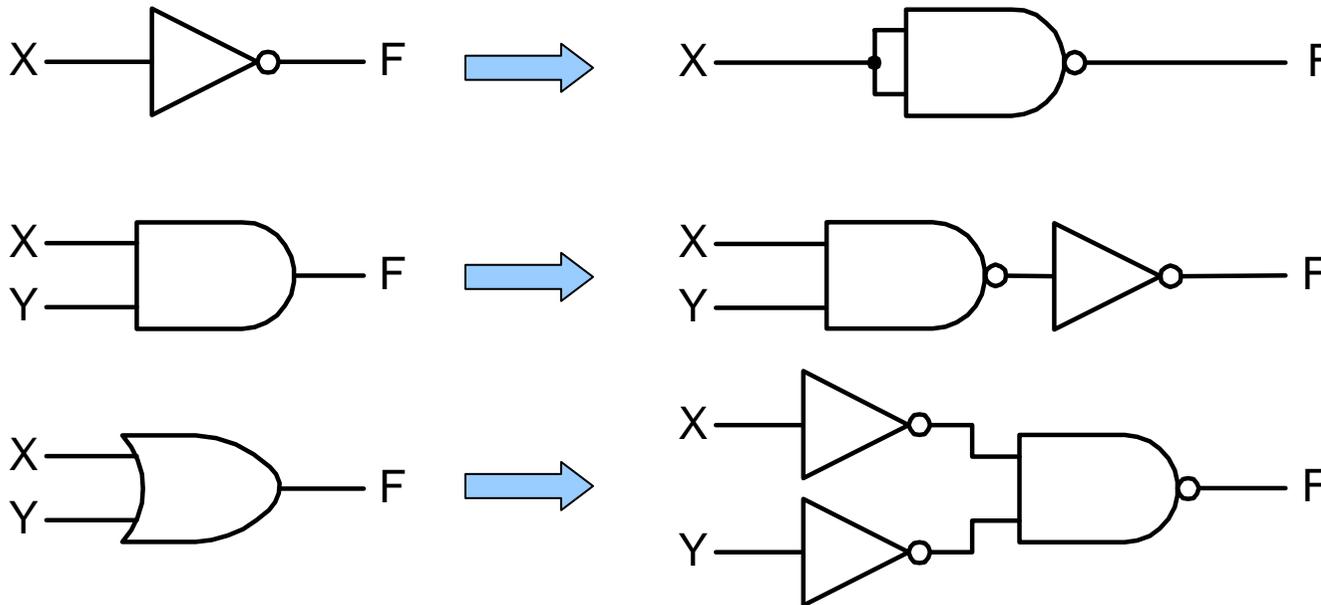


$$F = X$$

X	F
0	0
1	1

Portas Lógicas

- Qualquer circuito pode ser realizado utilizando apenas portas NAND ou apenas portas NOR

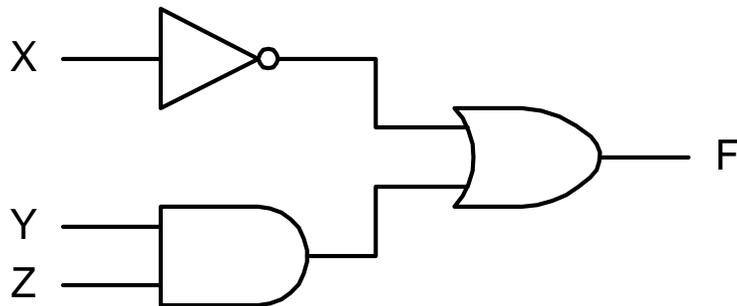


Portas Lógicas

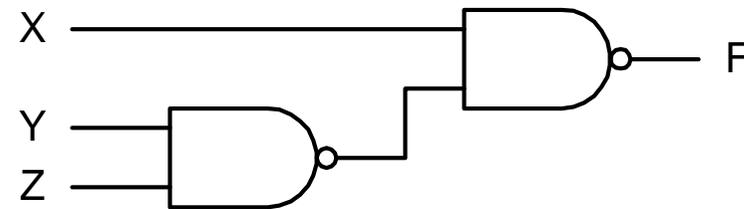
- Exemplo:

$$F = \bar{X} + YZ$$

Circuito directo

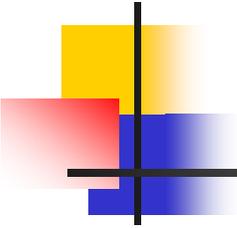


Circuito com NANDs



Algebricamente:

$$F = \bar{X} + Y.Z = \overline{\overline{\bar{X} + Y.Z}} = \overline{\overline{\bar{X}} \cdot \overline{Y.Z}} = \overline{X \cdot \overline{Y.Z}} = \overline{X \cdot \overline{Y.Z}}$$



Exemplos de Projecto

- Adicionador de 2 bits (*Half-adder*)

Pretende-se projectar um circuito que realize a adição de 2 bits. À saída deverão ser apresentados o resultado da adição e o bit de transporte.

Entradas:

X e Y – Bits a adicionar

Saídas:

S – Resultado da adição

C – Transporte (*Carry*)

Exemplos de Projecto

- Adicionador de 2 bits (*Half-adder*)

Tabela de Verdade

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Mapas de *Karnaugh*

		Função S		Função C	
		Y		Y	
X	0	0	1	0	0
	1	1	0	0	1

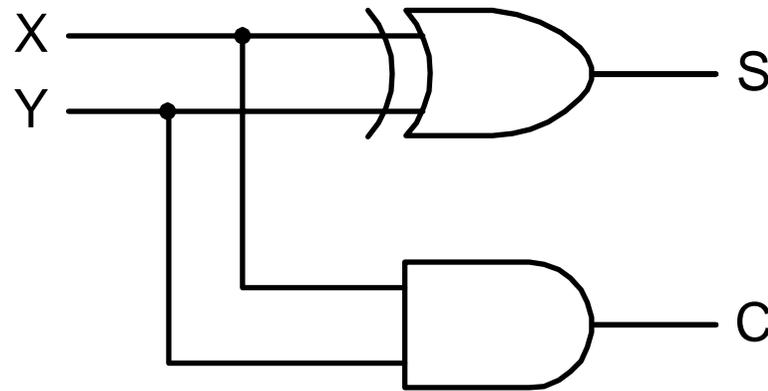
$$S = X\bar{Y} + \bar{X}Y \\ = X \oplus Y$$

$$C = XY$$

Exemplos de Projecto

- Adicionador de 2 bits (*Half-adder*)

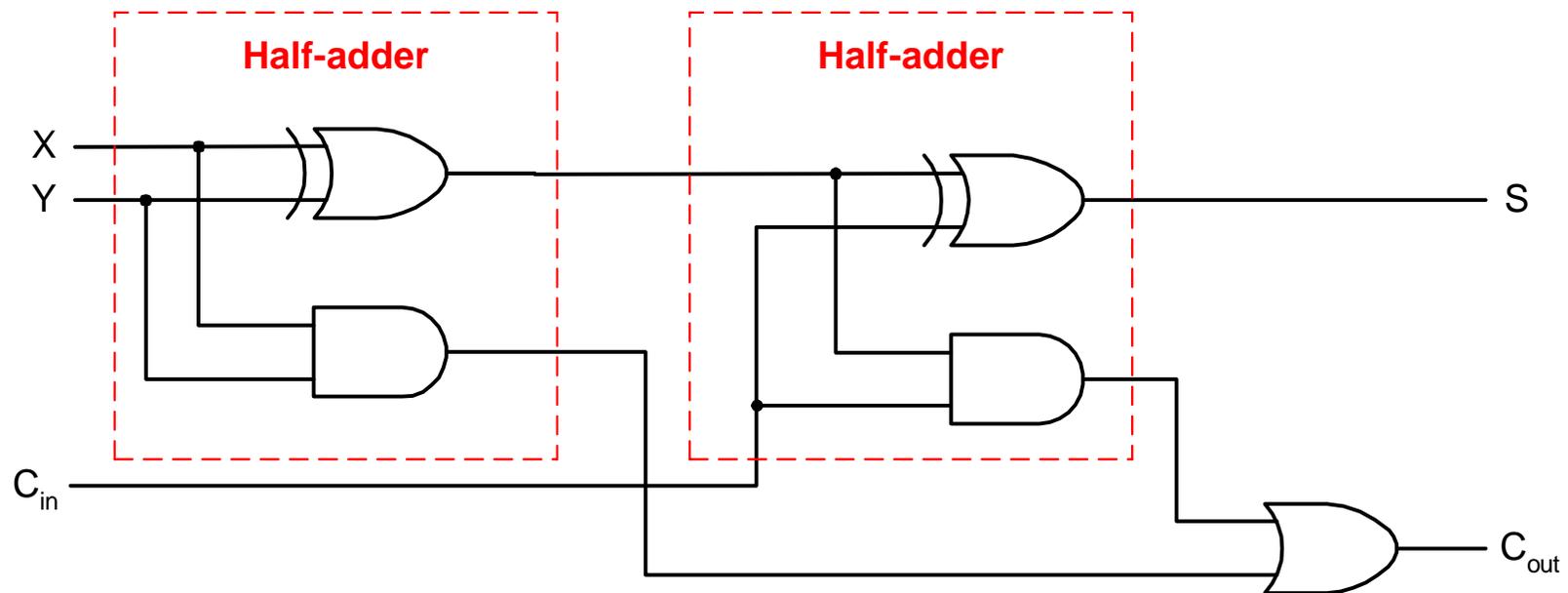
Circuito resultante



Exemplos de Projecto

- Adicionador de 2 bits completo (*Full Adder*)

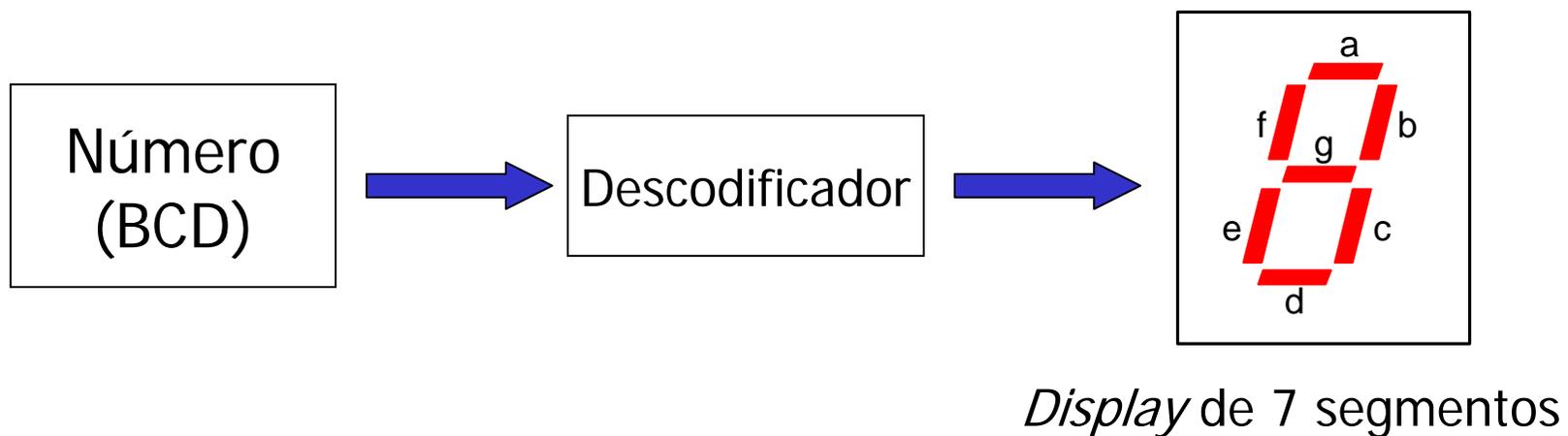
Com base no circuito anterior pode ser obtido um circuito, designado *Full Adder*, que adiciona dois bits (X e Y) ao bit de transporte de uma soma anterior (C_{in}).

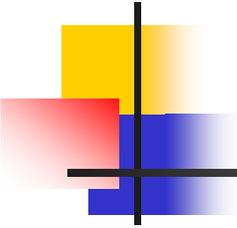


Exemplos de Projecto

- Descodificador BCD-Display 7 segmentos

Pretende-se projectar um circuito que descodifique um número representado em código BCD (números de 0 a 9, representados em binário), gerando à sua saída um conjunto de sinais que permitem apresentar o número num *display* de 7 segmentos.





Exemplos de Projecto

- Descodificador BCD-Display 7 segmentos

Tabela de verdade

B ₃	B ₂	B ₁	B 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
Restantes casos				0	0	0	0	0	0	0

Exemplos de Projecto

- Descodificador BCD-Display 7 segmentos
Mapas de *Karnaugh*

a

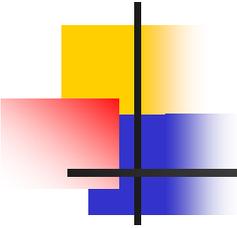
B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

$$a = \bar{B}_3 B_1 + \bar{B}_3 \bar{B}_2 \bar{B}_0 + \bar{B}_3 B_2 B_0 + B_3 \bar{B}_2 \bar{B}_1$$

b

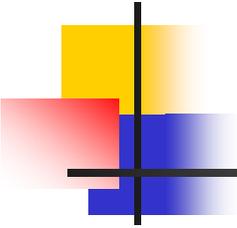
B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

$$b = \bar{B}_3 \bar{B}_2 + \bar{B}_3 \bar{B}_1 \bar{B}_0 + \bar{B}_3 B_1 B_0 + \bar{B}_2 \bar{B}_1$$



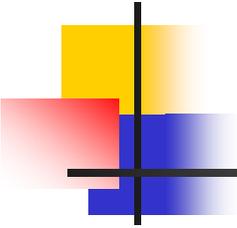
Tecnologias

- Densidade de integração
 - SSI (*Single Scale Integration*)
 - <10 portas lógicas por circuito integrado
 - MSI (*Medium Scale Integration*)
 - 10-100 portas / CI
 - Ex: Circuitos aritméticos, registos, contadores
 - LSI (*Large Scale Integration*)
 - 100-10000 portas / CI
 - Microprocessadores simples, memórias pequenas, controladores
 - VLSI (*Very Large Scale Integration*)
 - 10000 a mais de 100 Milhões de portas / CI
 - Processadores complexos (Pentium, Xeon, etc.)



Tecnologias

- Principais características
 - *Fan-in*
 - Número máximo de entradas que é possível implementar
 - *Fan-out*
 - Número máximo de portas que se podem ligar em série
 - Margem de ruído
 - Gamas de tensão correctamente interpretadas
 - Potência dissipada
 - Aquecimento dos circuitos
 - Tempos de propagação
 - t_{pHL} e t_{pLH}

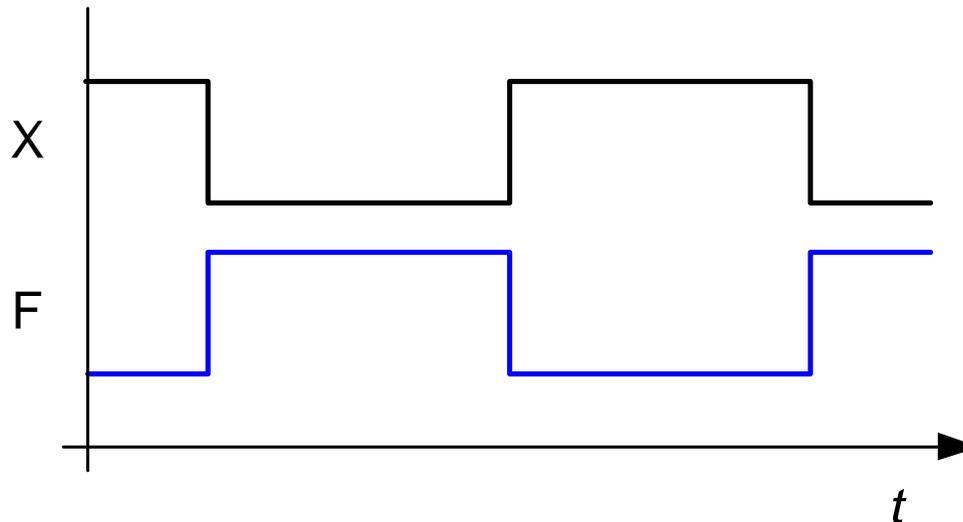
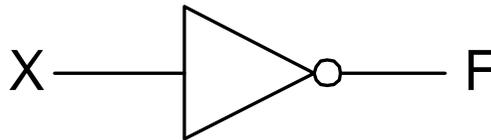


Tecnologias

- Principais famílias lógicas
 - TTL (*transistor-transistor logic*)
 - Rápidos; dissipam muita potência; actualmente em declínio
 - ECL (*emitter-coupled logic*)
 - Mais rápidos que TTL; dissipam muita potência
 - MOS (*metal-oxide semiconductor*)
 - Permitem uma grande densidade de integração
 - CMOS (*complementary metal-oxide semiconductor*)
 - Baixo consumo de energia; muito utilizados actualmente
 - BiCMOS (*bipolar complementary metal-oxide semiconductor*)
 - Combinação TTL e CMOS (mais rápido que apenas CMOS)
 - GaAs (*arseneto de gálio*)
 - A família mais rápida, mas também a mais cara

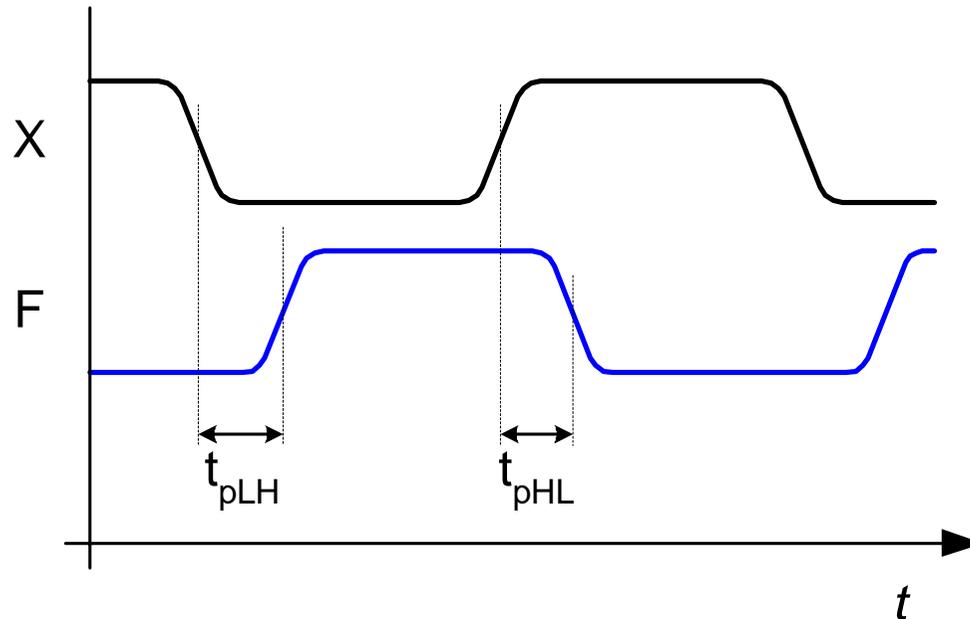
Tempos de Propagação

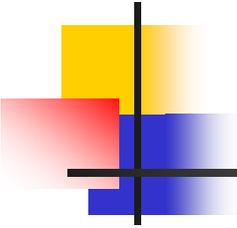
- Idealmente, as portas lógicas respondem instantaneamente a variações na entrada...



Tempos de Propagação

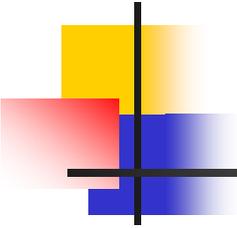
- ...mas, na realidade:
 - os sinais de entrada não são ondas perfeitas
 - as portas lógicas respondem com um atraso temporal





Tempos de Propagação

- t_{pLH} – Tempo de propagação LOW->HIGH
 - Tempo que a saída demora a subir após uma variação na entrada que cause a subida
- t_{pHL} – Tempo de propagação HIGH->LOW
 - Tempo que a saída demora a descer após uma variação na entrada que cause a descida
- t_{pd} – Tempo de propagação
 - $t_{pd} = \text{Max}(t_{pLH}, t_{pH})$



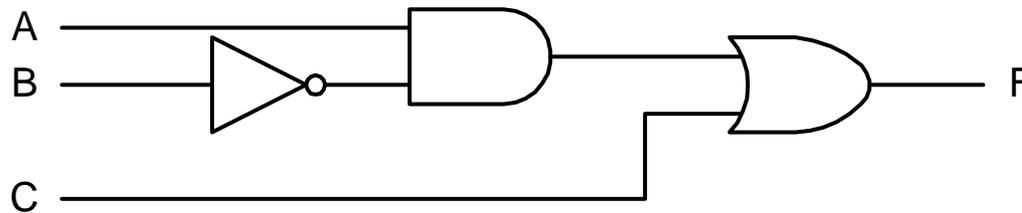
Tempos de Propagação

- Num dado circuito, muitas vezes interessa saber qual o pior caso de propagação
 - *Worst-case propagation delay* – impõe restrições temporais à velocidade com que operam os circuitos
- Ao pior caso de propagação, corresponde um dado caminho do circuito – *caminho crítico*.
 - Normalmente é o caminho que atravessa mais portas lógicas (isto se os atrasos das portas forem idênticos)

Tempos de Propagação

- Exemplo:

Qual será o pior caso de propagação no seguinte circuito ?
Indique uma situação que ilustre o pior caso.



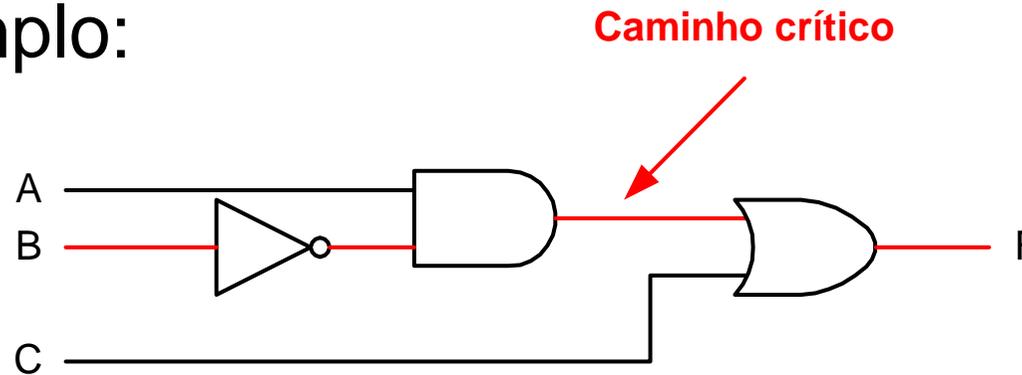
AND e OR: $t_{pd} = 10\text{ns}$

NOT: $t_{pd} = 7\text{ns}$

(Suponha $t_{pd} = t_{pLH} = t_{pHL}$ em todas as portas)

Tempos de Propagação

- Exemplo:



Exemplo de uma situação:

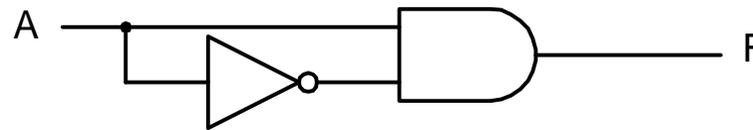
passagem de $A=1, B=1, C=0 \rightarrow F=0$

para $A=1, B=0, C=0 \rightarrow F=1$ (mudou-se o sinal de B)

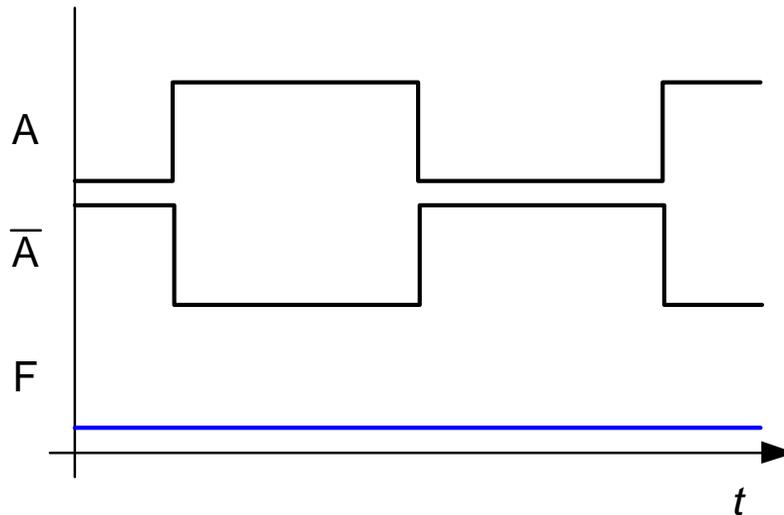
$$T_{pd(\text{Total})} = t_{pd(\text{NOT})} + t_{pd(\text{AND})} + t_{pd(\text{OR})} = 27\text{ns}$$

Tempos de Propagação

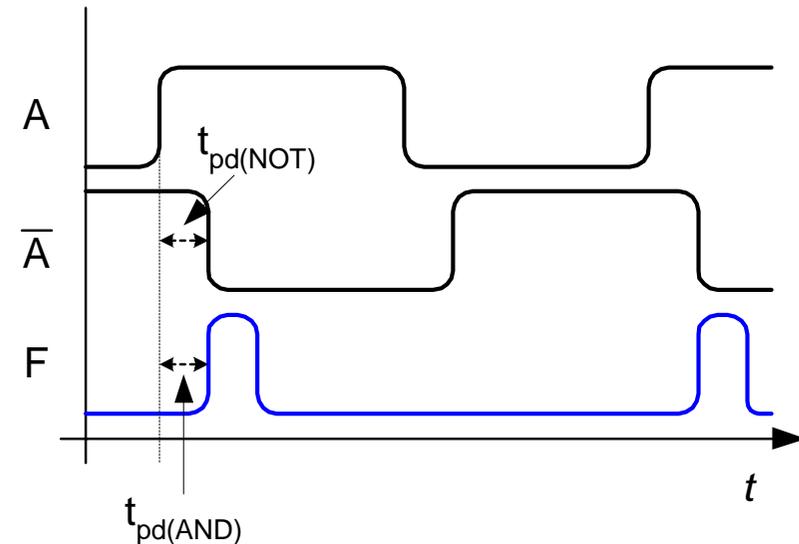
- Podem também ocorrer de picos na saída...
 - Exemplo:

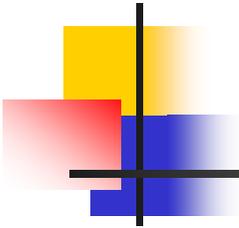


Caso ideal



Caso real





Sumário (Aulas 6 e 7)

- Circuitos combinatórios típicos
 - Descodificadores e codificadores
 - Multiplexers e desmultiplexers
 - Circuitos aritméticos
 - Notação em complemento para 2
 - Adicionador *ripple carry*
 - *Overflow*



Sistemas de Computação

Paulo Santos

Circuitos Combinatórios Típicos



UNIÃO EUROPEIA

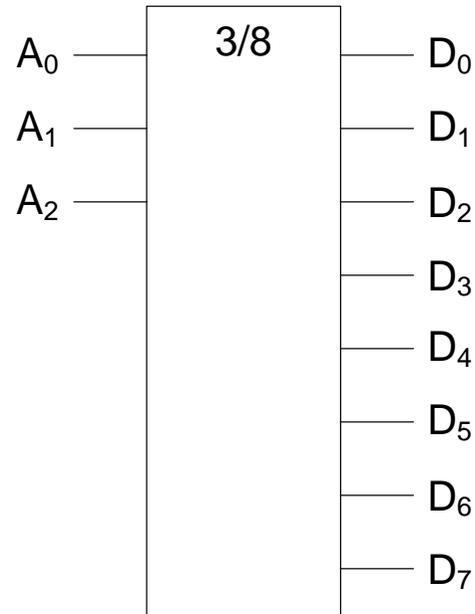
Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia



Descodificadores

- Circuito com **n entradas** e **2^n saídas**. Apenas uma saída virá com valor lógico diferente de todas as outras.
- Exemplo: descodificador 3-para-8 (ou 1 de 8)

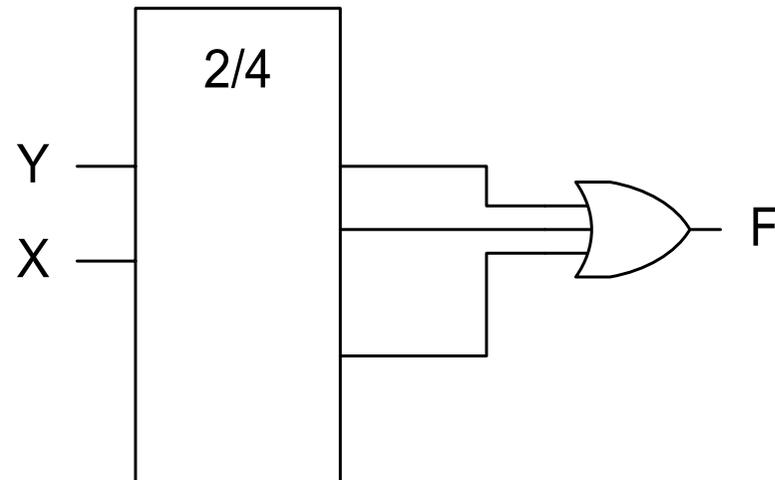
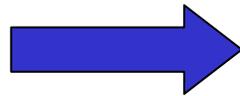


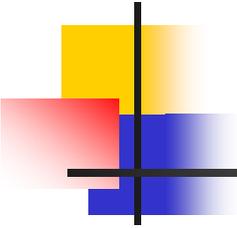
Entradas			Saídas							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Descodificadores

- Qualquer função lógica pode ser implementada utilizando um descodificador e portas OR
 - Tirando partido do facto de cada saída corresponder a um termo mínimo
- Exemplo

X	Y	F
0	0	1
0	1	1
1	0	0
1	1	1





Descodificadores

- Existem outros circuitos que se costumam designar por descodificador...
 - Descodificador BCD → 7Segmentos
 - Descodificador BCD → Excesso-3
 - Etc.

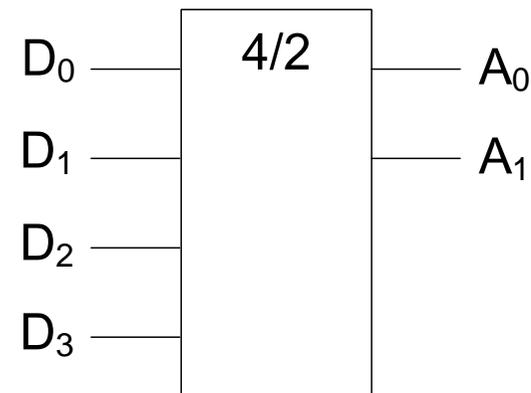
- Uma designação mais correcta para este tipo de circuitos seria *conversores de código*

Codificadores

- Circuito com **2ⁿ entradas** e **n saídas** que realiza a operação inversa de um decodificador.
 - A saída será o número da entrada que está activa
- Exemplo: Codificador 4-para-2

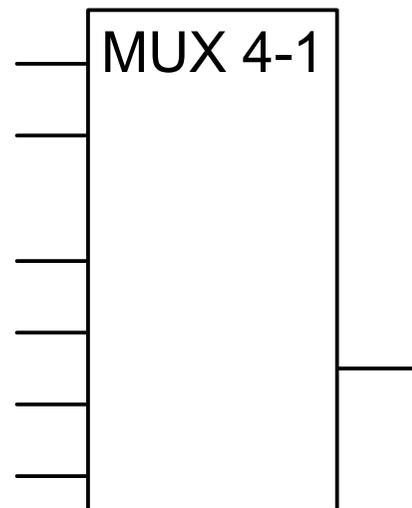
D ₃	D ₂	D ₁	D ₀	A ₁	A ₀
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$A_0 = D_1 + D_3 \quad A_1 = D_2 + D_3$$



Multiplexers

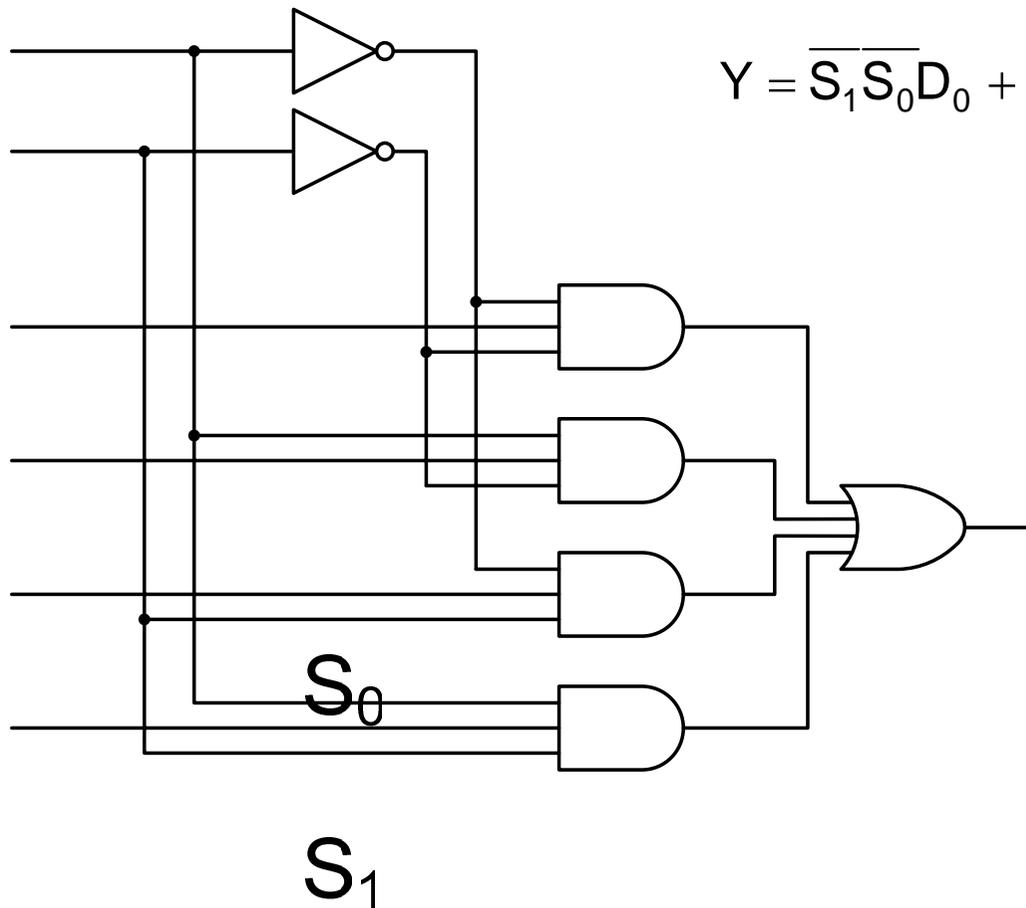
- Circuito combinatório cuja finalidade é seleccionar uma de $m=2^n$ **entradas** possíveis
- A selecção é feita através de n **linhas de controlo**.
- Exemplo: Multiplexer 4-para-1



S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Multiplexers

- Esquema interno e função lógica



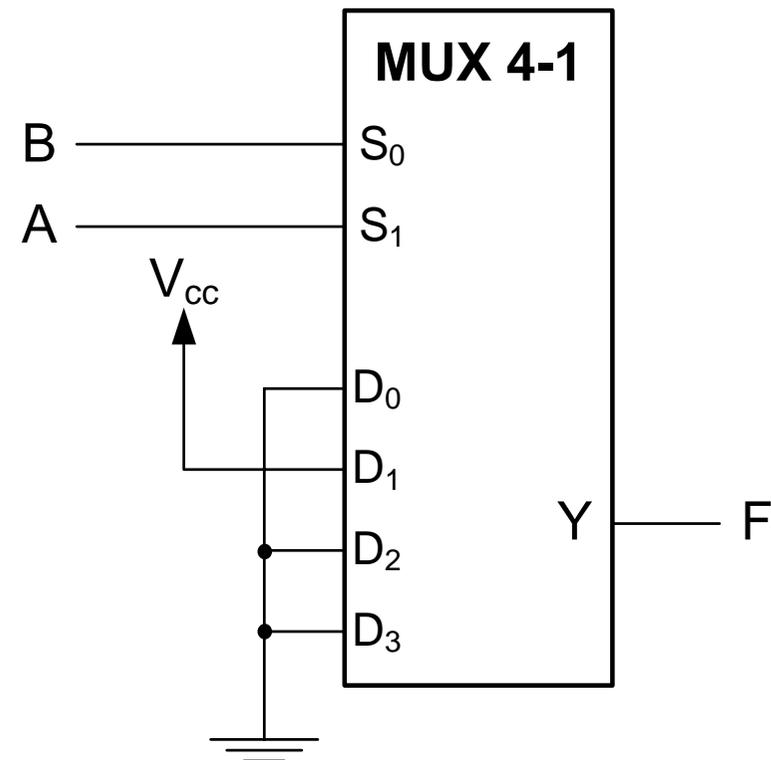
$$Y = \overline{S_1}\overline{S_0}D_0 + \overline{S_1}S_0D_1 + S_1\overline{S_0}D_2 + S_1S_0D_3$$

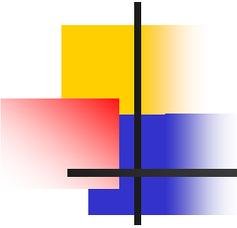
Multiplexers

- Função lógica de n variáveis
 - facilmente implementada com um multiplexer com n selectores

- Exemplo:

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



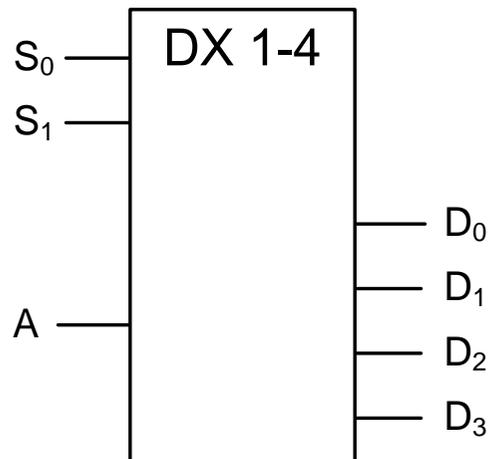


Multiplexers

- 2 temporizações com interesse:
 - T_{PD} entradas de dados \Rightarrow saída
 - T_{PD} selectores \Rightarrow saída
- Também poderão existir sinais de *output enable*
- Um circuito integrado pode conter vários multiplexers
 - Muitas vezes com linhas de selecção comuns

Desmultiplexers

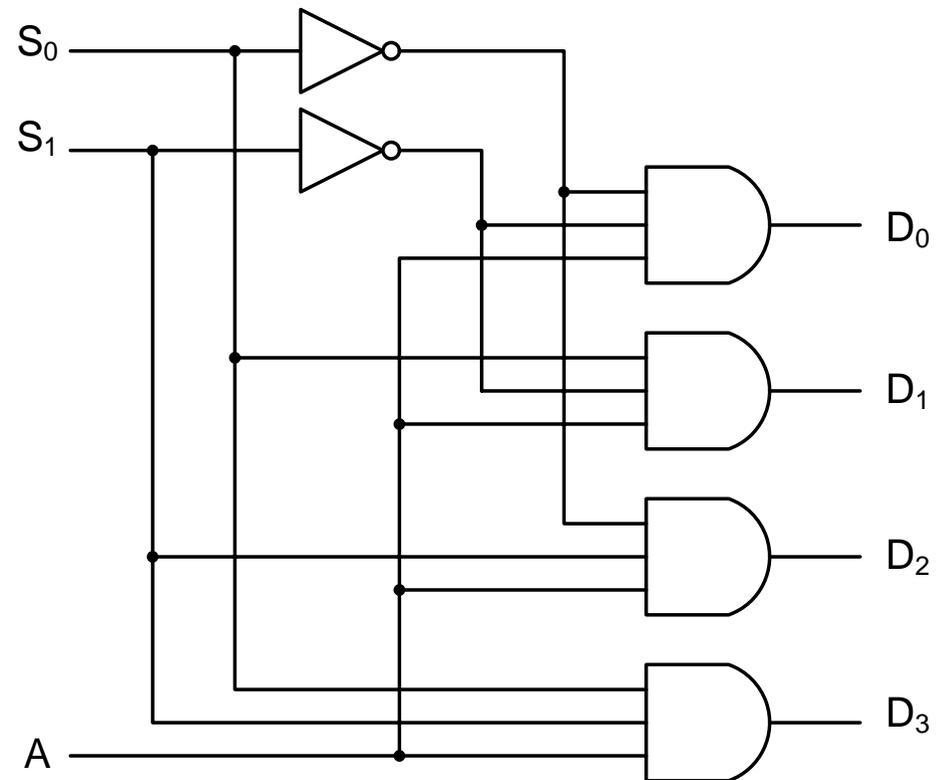
- Circuito que realiza a operação inversa de um multiplexer.
 - Direcçiona a entrada para uma de **2ⁿ linhas de saída**, que é seleccionada através de **n linhas de controlo**
- Exemplo: Desmultiplexer 1-para-4 linhas



S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

Desmultiplexers

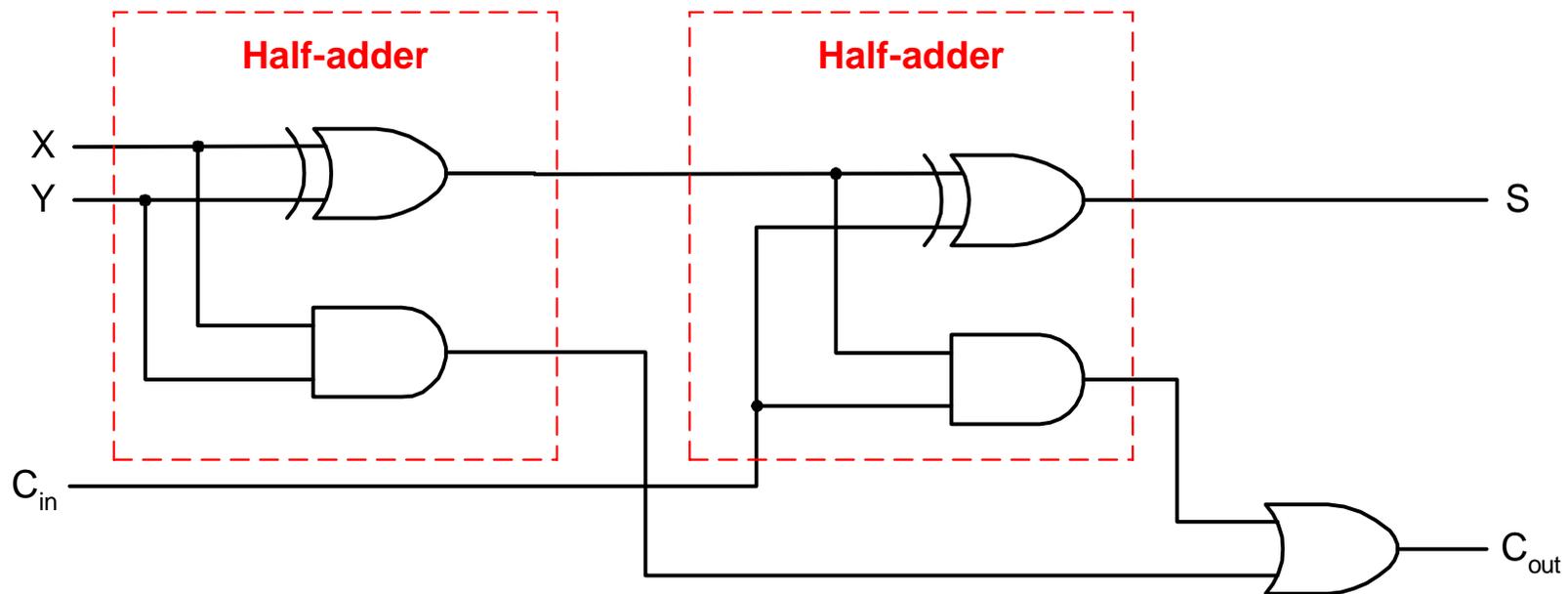
- Esquema interno



Adição Binária

■ Adicionador

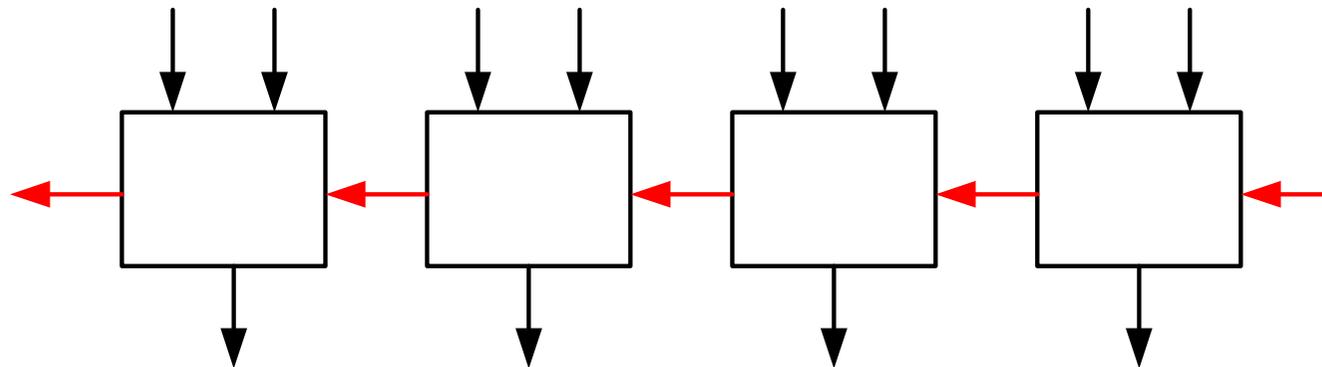
Relembrando uma das aulas anteriores...



Adição Binária

- Adicionador de 4 bits

... é possível obter um circuito que adiciona 4 bits, utilizando *Full Adders* e propagando o bit de transporte



Este circuito designa-se por *Ripple Carry Adder*

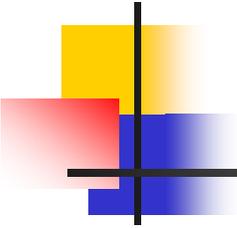
A₃

B₃

A₂

121

B₂



Subtracção Binária

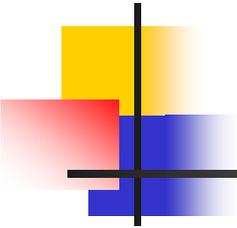
- Complemento para 2

- Permite uma representação coerente de números negativos em base 2
- Bit de maior peso \Rightarrow Bit de sinal
- O complemento para 2 obtém-se **complementando o número e adicionando 1**

- Exemplos

- $5 = 0101 \Rightarrow -5 = 1010 + 1 = 1011$

- $3 = 0011 \Rightarrow -3 = 1100 + 1 = 1101$



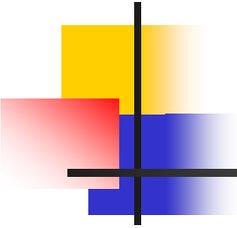
Subtracção Binária

- Complemento para 2
 - Números inteiros no intervalo [-8;7]

7	0111	-1	1111
6	0110	-2	1110
5	0101	-3	1101
4	0100	-4	1100
3	0011	-5	1011
2	0010	-6	1010
1	0001	-7	1001
0	0000	-8	1000

Decimal

**Complemento
para 2**



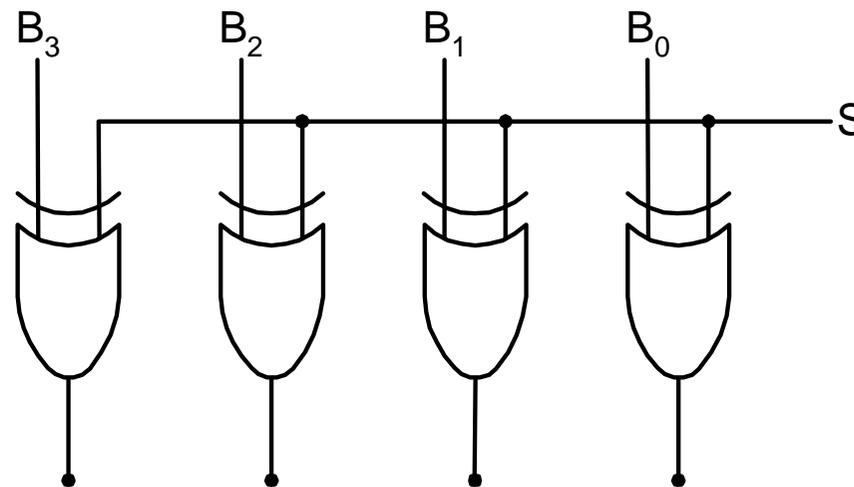
Subtracção Binária

- Complemento para 2
 - Subtrair um número inteiro é o mesmo que somar o seu complemento para 2
 - Exemplos

11000		11000	
0100	4	1101	-3
+ 1110	-2	+ 1100	-4
<hr/>		<hr/>	
10010	2	11001	-7

Subtraç o Bin ria

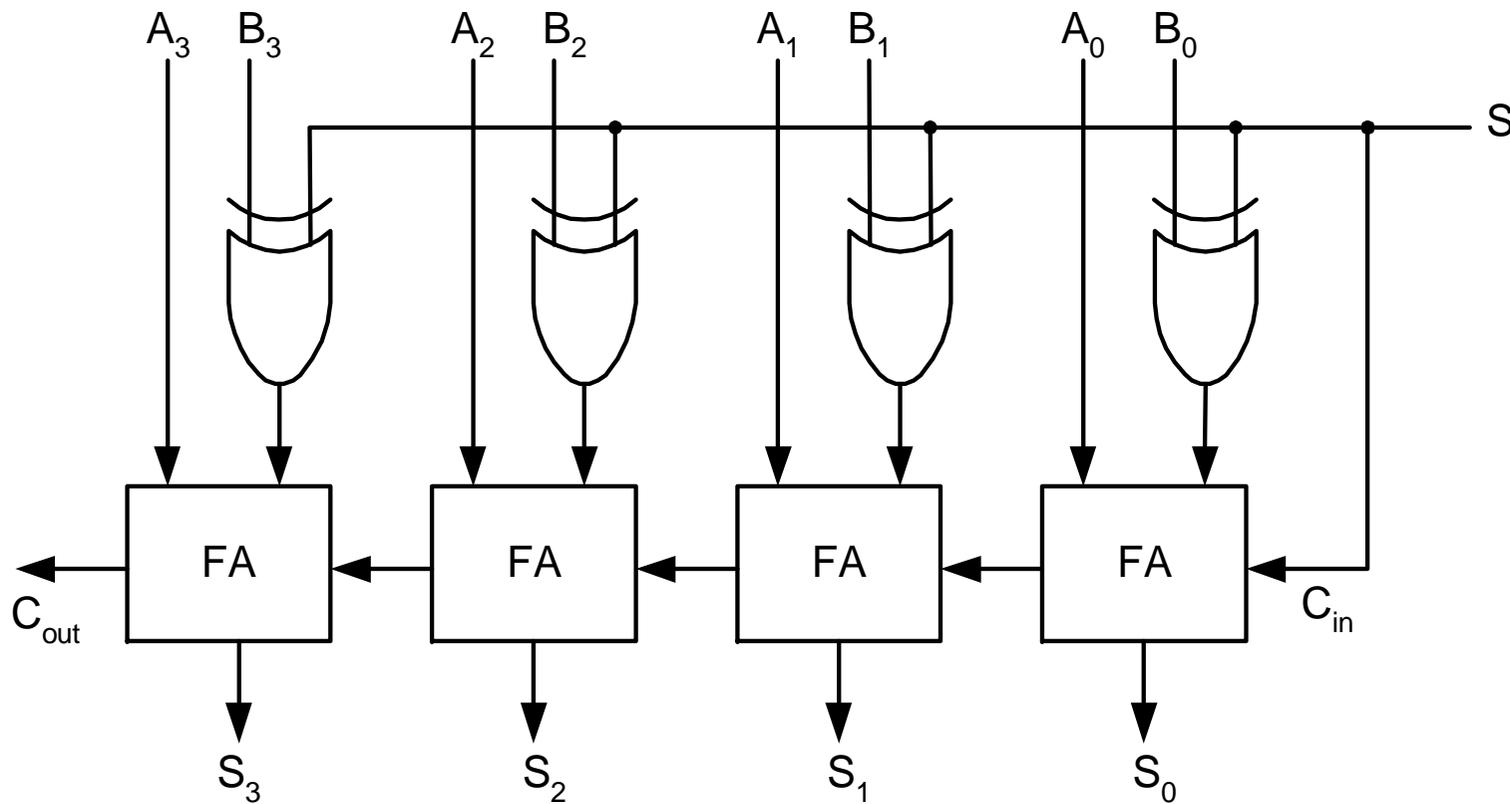
- Circuito que complementa (ou n o) um n mero de 4 bits, em funç o de S (selec o de operaç o)

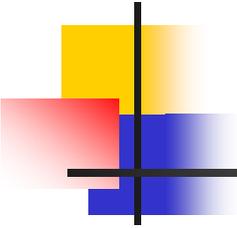


- S=0 – n o complementa B (para somar)
- S=1 – complementa B (para subtrair)

Subtraç o Bin ria

- Adicionador / subtrator de 4 bits



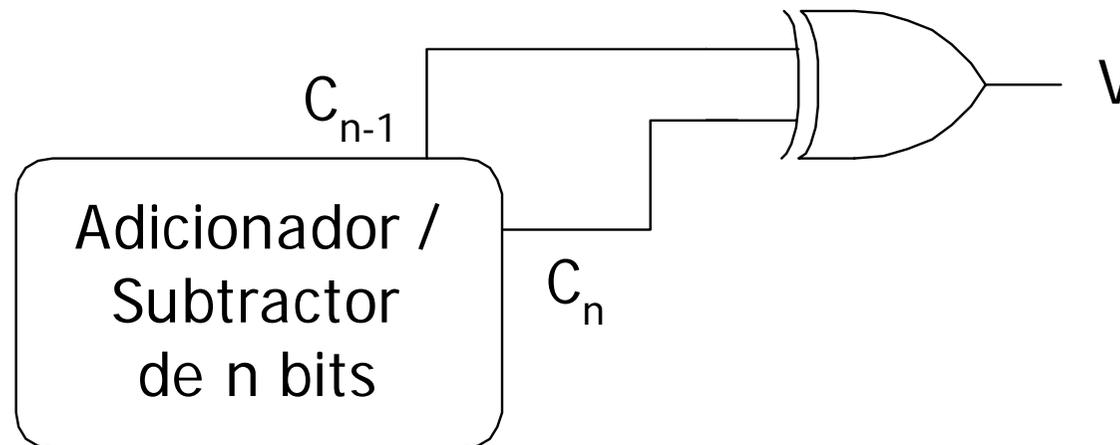


Overflow

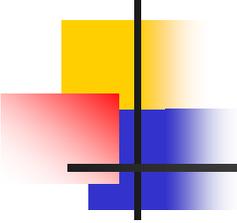
- Resultado de uma adição/subtração fora do intervalo de operação \Rightarrow *overflow*
- Exemplo (4 bits):
 - Intervalo de operação: [-8;7]
 - Operações em que ocorre *overflow*
 - 6+3
 - -3-7, etc.

Overflow

- Detecção de *overflow*
 - A ocorrência de *overflow* pode ser detectada analisando os 2 últimos *carrys*



- A saída V indica se há ou não *overflow*



Sumário (Aulas 8 e 9)

- Circuitos lógicos sequenciais
 - Definição de circuito sequencial
 - *Latches*
 - *Latch SR*
 - *Latch D*
 - *Flip-flops*
 - *Flip-flops JK e SR Master-Slave*
 - *Flip-Flop JK e D Edge-Triggered*



Sistemas de Computação

Paulo Santos

Circuitos Lógicos Sequenciais

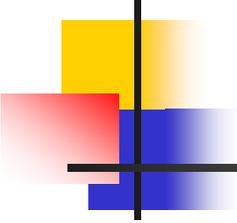


UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

IQF
Instituto para a Qualidade
na Formação, I.P.

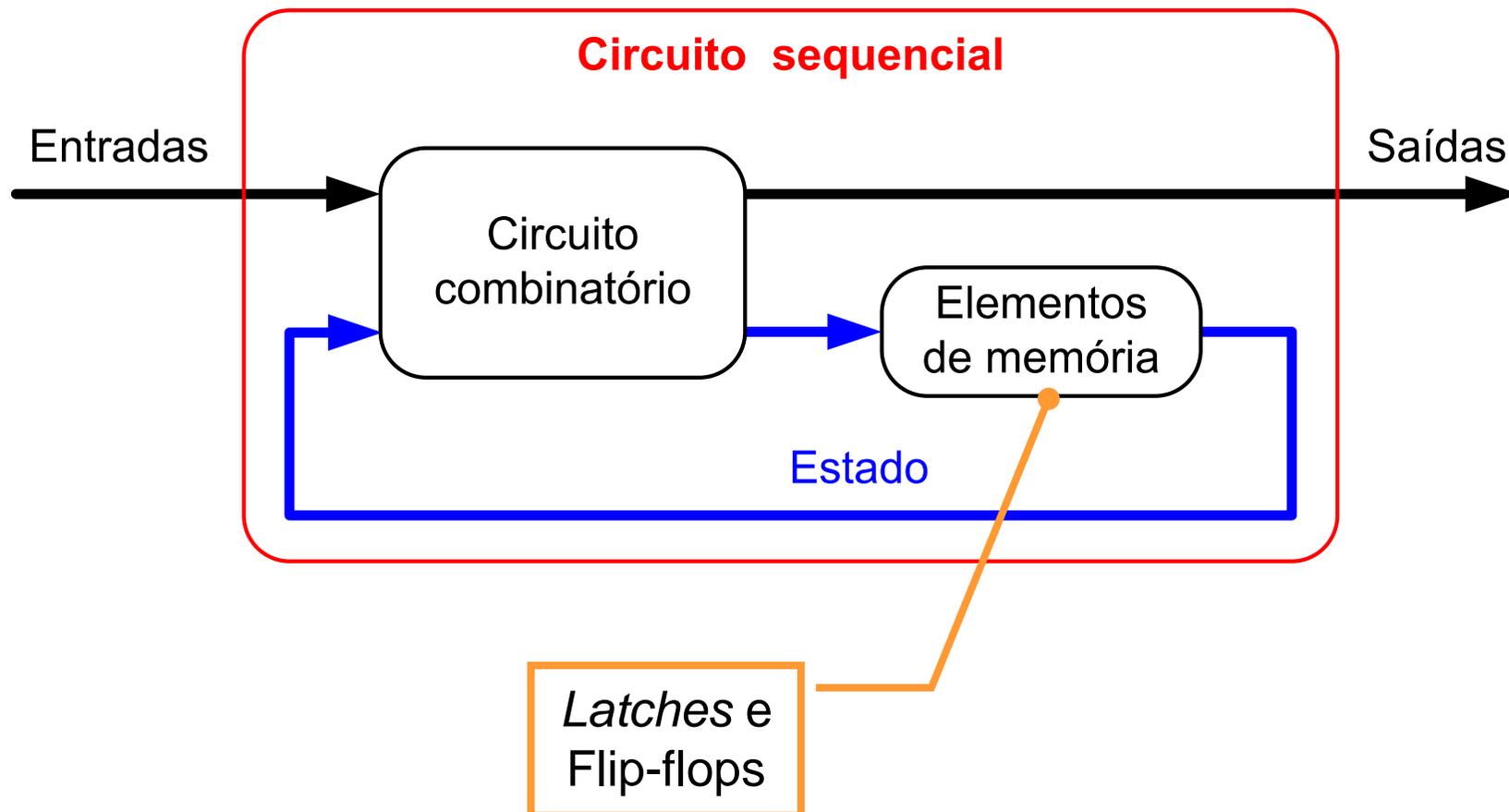


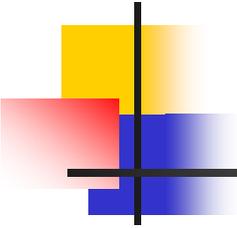
Circuitos Sequenciais

- **Definição** – um circuito diz-se **sequencial** quando as suas saídas dependem não só das entradas, mas também do **estado** do circuito.
 - Um circuito sequencial possui elementos de memória
 - O estado do circuito corresponde aos bits armazenados nos elementos de memória
 - A mesma combinação de entradas pode originar valores diferentes na(s) saída(s)

Circuitos Sequenciais

- Modelo





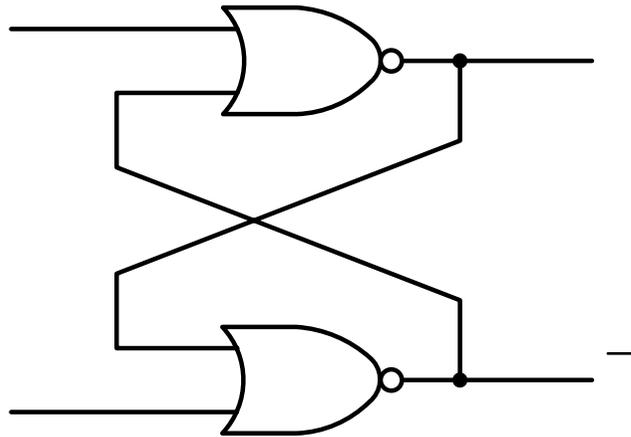
Latches e Flip-Flops

- Elementos de memória
 - *Latch* – elemento básico que permite armazenar um bit de informação
 - *Flip-Flop* – elemento construído a partir de *latches*, que permite um maior controlo no armazenamento da informação

Latches

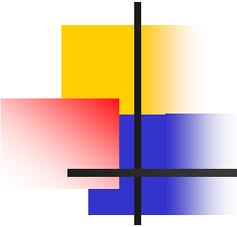
■ Latch SR

Esquema



Funcionamento

R	S	Q	\bar{Q}	
1	0	0	1	Reset
0	0	0	1	
0	1	1	0	Set
0	0	1	0	
1	1	0	0	
0	0	?	?	Indefinido



Latches

- *Latch* SR – Resumo das operações

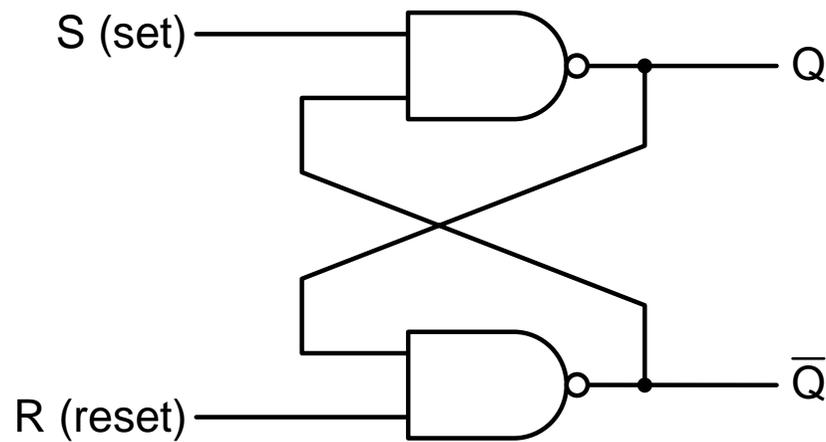
S	R	Q_{t+1}	Obs.
0	0	Q_t	Manter estado anterior
0	1	0	<i>Reset</i> (guardar '0')
1	0	1	<i>Set</i> (guardar '1')
1	1	0	Não utilizado *

* – Pode conduzir a um estado indefinido

Latches

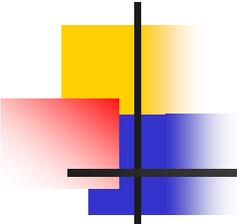
■ Latch $\overline{S}\overline{R}$

Esquema



Funcionamento

	—	
		Reset
		Set
		Indefinido



Latches

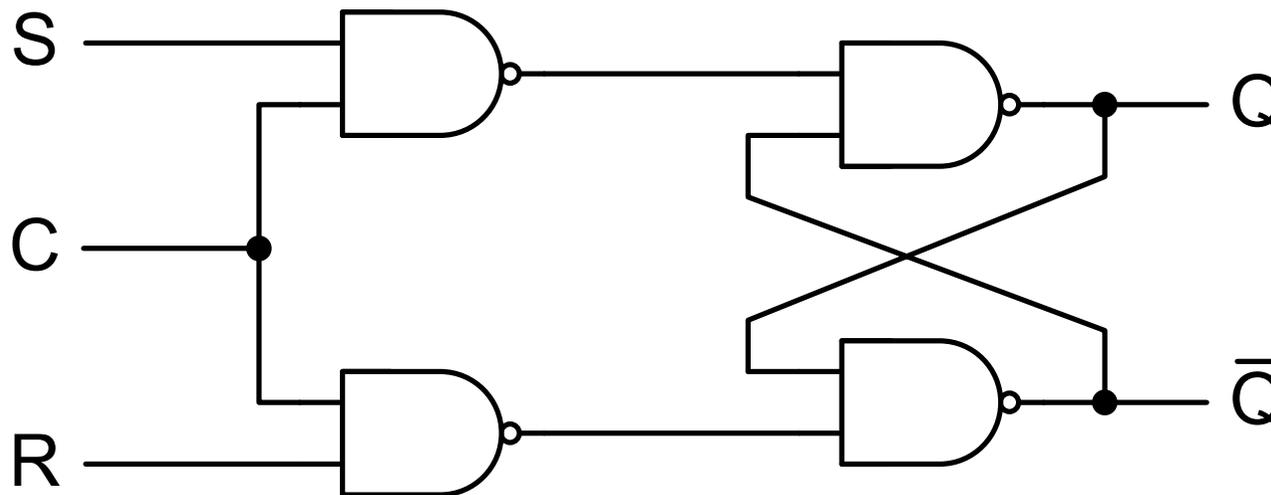
- *Latch* $\overline{S}\overline{R}$ – Resumo das operações

S	R	Q_{t+1}	Obs.
0	0	1	Não utilizado
0	1	1	<i>Set</i>
1	0	0	<i>Reset</i>
1	1	Q_t	Manter estado anterior

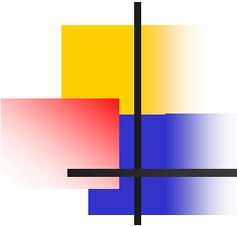
O funcionamento é semelhante ao do *latch* SR, mas com os níveis lógicos complementados.

Latches

- Introduzindo uma **variável de controlo** C melhora-se o controlo sobre o armazenamento da informação



Este *latch* SR só permite as operações *Set* e *Reset* quando a variável C está no nível lógico '1'.



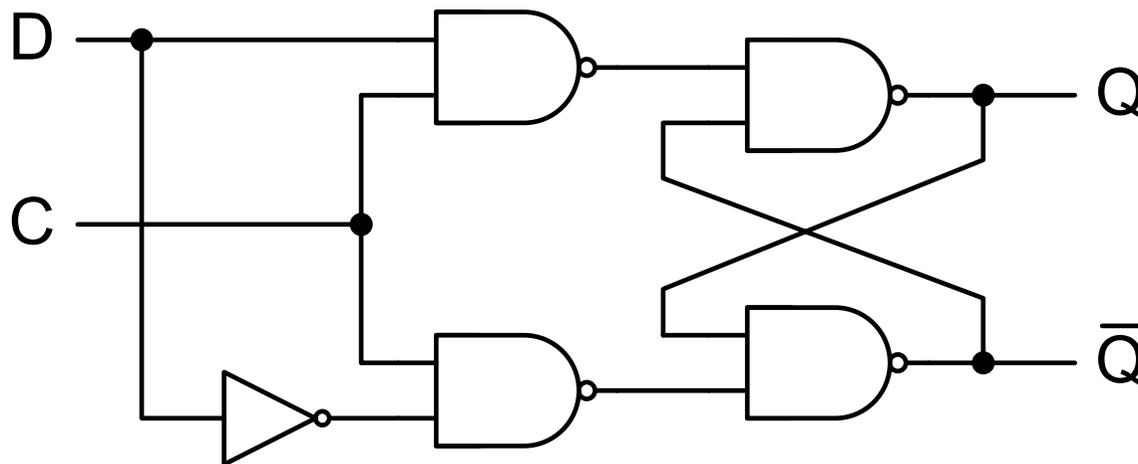
Latches

- *Latch* SR com controlo
Resumo das operações

C	S	R	Q_{t+1}	Obs.
0	x	x	Q_t	Manter estado anterior
1	0	0	Q_t	Manter estado anterior
1	0	1	0	<i>Reset</i>
1	1	0	1	<i>Set</i>
1	1	1	1	Não utilizado

Latches

■ Latch D

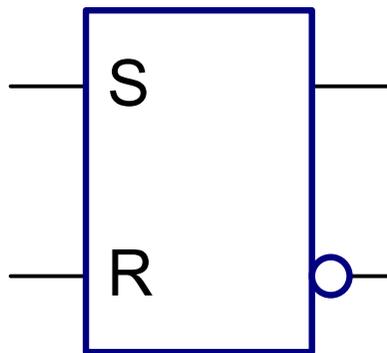


C	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

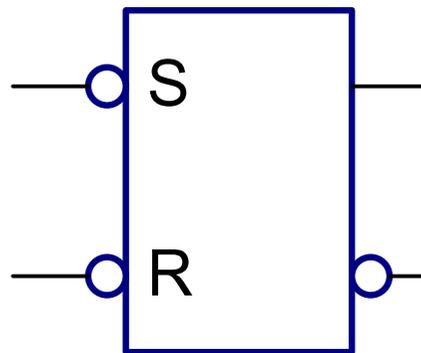
O *latch* D não tem o problema do estado indefinido.

Latches

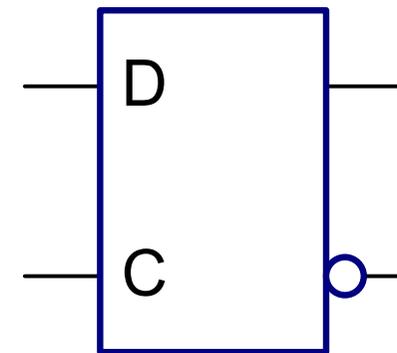
- Alguns símbolos...



SR



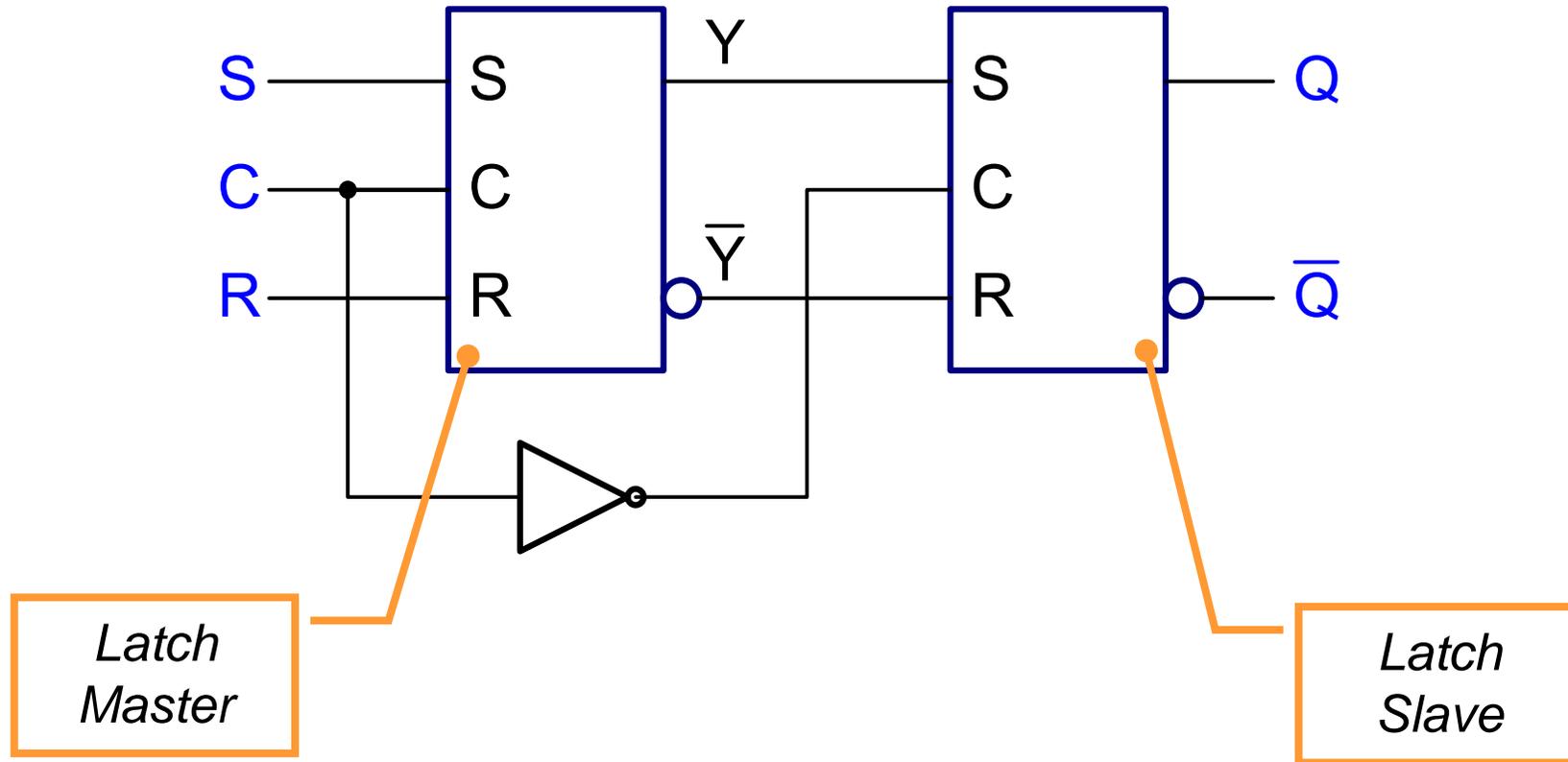
$\overline{S}\overline{R}$



D
(com controlo)

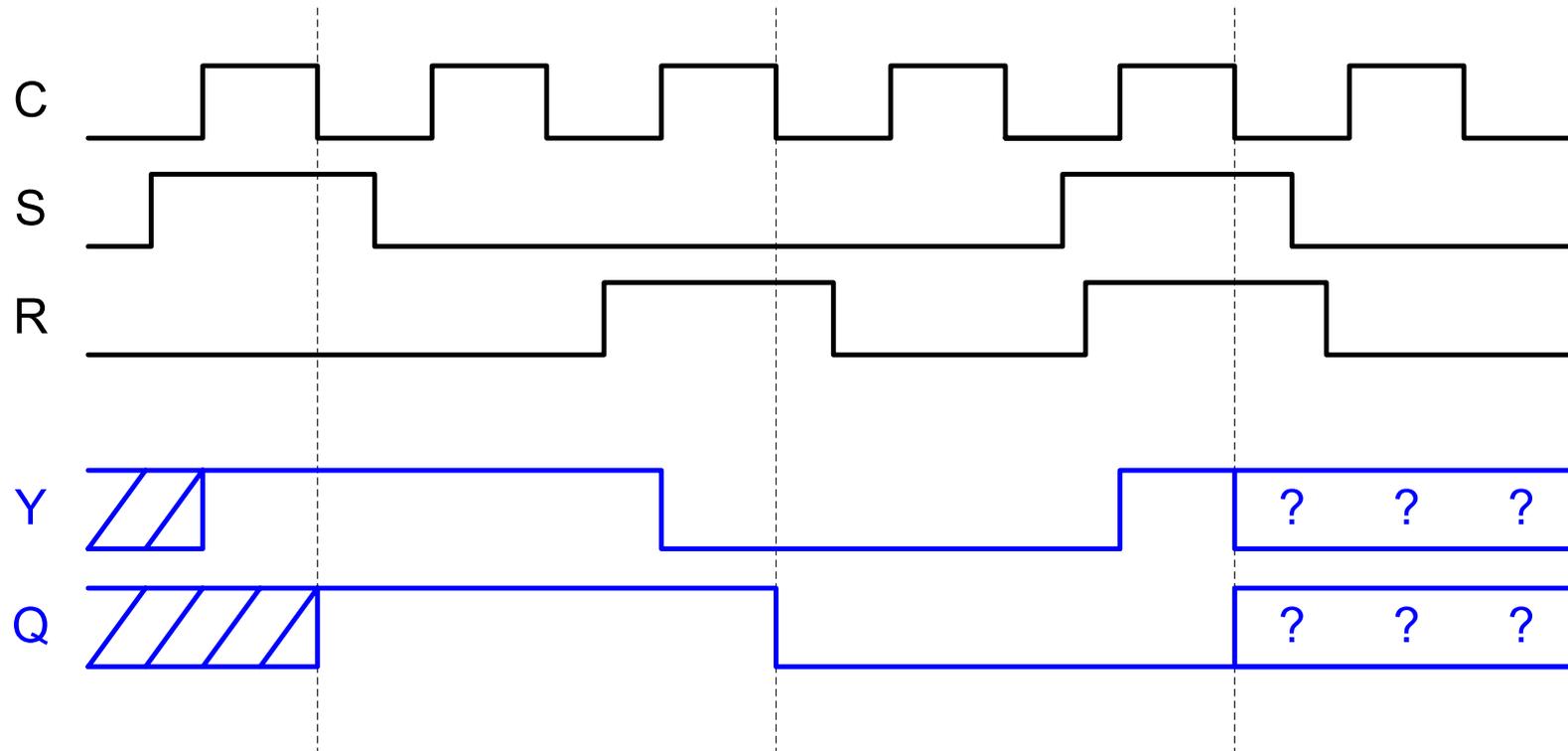
Flip-flops

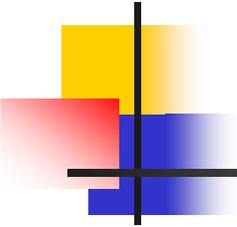
- Flip-flop SR Master-Slave



Flip-flops

■ Flip-flop SR Master-Slave





Flip-flops

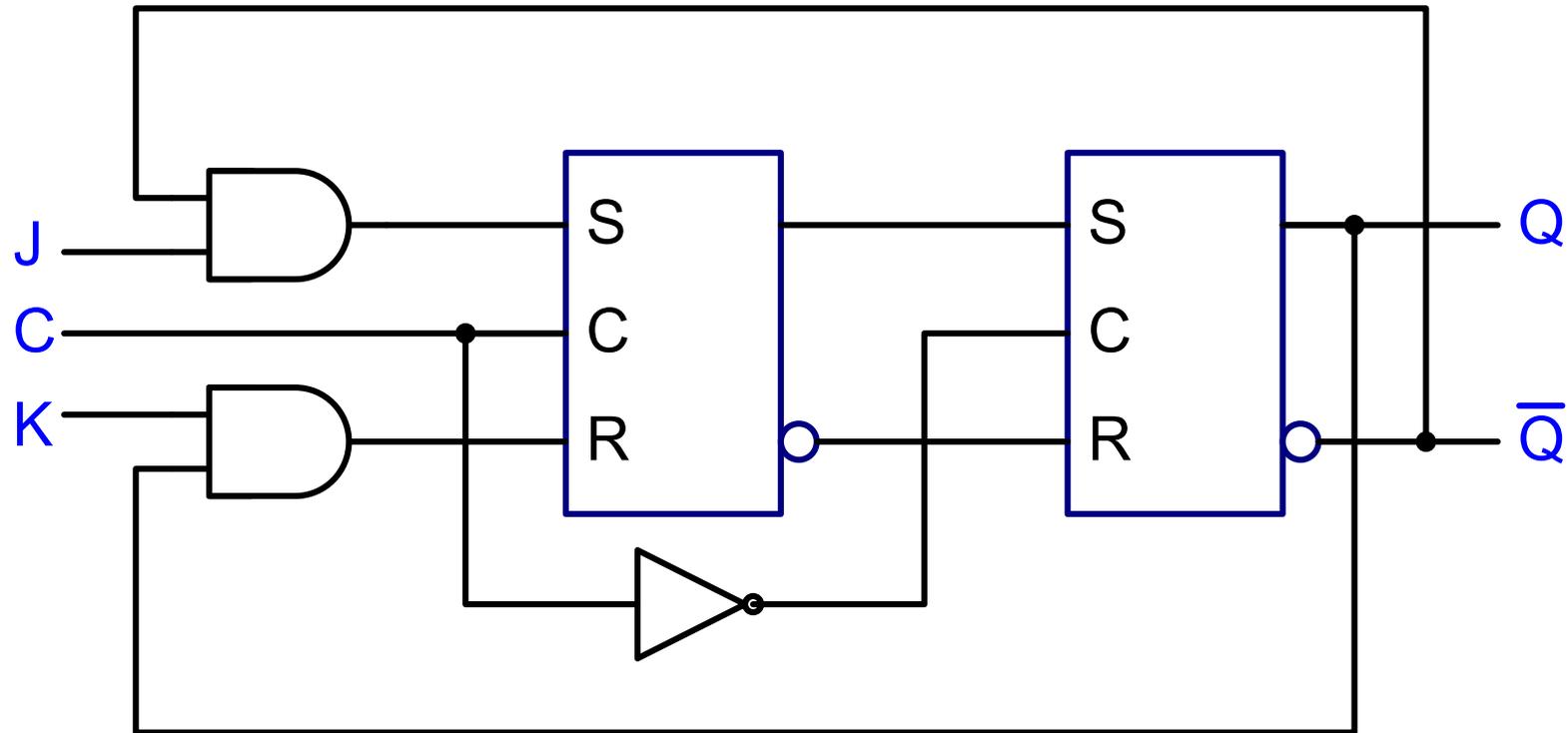
- *Flip-flop SR Master-Slave*

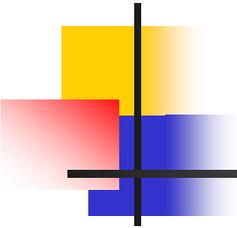
C	S	R	Q_{t+1}	Obs.
1	0	0	Q_t	Manter estado anterior
1	0	1	0	<i>Reset</i>
1	1	0	1	<i>Set</i>
1	1	1	?	Não utilizado (indefinido)

O *flip-flop SR Master-Slave* tem o problema do estado indefinido... Esta situação acontece quando ocorre um estímulo de C estando as entradas S e R a '1'.

Flip-flops

- *Flip-flop JK Master-Slave*





Flip-flops

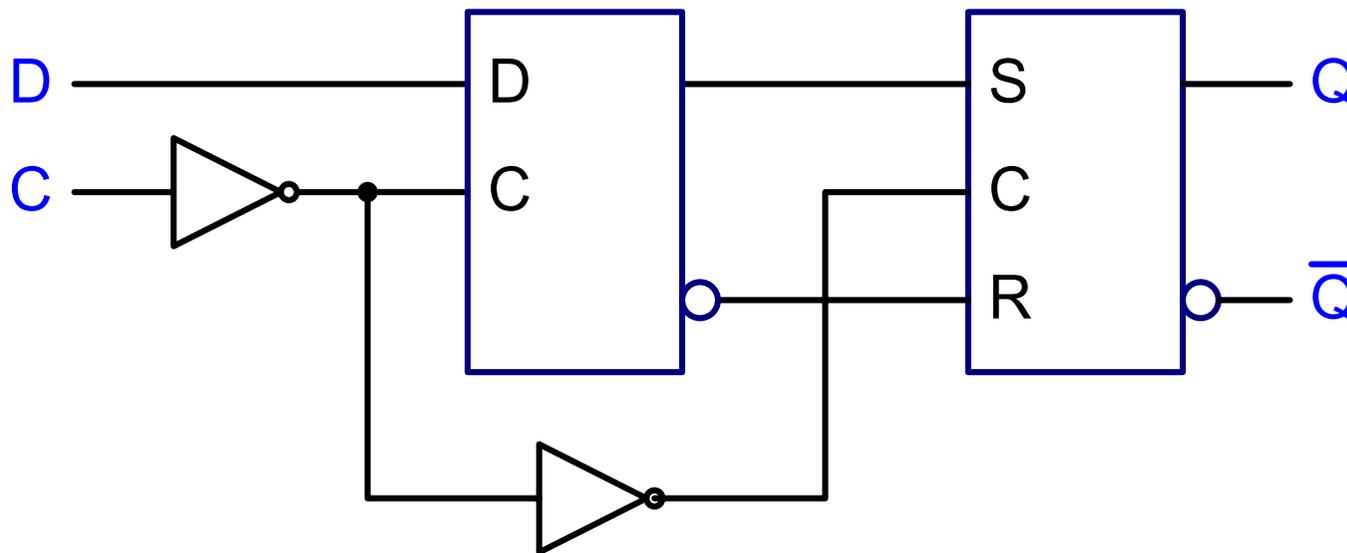
- *Flip-flop JK Master-Slave*

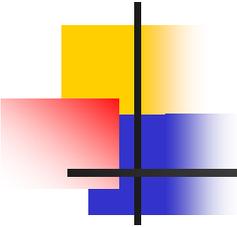
C	J	K	Q_{t+1}	Obs.
┐	0	0	Q_t	Manter estado anterior
┐	0	1	0	<i>Reset</i>
┐	1	0	1	<i>Set</i>
┐	1	1	$\overline{Q_t}$	Complementa estado anterior

O *flip-flop JK Master-Slave* não tem o problema do estado indefinido...

Flip-flops

- *Flip-flop D Edge-Triggered*





Flip-flops

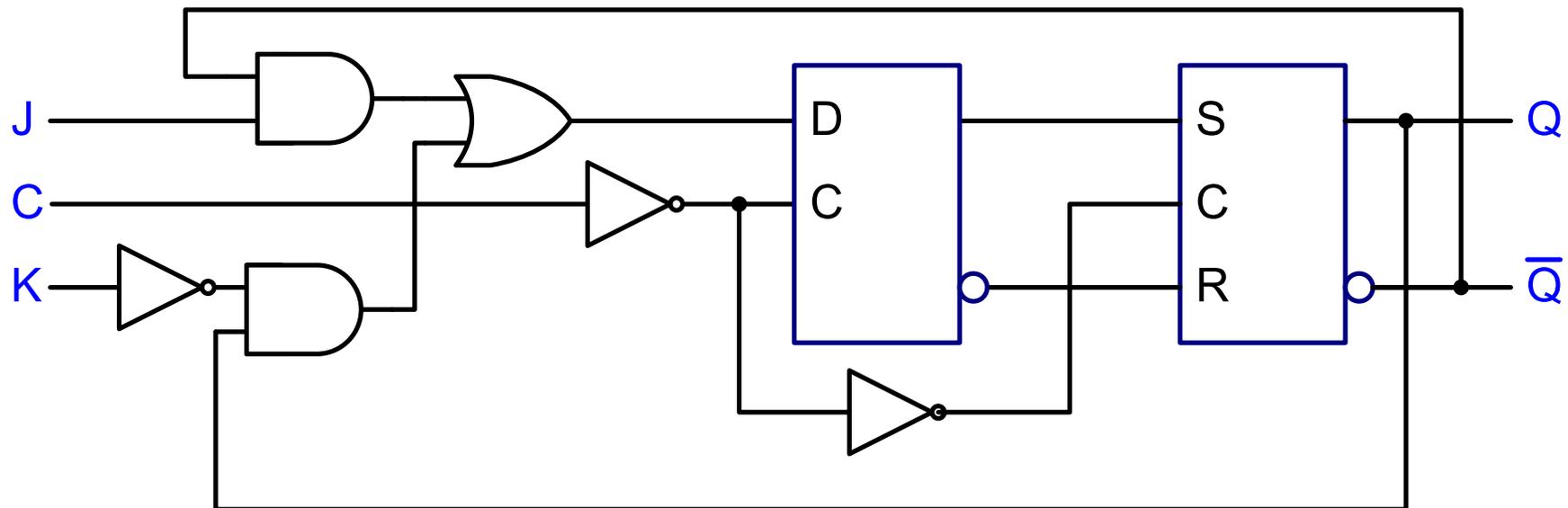
- *Flip-flop D Edge-Triggered*

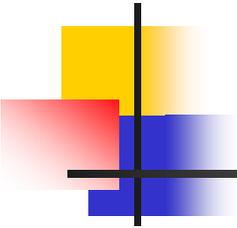
C	D	Q_{t+1}
	0	0
	1	1

Um *flip-flop edge-Triggered* só reage quando ocorre uma transição no nível lógico do sinal C (o FF do esquema só reage quando C varia de '0' para '1').

Flip-flops

- *Flip-Flop JK Edge-Triggered*





Flip-flops

- *Flip-flop JK Edge-Triggered*

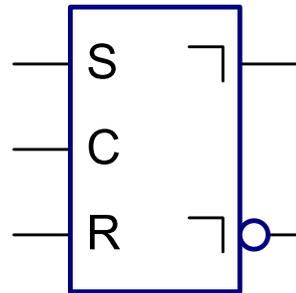
C	J	K	Q_{t+1}	Obs.
\uparrow	0	0	Q_t	Manter estado anterior
\uparrow	0	1	0	<i>Reset</i>
\uparrow	1	0	1	<i>Set</i>
\uparrow	1	1	$\overline{Q_t}$	Complementa estado anterior

Tal como o FF D edge-triggered, este flip-flop reage às transições do sinal C (neste caso as transições de '0' para '1').

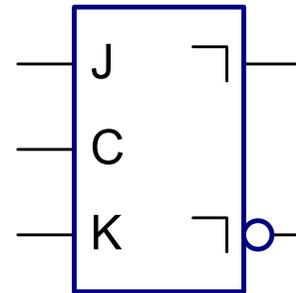
Flip-flops

■ Alguns símbolos

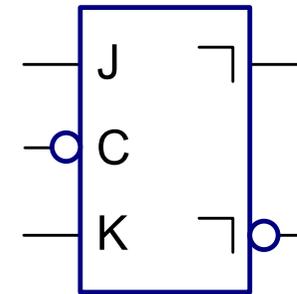
Master
Slave



SR positive pulse

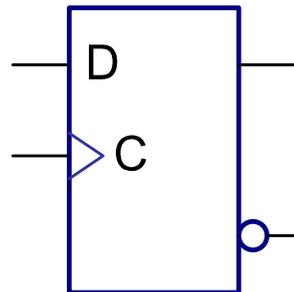


JK positive pulse

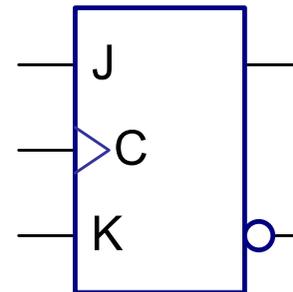


JK negative pulse

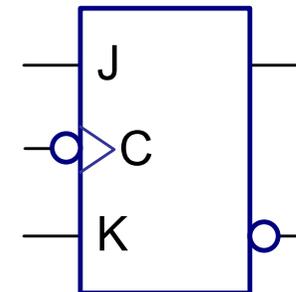
Edge
Triggered



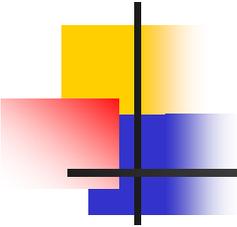
D positive
edge-triggered



JK positive
edge-triggered



JK negative
edge-triggered



Sumário (Aulas 10 a 13)

- Circuitos síncronos e assíncronos
- Análise de circuitos sequenciais
 - Tabela de transição de estados
 - Diagrama de estados
- Projecto de circuitos sequenciais
 - Modelos de *Mealy* e de *Moore*
 - Projecto com FFs D e com FFs JK
 - Exemplos
- Temporizações em circuitos sequenciais



Sistemas de Computação

Paulo Santos

Análise de Circuitos Sequenciais

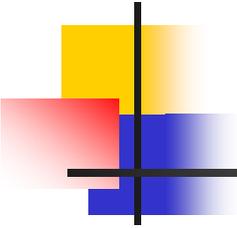


UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

IQF
Instituto para a Qualidade
na Formação, I.P.



Circuitos síncronos/assíncronos

- Circuito síncrono

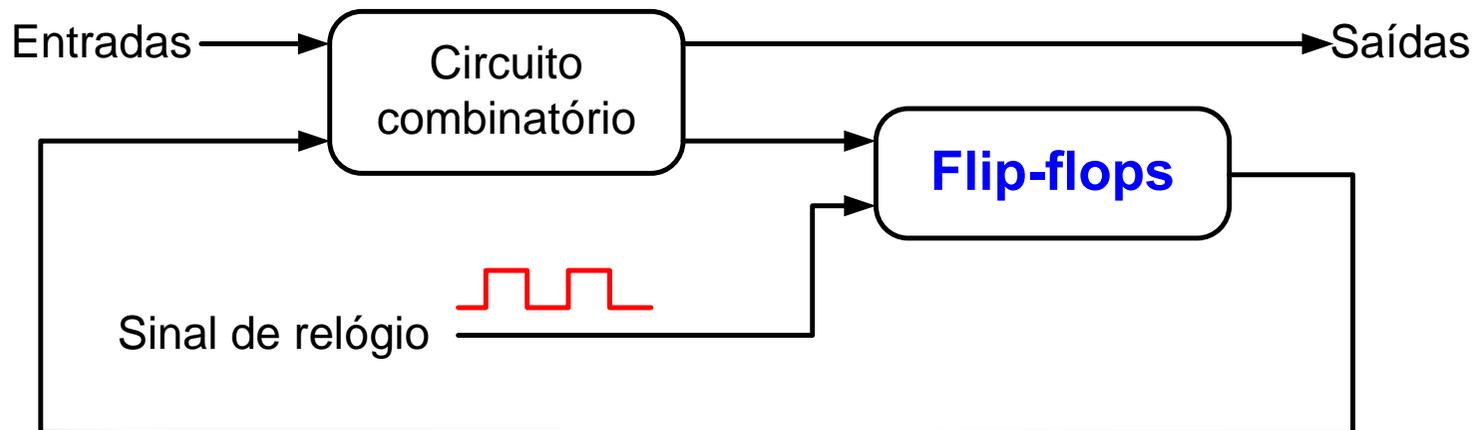
O circuito muda de estado em instantes temporais definidos por um sinal de referência – o **sinal de relógio**.

- Circuito assíncrono

As mudanças de estado do circuito são causadas pelas sequências de valores apresentadas nas entradas.

Circuitos Síncronos

- Nos circuitos sequenciais síncronos é o **sinal de relógio** (*Clock*) que faz reagir os flip-flops



Os geradores de sinais de relógio são habitualmente construídos a partir de cristais de quartzo

Tabelas Características

Relembrando a aula anterior...

Flip-flop SR

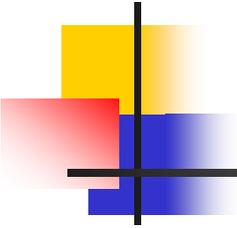
S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	?

Flip-flop JK

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q_t}$



As tabelas características dos flip-flops são fundamentais para a **análise** e o **projecto** de circuitos sequenciais.

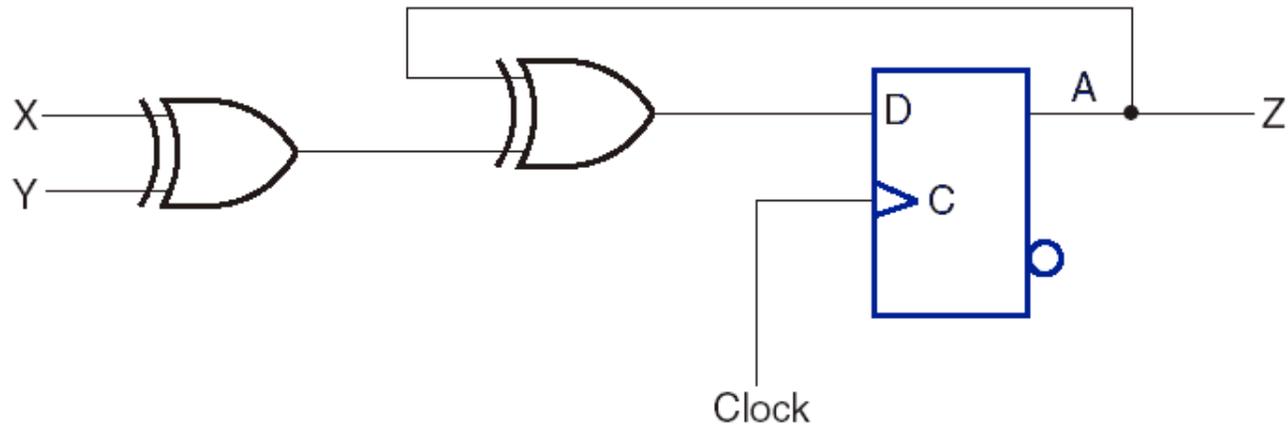


Análise de Circuitos Sequenciais

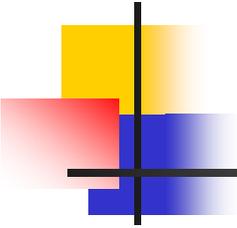
- Ferramentas para análise e projecto de circuitos sequenciais
 - Equações de entrada dos *flip-flops* e das saídas do circuito
 - Tabela de transição de estados do circuito
 - Diagrama de estados

Análise de Circuitos Sequenciais

- Exemplo:



- Entradas: X e Y
- Saídas: Z



Análise de Circuitos Sequenciais

- Tabela de estados

Estado actual	Entradas		Próximo estado	Saída
A_t	X	Y	A_{t+1}	Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

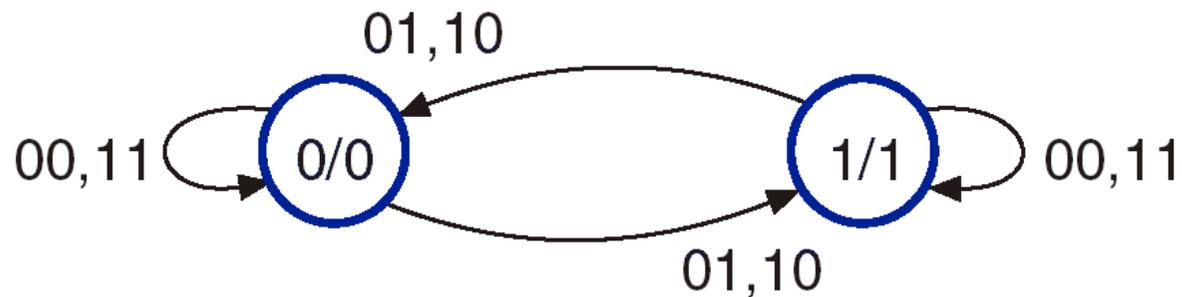
Análise de Circuitos Sequenciais

- Equações

- Entrada dos Flip-flops: $D_A = A \oplus (X \oplus Y)$

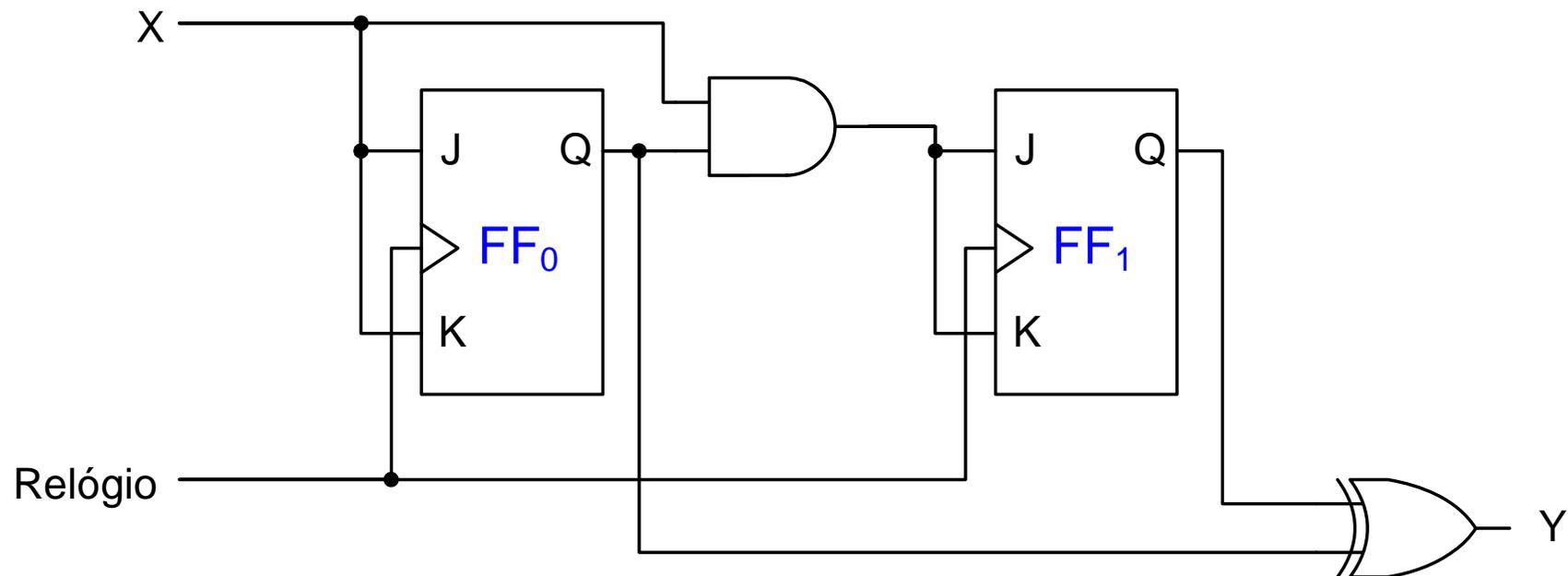
- Saídas do circuito: $Z = A$

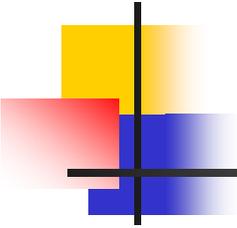
- Diagrama de estados



Análise de Circuitos Sequenciais

- Outro exemplo:





Análise de Circuitos Sequenciais

- Equações

- Entradas dos FFs:

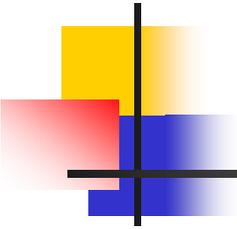
$$J_0 = K_0 = X$$

$$J_1 = K_1 = XQ_0$$

- Saída:

$$Y = Q_0 \oplus Q_1$$

Estado		Saída
Q_1	Q_0	Y
0	0	0
0	1	1
1	0	1
1	1	0



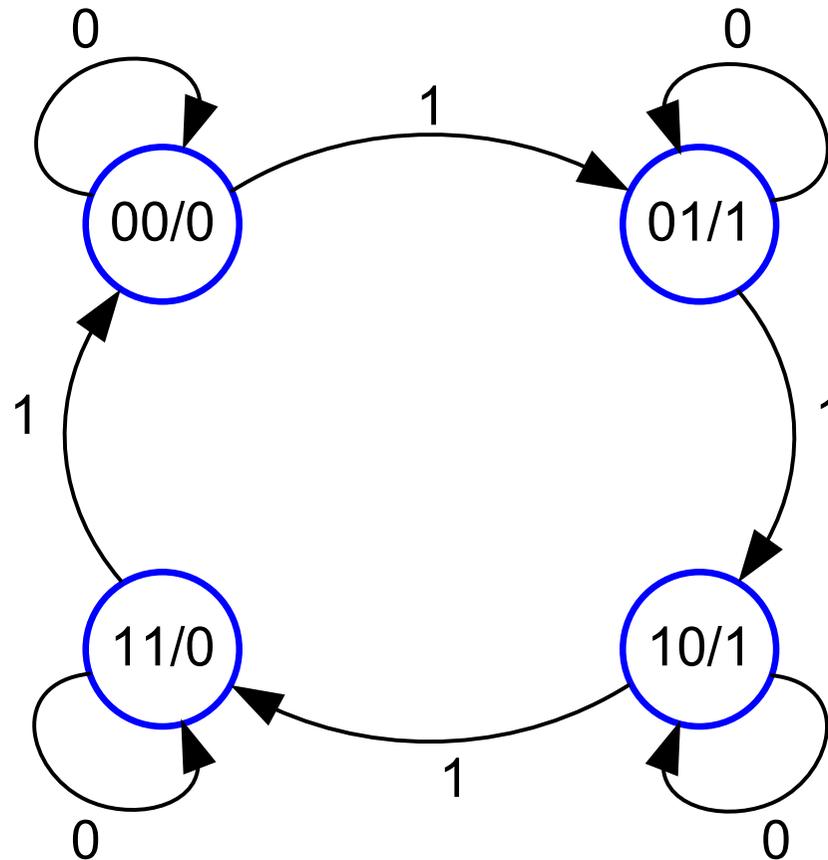
Análise de Circuitos Sequenciais

- Tabela de transição de estados

Estado actual			Próximo estado	
Q_1	Q_0	X	Q_1'	Q_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Análise de Circuitos Sequenciais

- Diagrama de estados





Sistemas de Computação

Paulo Santos

Projecto de Circuitos Sequenciais



UNIÃO EUROPEIA

Fundo Social Europeu

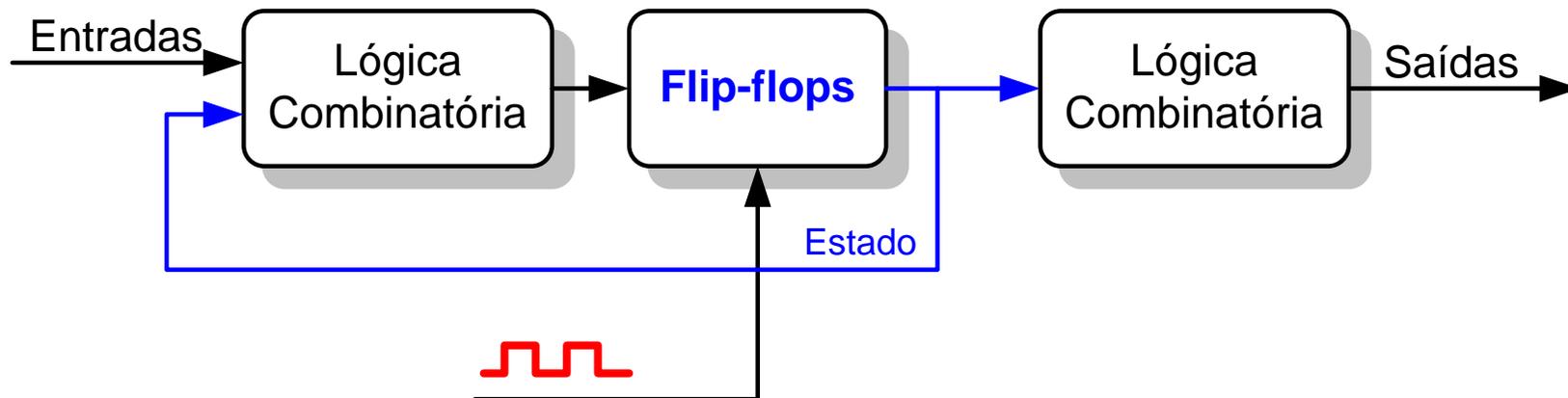
prime
Programa de Incentivos à
Modernização da Economia

IQF
Instituto para a Qualidade
na Formação, I.P.

Modelos de Circuitos Sequenciais

■ Modelo de Moore

- Os valores nas saídas dependem apenas do estado do circuito

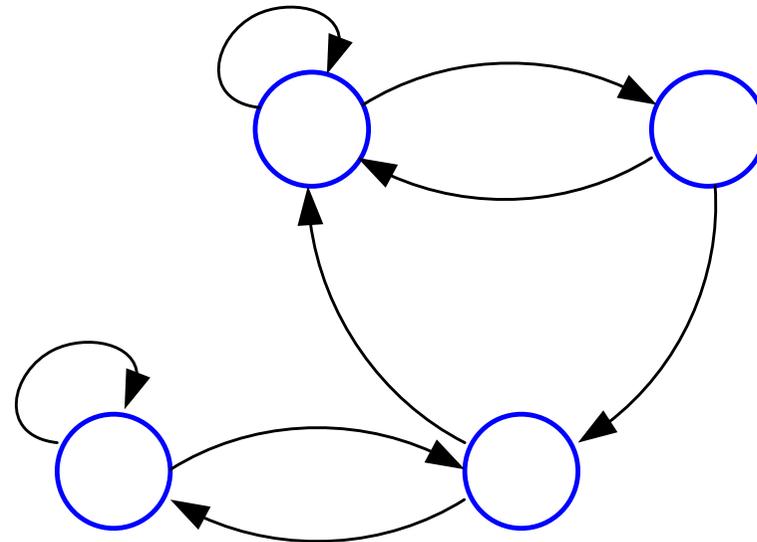
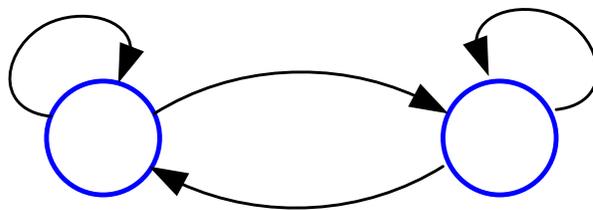


Sendo assim, as saídas só podem mudar quando o estado muda.

Modelos de Circuitos Sequenciais

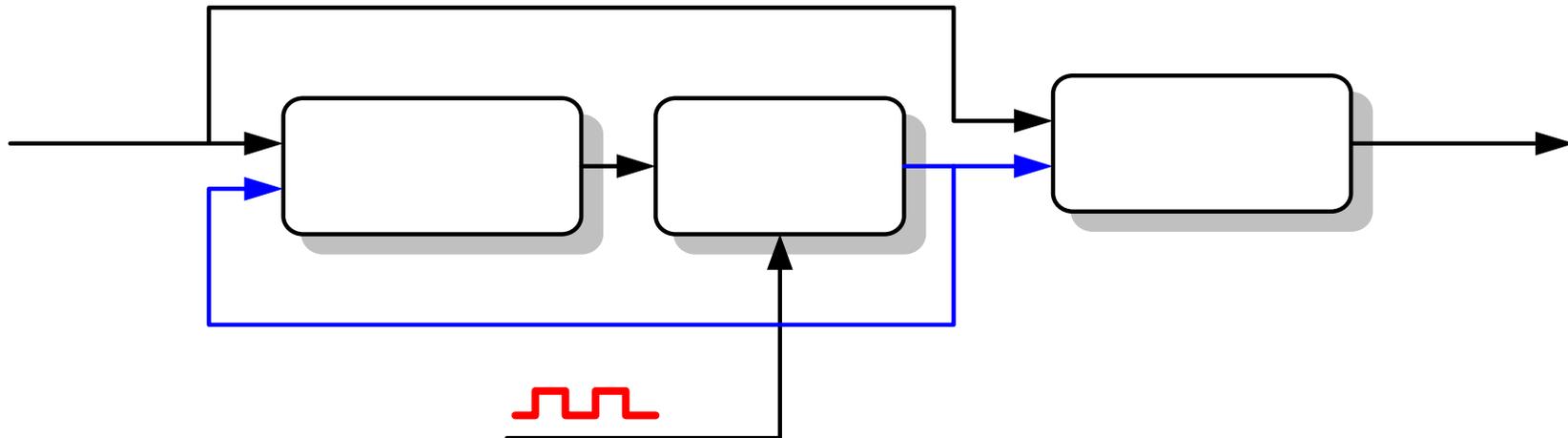
■ Modelo de Moore

- No diagrama de estados, as saídas aparecem associadas a cada estado (dentro das “bolas”)
- Exemplos:



Modelos de Circuitos Sequenciais

- Modelo de Mealy
 - Os valores nas saídas dependem do estado e do valor das entradas

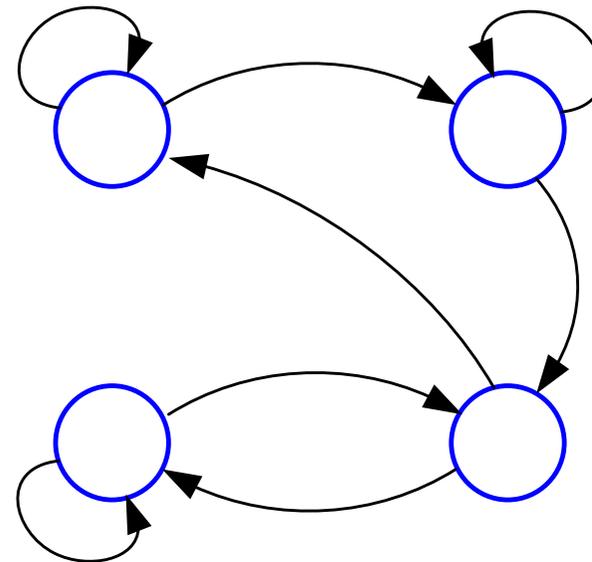
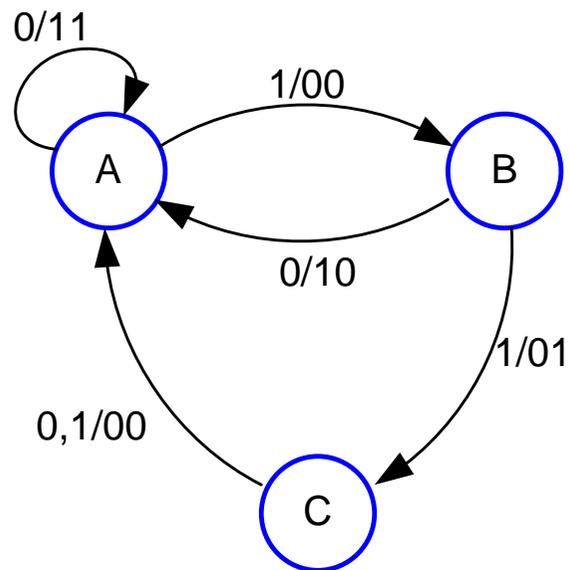


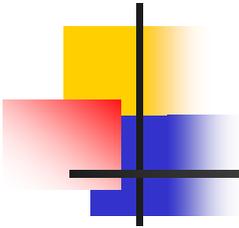
Uma alteração nos valores das entradas pode causar imediatamente uma alteração nos valores das saídas.

Modelos de Circuitos Sequenciais

- Modelo de Mealy

- No diagrama de estados, o valor das saídas é representado junto das entradas (nas “setas”)
- Exemplos:





Procedimentos de Projecto

- A partir da especificação, obter o **diagrama de estados** (modelo de *Moore* ou *Mealy*)
- Atribuir **códigos binários** a cada estado do diagrama
- Obter a **tabela de estados**
- Escolher o tipo de *flip-flops* a utilizar
- Obter as **equações de entrada** de cada *flip-flop*
- Obter as **equações das saídas**
- Desenhar o circuito

Projecto com *Flip-flops* D

Pretende-se obter o circuito correspondente ao seguinte diagrama de estados. Vai-se projectar o circuito utilizando *flip-flops* D.

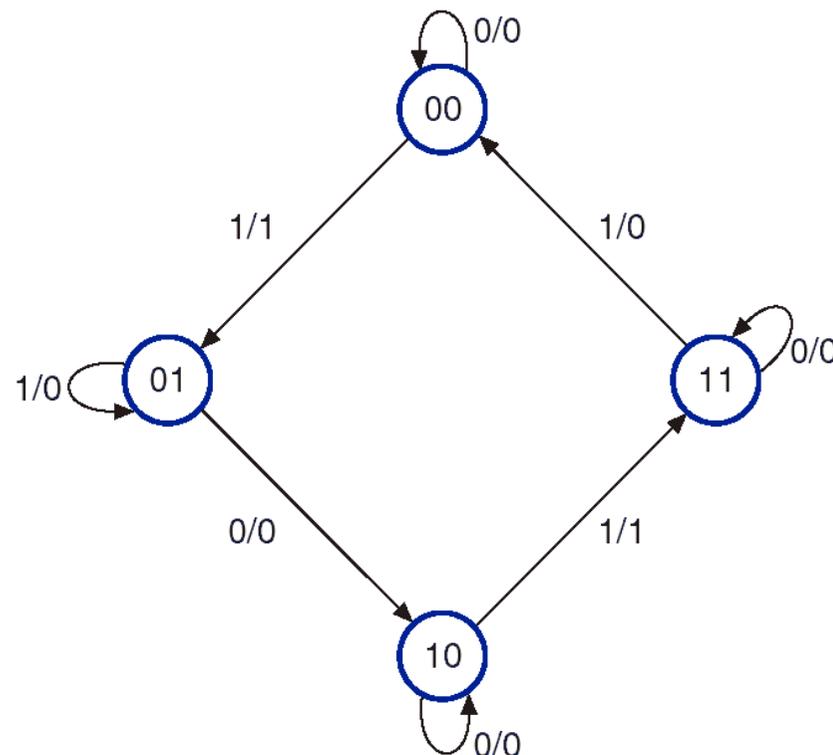
Entrada: X

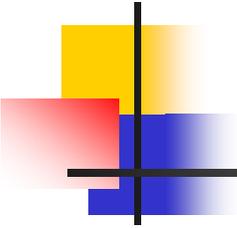
Saída: Y

Nº de estados: 4

Nº de flip-flops: 2

Modelo: Mealy





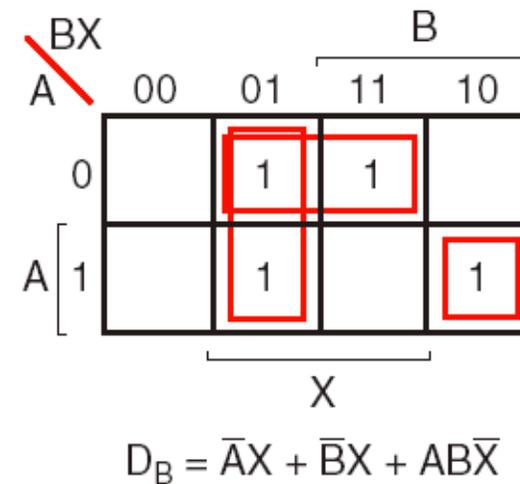
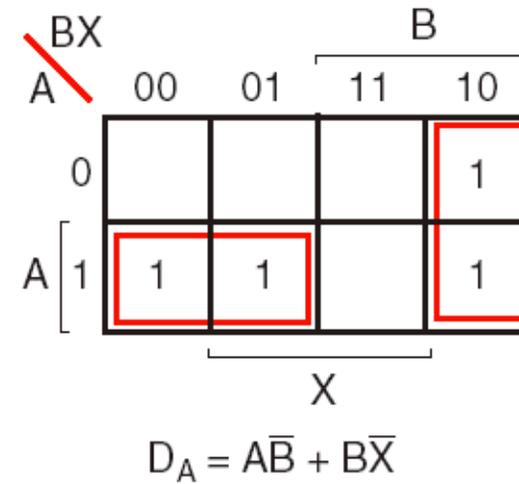
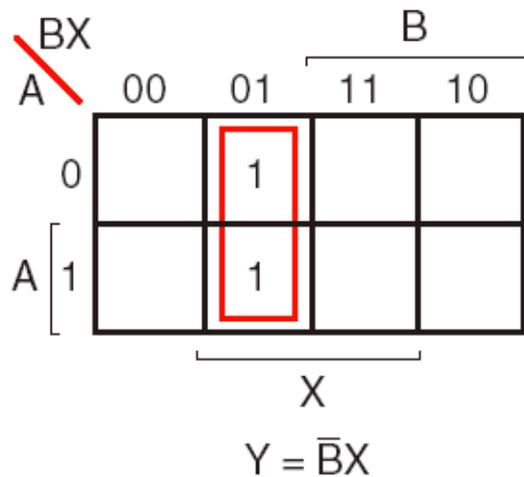
Projecto com *Flip-flops* D

- Tabela de estados

Estado actual		Entrada	Próximo estado		Saída
A	B	X	A'	B'	Y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

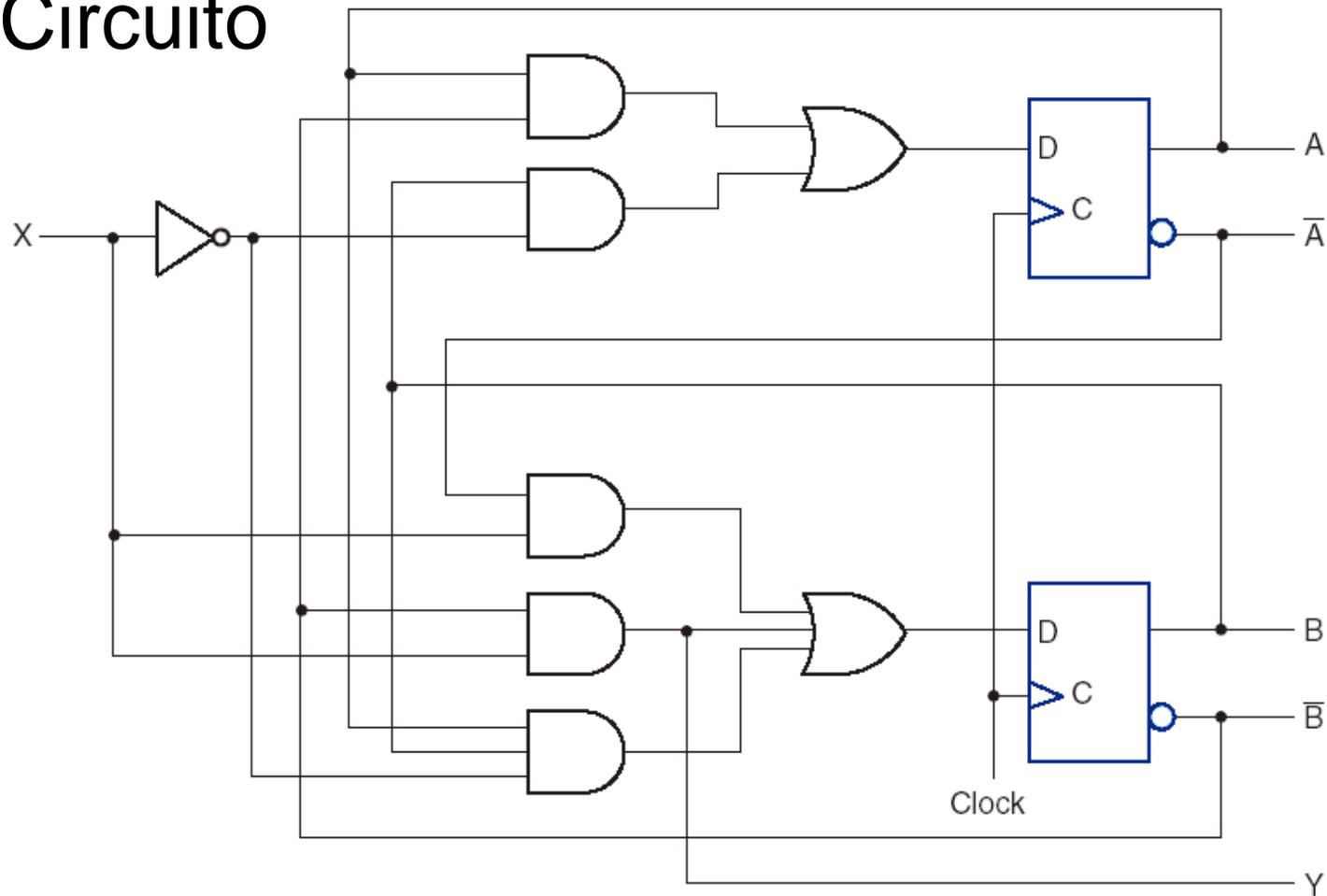
Projecto com *Flip-flops D*

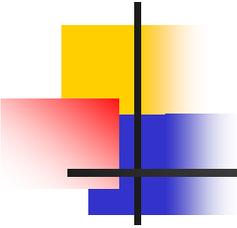
Equações



Projecto com *Flip-flops D*

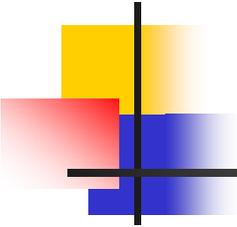
■ Circuito





Projecto com *Flip-flops* JK

- Projecto com *flip-flops* JK
 - Quando se projectam circuitos com *flip-flops* D, as equações à entrada dos *flip-flops* são obtidas directamente a partir do próximo estado.
 - Com *flip-flops* JK, será necessário derivar equações para as entradas J e K de cada *flip-flop*. Isso poderá ser realizado com base nas **tabelas de excitação** dos *flip-flops*.



Projecto com *Flip-flops* JK

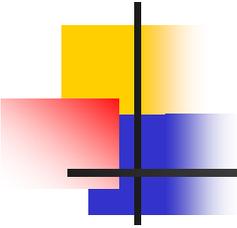
- Tabelas de excitação

Flip-flop JK

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Flip-flop SR

Q_t	Q_{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0



Projecto com *Flip-flops* JK

- Tabelas de excitação

Flip-flop D

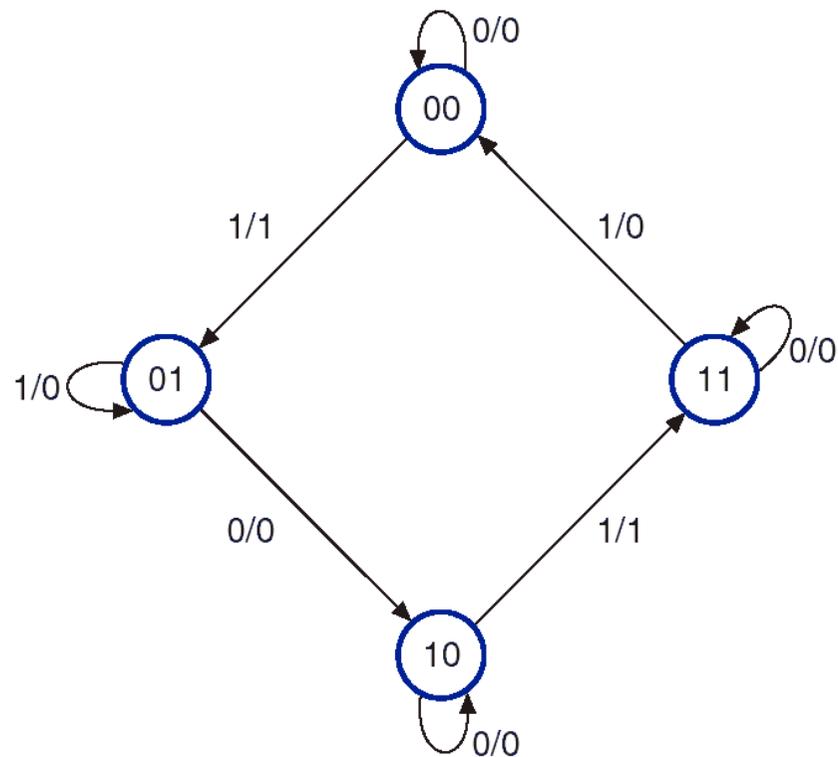
Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

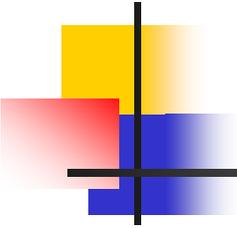
Flip-flop T

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Projecto com *Flip-flops* JK

Pretende-se realizar um circuito correspondente ao diagrama de estados anterior, mas utilizando flip-flops JK.





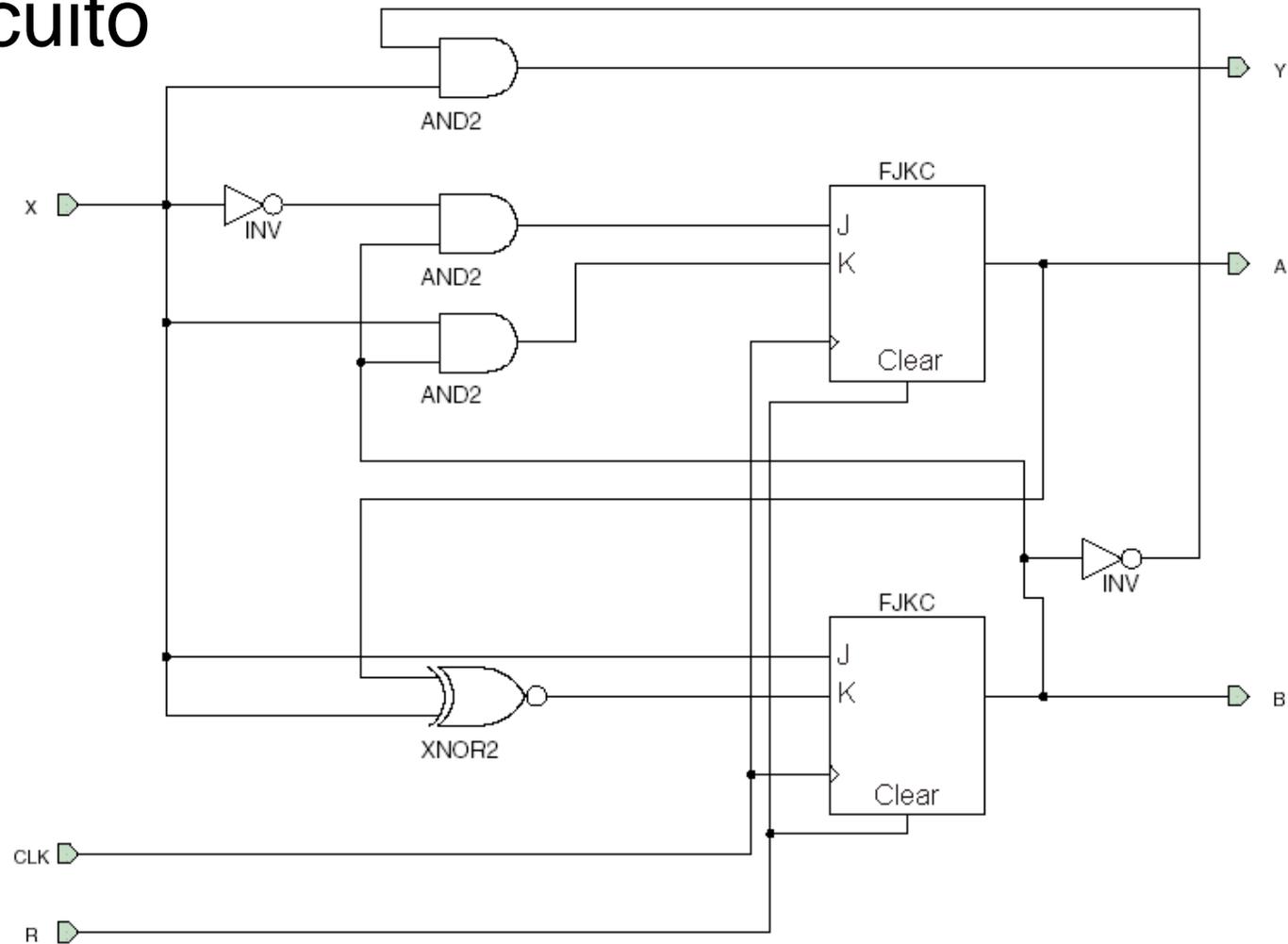
Projecto com *Flip-flops* JK

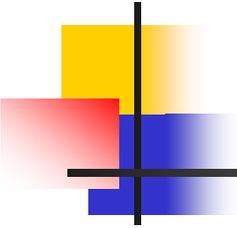
■ Tabela de estados

Estado actual		Entrada	Próximo estado		Entradas dos flip-flops				Saída
A	B	X	A'	B'	J _A	K _A	J _B	K _B	Y
0	0	0	0	0	0	X	0	X	0
0	0	1	0	1	0	X	1	X	1
0	1	0	1	0	1	X	X	1	0
0	1	1	0	1	0	X	X	0	0
1	0	0	1	0	X	0	0	X	0
1	0	1	1	1	X	0	1	X	1
1	1	0	1	1	X	0	X	0	0
1	1	1	0	0	X	1	X	1	0

Projecto com *Flip-flops JK*

■ Circuito





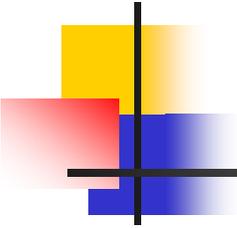
FFs D vs. FFs JK

- *Flip-flops D*

- O projecto do circuito é mais simples. mas o circuito resultante é geralmente mais complexo (mais lógica combinatória)

- *Flip-flops JK*

- O projecto do circuito é mais complicado, mas o circuito resultante é normalmente mais simples



Exemplos

- **Detector de sequências**

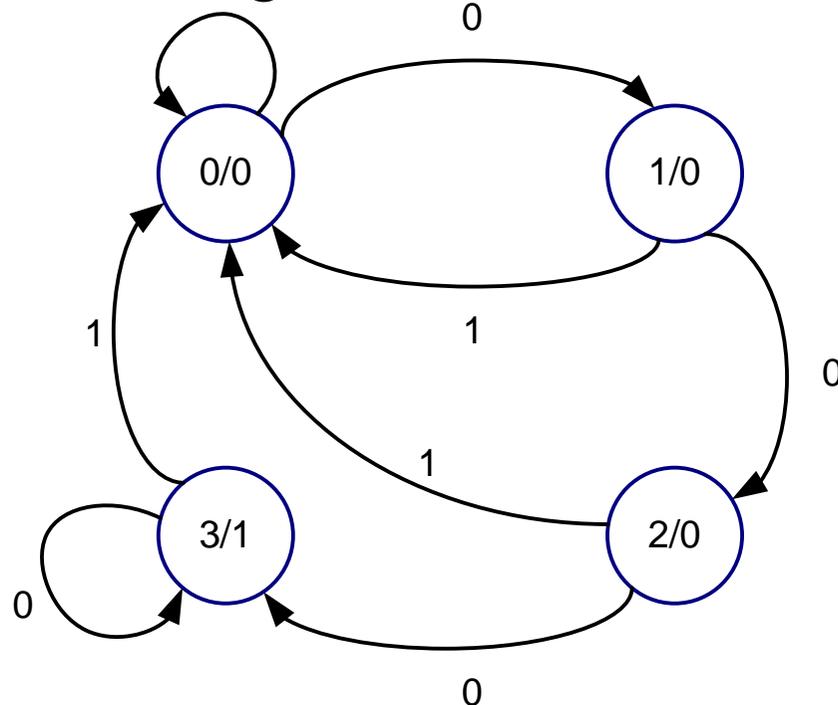
Pretende-se projectar um circuito sequencial com uma entrada série, designada X , em que entram bits ao mesmo ritmo do sinal de relógio.

A saída do circuito deverá ser '1' quando é apresentado o valor '0' à entrada durante 3 ou mais impulsos de relógio consecutivos. Nos restantes casos, a saída deverá ser '0'.

O projecto deverá seguir o modelo de Moore.

Exemplos

■ Diagrama de estados



Codificação dos estados:

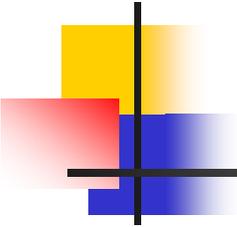
0 – 00

1 – 01

2 – 10

3 – 11

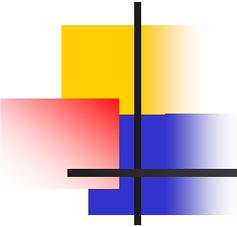
4 estados \Rightarrow 2 Flip-flops



Exemplos

- Tabela de transições de estados

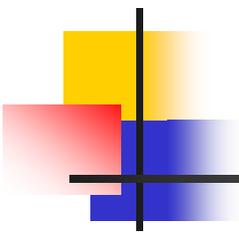
Estado actual			Próximo estado	
A	B	X	A'	B'
0	0	0	0	1
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0



Exemplos

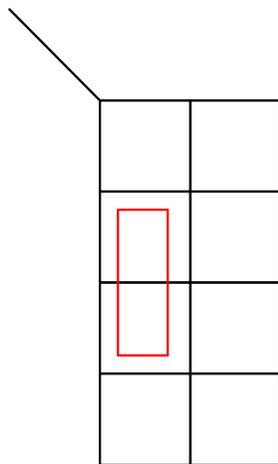
- Projecto com *flip-flops* JK

Estado actual			Próximo estado					
A	B	X	A'	B'	J _A	K _A	J _B	K _B
0	0	0	0	1	0	x	1	x
0	0	1	0	0	0	x	0	x
0	1	0	1	0	1	x	x	1
0	1	1	0	0	0	x	x	1
1	0	0	1	1	x	0	1	x
1	0	1	0	0	x	1	0	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1



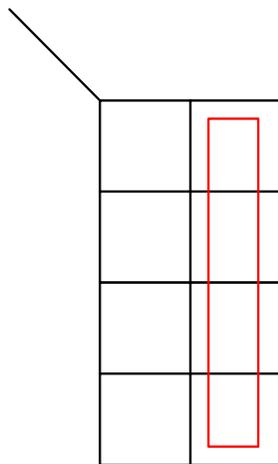
Exemplos

- Equações de entrada nos *flip-flops*



$$J_A = B\bar{X}$$

AB

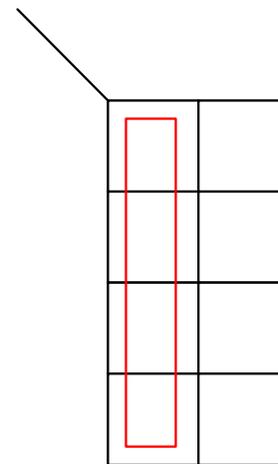


$$K_A = X \quad \mathbf{J_A}$$

X

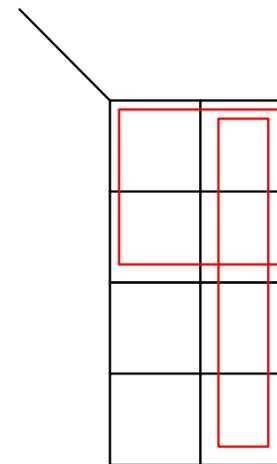
0

1



$$J_B = \bar{X}$$

AB



$$K_B = X + \bar{A}K_A$$

X

0

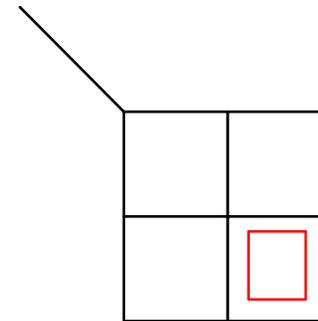
¹⁸⁷

1

Exemplos

- Equação da saída

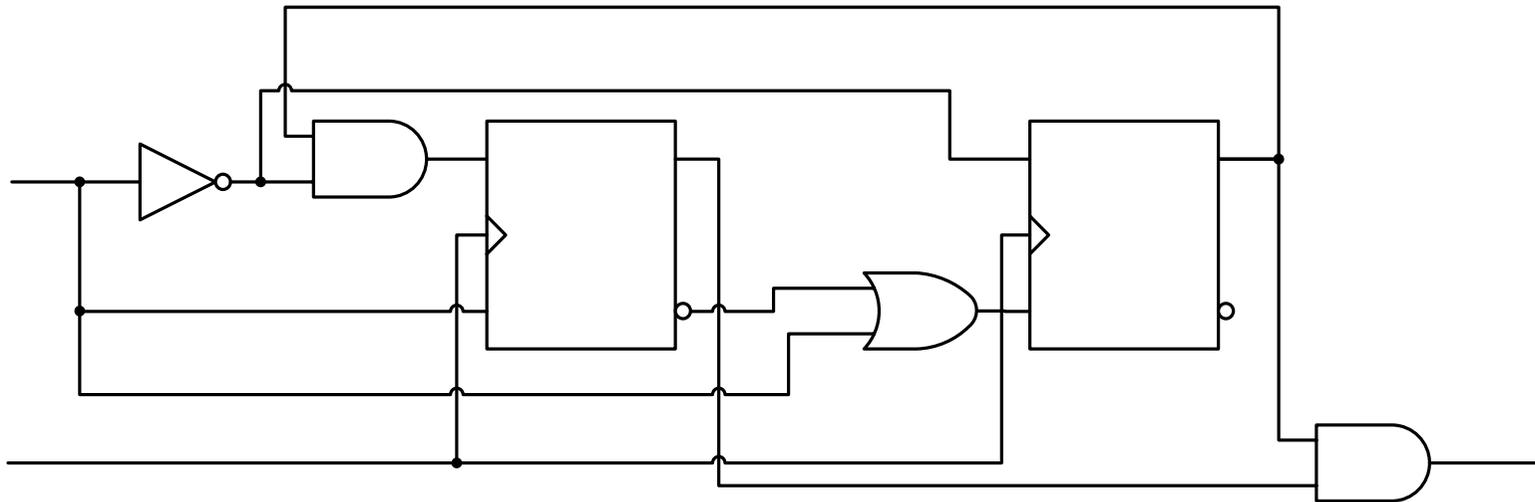
Estado actual		Saída
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



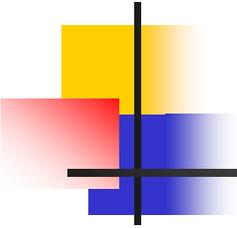
Como a máquina projectada segue o modelo de Moore, a equação da saída depende apenas do estado.

Exemplos

- Circuito resultante



X



Exemplos

■ Divisor de frequência

Pretende-se projectar um circuito sequencial com duas entradas, designadas S_1 e S_0 , segundo o modelo de Moore.

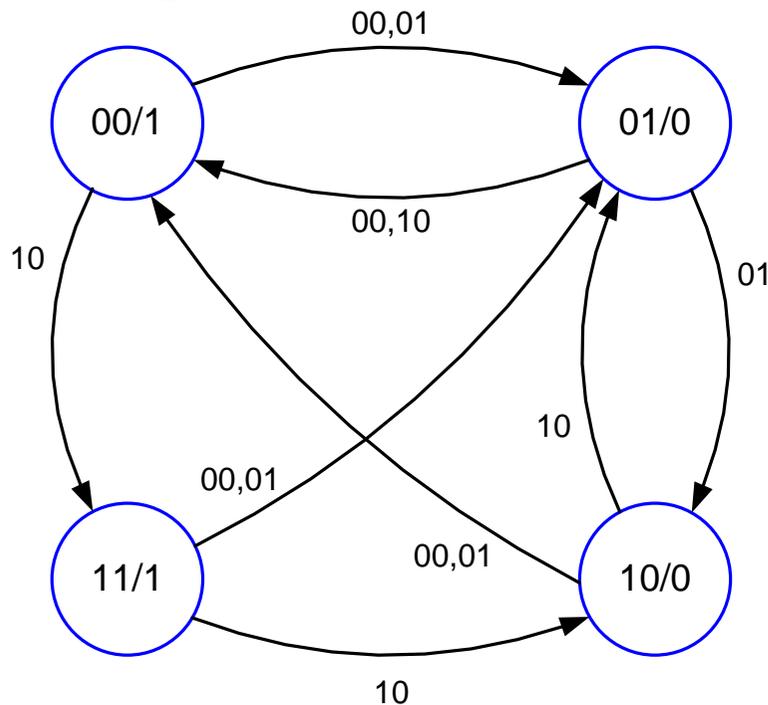
Consoante os valores S_1S_0 apresentados à entrada, a saída deverá seguir uma das seguintes sequências binárias:

- Entradas 00 – Sequência 10101010...
- Entradas 01 – Sequência 100100100...
- Entradas 10 – Sequência 11001100...
- Entradas 11 – Não especificado.

Na prática, e tendo em conta as formas de onda da saída, este circuito comporta-se como um divisor da frequência de relógio.

Exemplos

■ Diagrama de estados



Entradas: S_1 e S_0

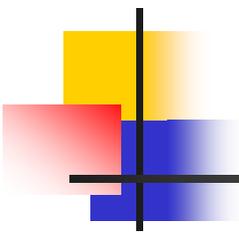
Saída: Y

Variáveis de estado: Q_1 e Q_0

Exemplos

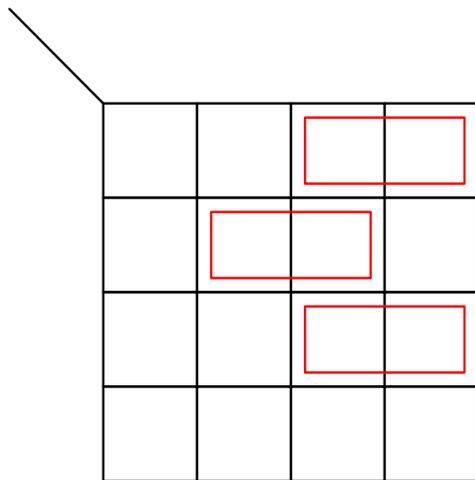
- Tabela de transição de estados

Estado Actual		Entradas		Próximo Estado	
Q ₁	Q ₀	S ₁	S ₀	Q ₁ '	Q ₀ '
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	X	X
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	X	X
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	X	X
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	X	X



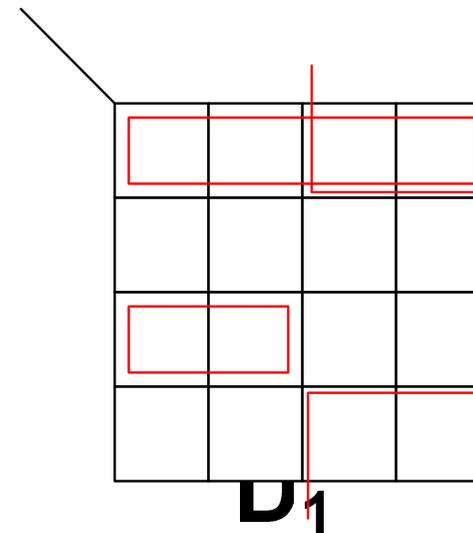
Exemplos

- Equações de entrada dos flip-flops



$$\begin{aligned}
 D_1 &= \bar{Q}_1 \bar{Q}_0 S_1 + Q_1 Q_0 S_1 + \bar{Q}_1 Q_0 S_0 \\
 &= S_1 (\overline{Q_1 \oplus Q_0}) + \bar{Q}_1 Q_0 S_0
 \end{aligned}$$

$Q_1 Q_0$



$$D_0 = \bar{Q}_1 \bar{Q}_0 + \bar{Q}_0 S_1 + Q_1 Q_0 \bar{S}_1$$

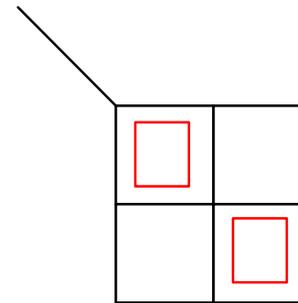
$00 \quad 01 \quad 11 \quad 10$

$00 \quad 0 \quad 0 \quad X \quad 1$

Exemplos

- Equação da saída

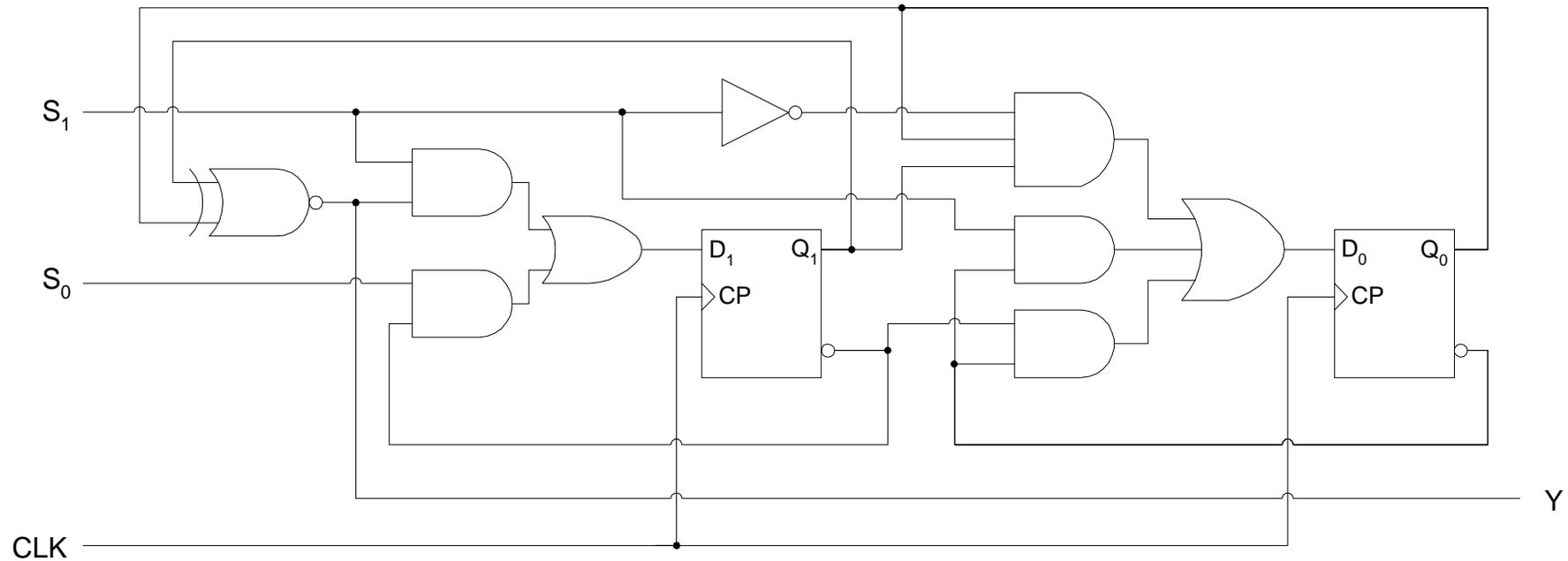
Estado actual		Saída
Q_1	Q_0	Y
0	0	1
0	1	0
1	0	0
1	1	1

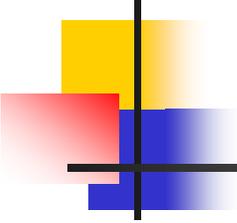


$$Y = \overline{Q_1} \oplus Q_0$$

Exemplos

- Circuito resultante



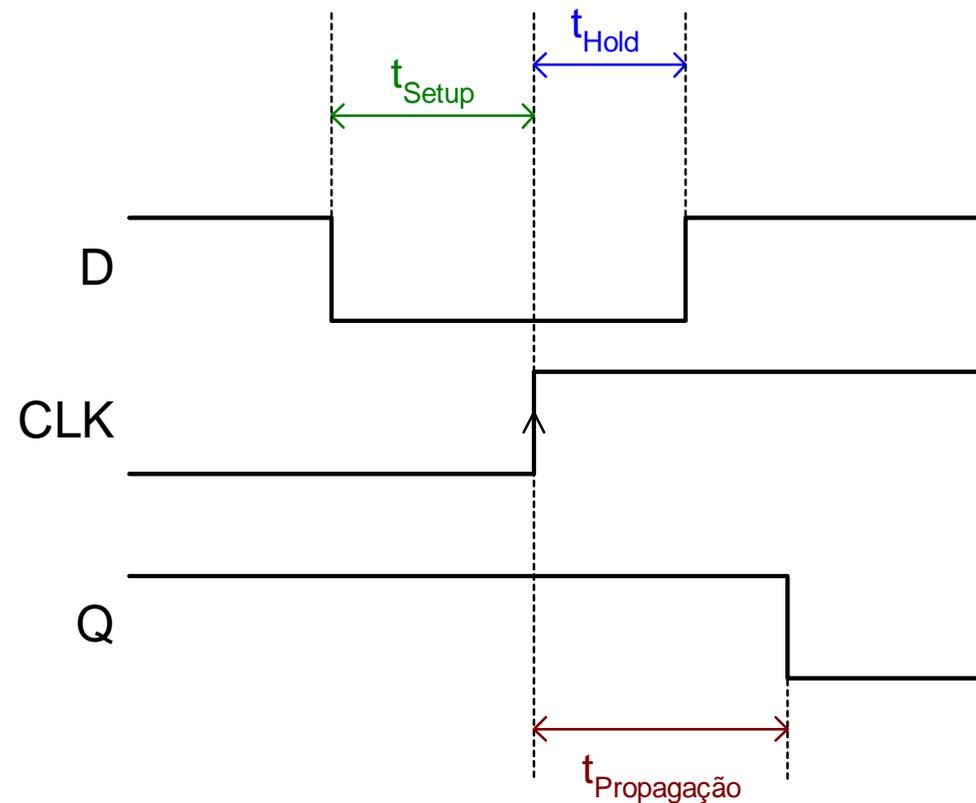


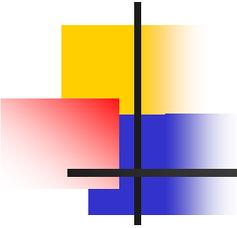
Temporizações

- Temporizações típicas de um flip-flop
 - Tempo de preparação (*Setup time*)
tempo necessário ter a entrada estável antes do *flip-flop* reagir
 - Tempo de manutenção (*Hold time*)
tempo necessário ter a entrada estável após o *flip-flop* reagir
 - Tempo de propagação
tempo que demora até que saída se encontre estável, após a reacção do *flip-flop*

Temporizações

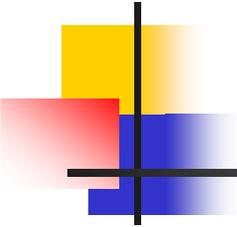
- Formas de onda
(flip-flop D *positive edge-triggered* com saída Q)





Frequência máxima

- No projecto de circuitos sequenciais, é importante saber qual a frequência máxima de relógio para a qual o circuito funciona correctamente
- A essa frequência máxima corresponde um período mínimo do sinal de relógio
- Após um impulso de relógio será necessário garantir as entradas dos flip-flops estejam preparadas antes de chegar o próximo impulso de relógio



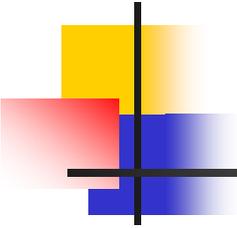
Frequência máxima

- Sendo assim, ao período mínimo de funcionamento corresponde a situação em que se verifica o pior caso de propagação existente nos vários caminhos entre as saídas dos flip-flops e suas entradas.
- Também é necessário ter em conta o tempo de propagação dos flip-flops e o tempo de preparação
- Deste modo, vem para o período mínimo:

$$T_{\min} = t_{\text{PDFF}} + T_{\text{PD caminho crítico}} + t_{\text{SetupFF}}$$

- E para a frequência máxima de funcionamento:

$$f_{\max} = \frac{1}{T_{\min}} = \frac{1}{t_{\text{PDFF}} + T_{\text{PD caminho crítico}} + t_{\text{SetupFF}}}$$



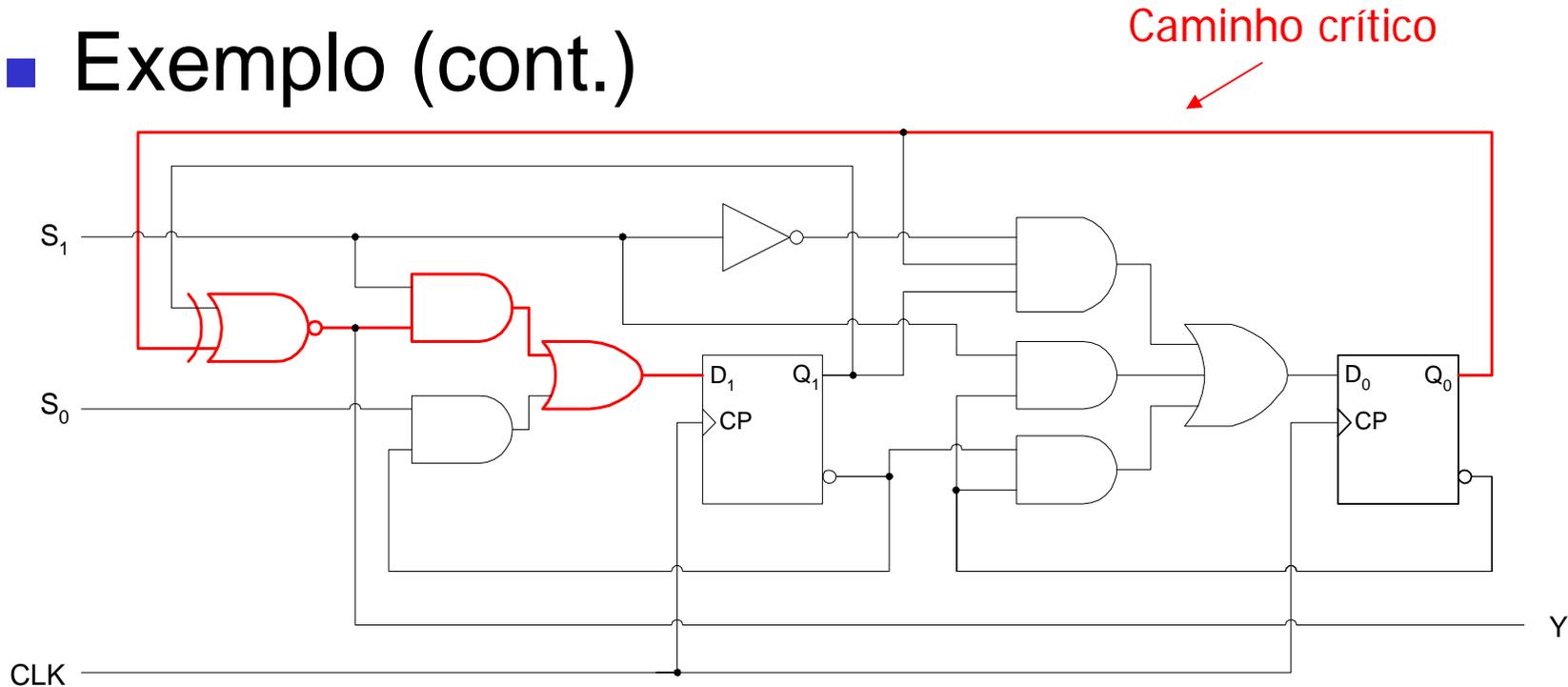
Frequência máxima

- Exemplo

- Qual será a máxima frequência de funcionamento do circuito “divisor de frequência” obtido anteriormente ?
- Suponha as seguintes temporizações:
 - t_{PD} portas lógicas = 5ns
 - t_{PD} flip-flops = 10ns
 - t_{Setup} flip-flops = 5ns
 - t_{Hold} flip-flops = 4ns
 - Suponha ainda que $t_{PLH} = t_{pHL}$ em todos os casos

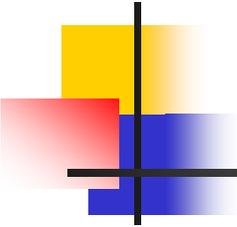
Frequência máxima

■ Exemplo (cont.)



$$\begin{aligned} T_{\min} &= t_{\text{PDFF}} + T_{\text{PD caminho crítico}} + t_{\text{SetupFF}} \\ &= 10\text{ns} + 3 \times 5\text{ns} + 5\text{ns} \\ &= 30\text{ns} \end{aligned}$$

$$f_{\max} = \frac{1}{T_{\min}} = \frac{1}{30\text{ns}} \approx 33,3\text{MHz}$$



Sumário (Aula 14)

- Registos
 - Estrutura de um registo
 - Inicialização
 - Carregamento paralelo
 - *Clock gating* e falhas de sincronização
 - Carregamento paralelo síncrono
 - Operações de deslocamento
 - Exemplos de registos



Sistemas de Computação

Paulo Santos

Registos

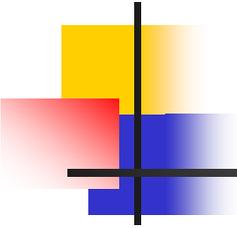


UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia



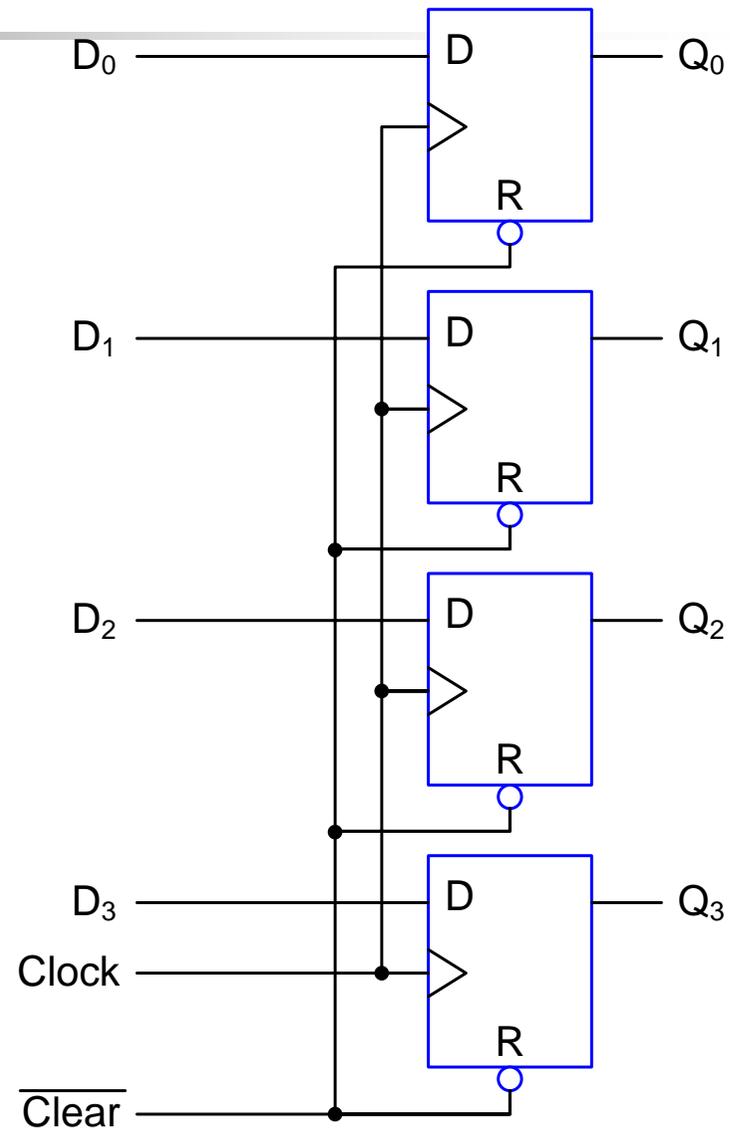
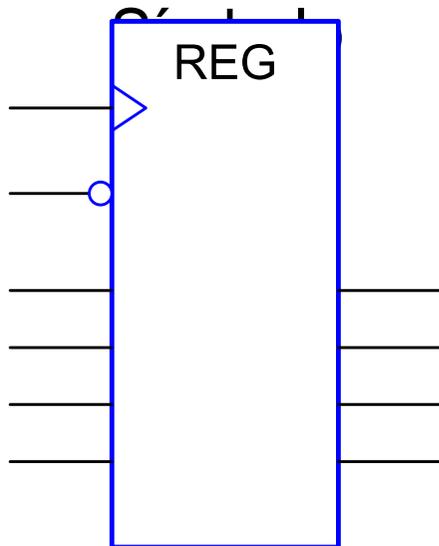


Registos

- **Registo**
 - Conjunto de *flip-flops* e lógica adicional que permite armazenamento e manipulação de informação
 - *Flip-flops* – guardam a informação
 - Um *flip-flop* por cada bit que se pretende guardar
 - Lógica adicional – permite operações sobre os dados
 - Inicializar os valores do registo a zero – **CLEAR**
 - Carregar o registo com novos dados – **LOAD**
 - Deslocar os dados dentro do registo – **SHIFT**

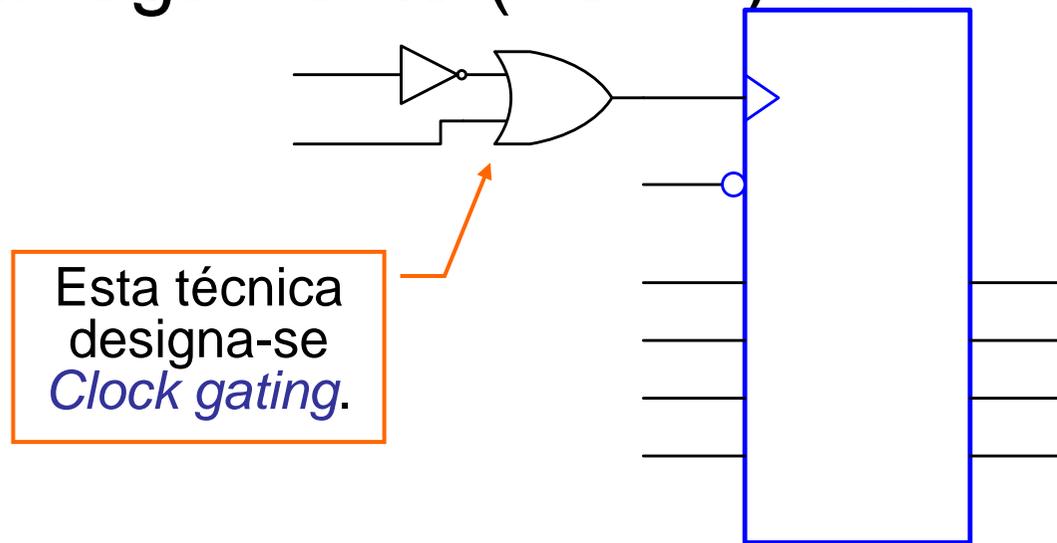
Registos

- Registo de 4 bits com *CLEAR*



Registos

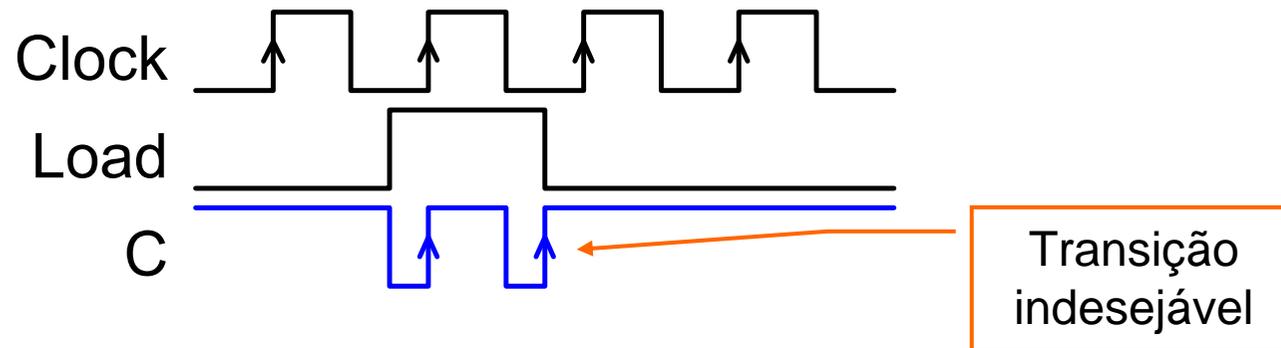
■ Carregamento (LOAD) e CLEAR



Clear	Load	Q_i'	
0	x	0	(Inicializa a '0')
1	0	Q_i	(Mantém o valor guardado)
1	1	D_i	(Carrega novo valor)

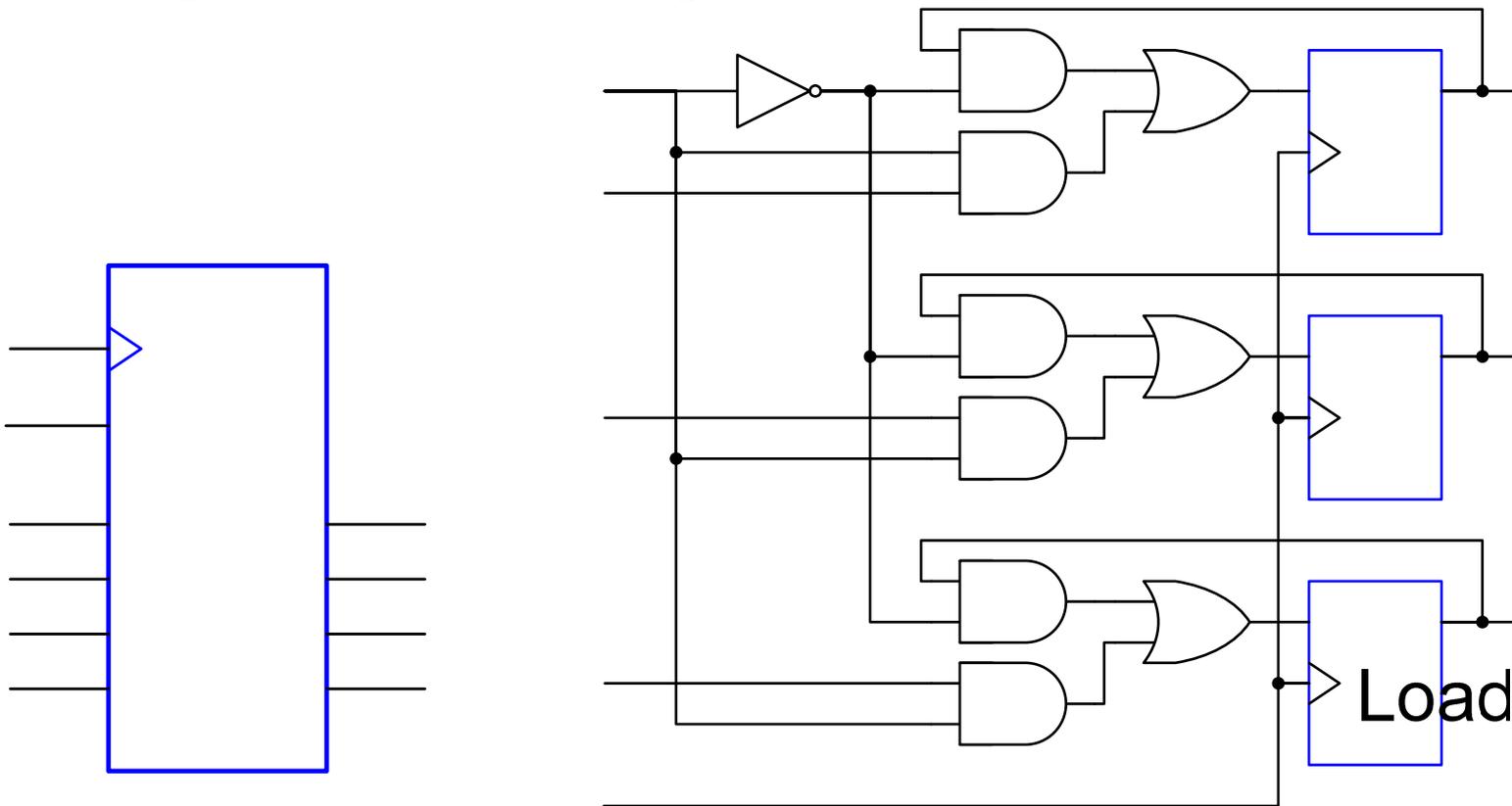
Registos

- A técnica de *clock gating* apresenta algumas desvantagens...
 - Num circuito maior, o registo anterior poderá reagir mais tarde do que outros elementos, o que poderá causar faltas de sincronização – *clock skew* (*escorregamento* do sinal de relógio)
 - Possibilidade de ocorrerem transições “extra” no sinal aplicado à entrada de relógio do registo



Registos

- Registo com carregamento paralelo síncrono

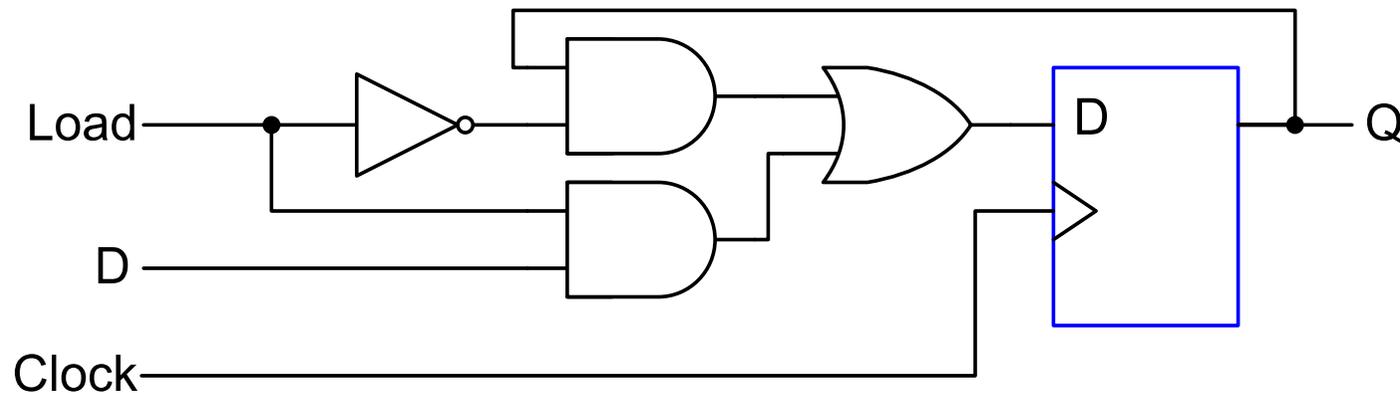


Este esquema é preferível, pois não utiliza *clock gating*.

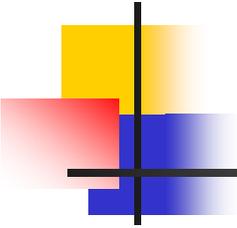
D₀

Registos

- Carregamento paralelo



Load	Q_i'
0	Q_i (Mantém o valor guardado)
1	D_i (Carrega novo valor)



Registos

- Registos de deslocamento

Um **registo de deslocamento** permite **deslocar a informação armazenada** numa dada direcção.

- Deslocamento para a esquerda (*shift left*)

→ na direcção do bit mais significativo

- Exemplo

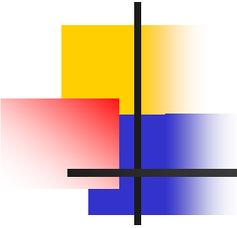
0010 1101 => 0101 101*b* (*b* é um bit que entra)

Deslocamento para a direita (*shift right*)

→ na direcção do bit menos significativo

- Exemplo

0010 1101 => *b*001 0110



Registos

- Formas mais vulgares de deslocamento
 - Lógico (para a esquerda ou para a direita)
 - O bit que entra é sempre '0'
 - Aritmético para a direita
 - O bit que entra é igual ao bit de sinal
 - Rotativo (para a esquerda ou para a direita)
 - O bit que sai por um lado entra pelo outro
- As operações de deslocamento muito utilizadas para:
 - multiplicação e divisão (em binário)
 - conversões série/paralelo e paralelo/série

Registos

- Registo de deslocamento série/paralelo

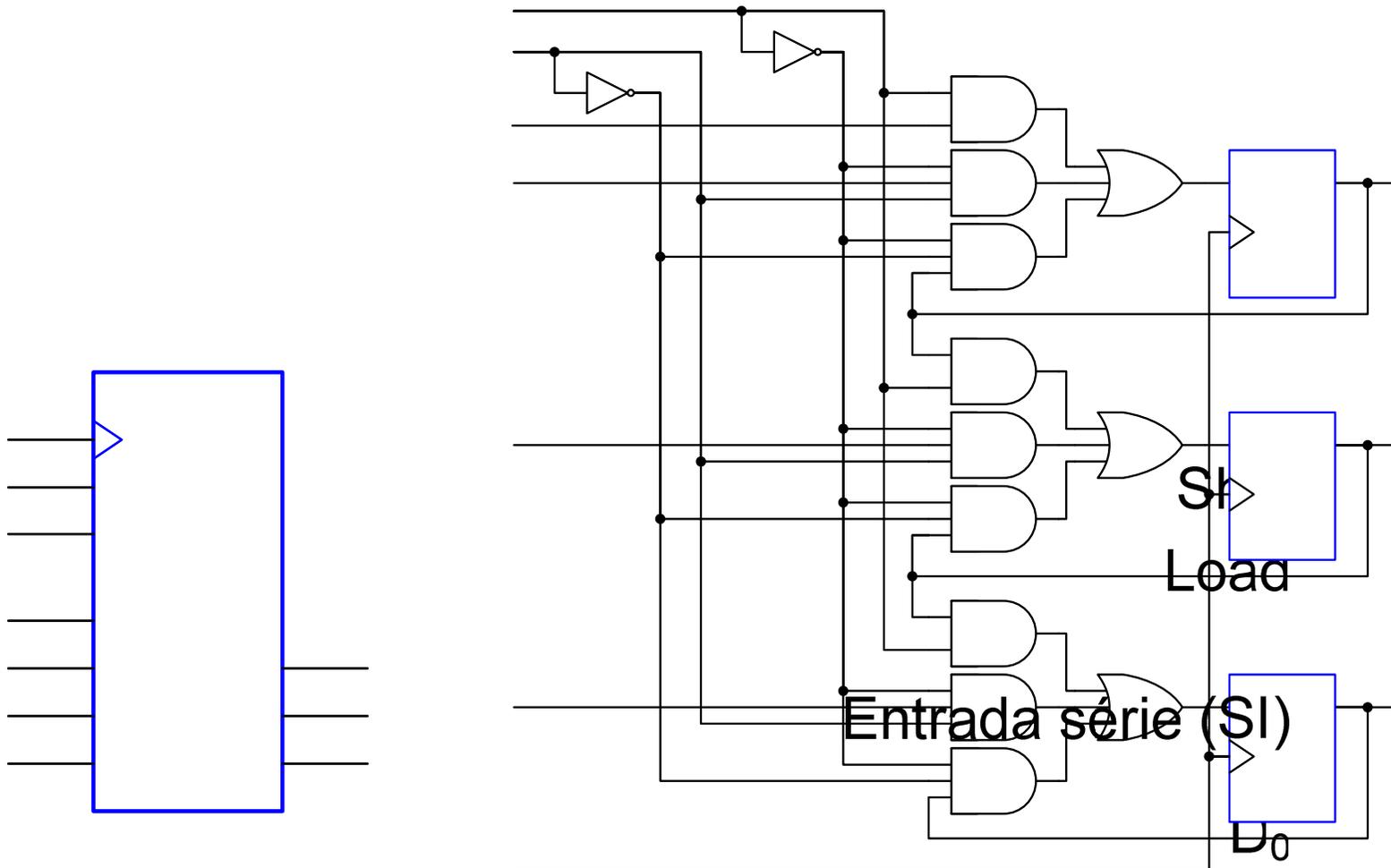


Neste registo, cada impulso do relógio causa um deslocamento para a esquerda.

O objectivo é distribuir os bits que entram em série na entrada SI pelo conjunto de linhas Q₀ a Q₃.

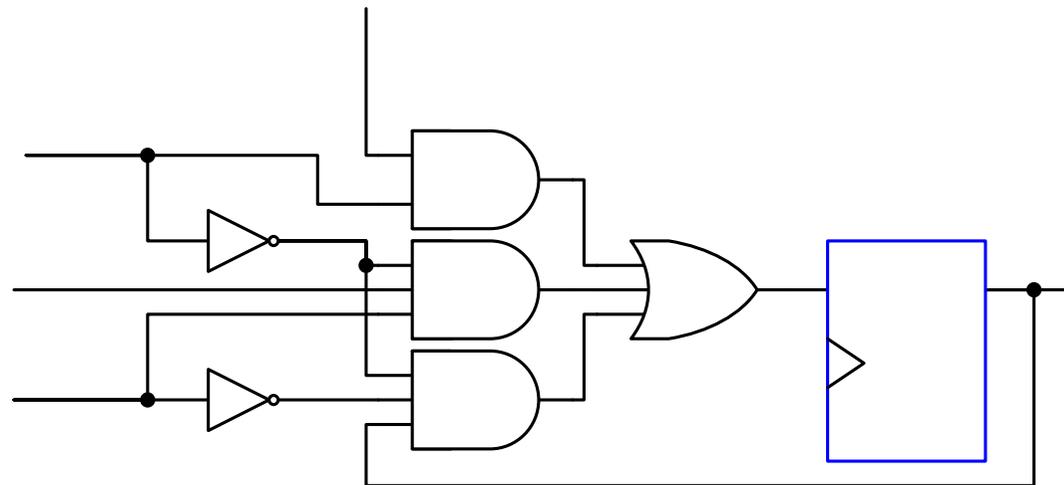
Registos

- Registo de deslocamento com carregamento paralelo



Registos

- Registo de deslocamento com carregamento paralelo



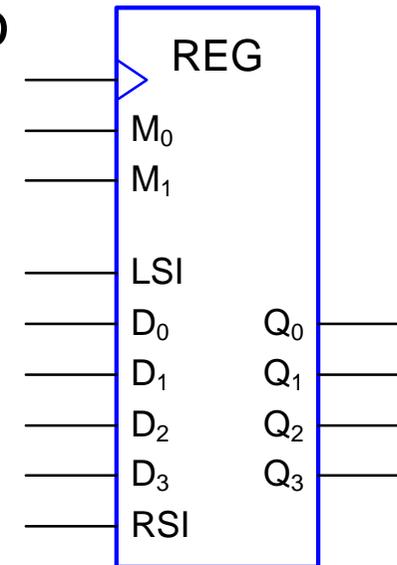
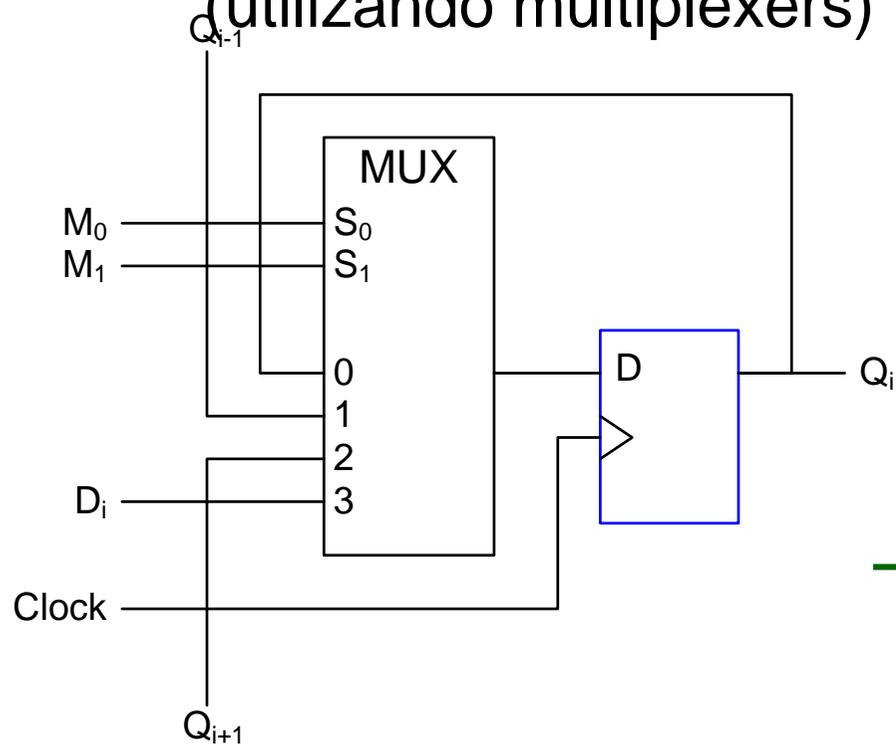
Shift	Load	Q_i'	
0	0	Q_i	(Mantém o valor guardado)
0	1	D_i	(Carrega novo valor)
1	x	Q_{i-1}	(Deslocamento para a esquerda)

Shift

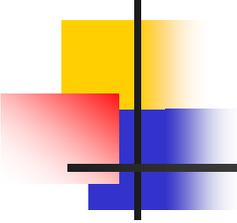
Q_{i-1}

Registos

- Deslocamento e carregamento paralelo (utilizando multiplexers)

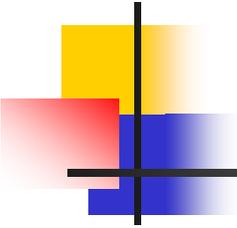


M_1	M_0	Q_i'	
0	0	Q_i	(Mantém)
0	1	Q_{i-1}	(Deslocamento p/ esquerda)
1	0	Q_{i+1}	(Deslocamento p/ direita)
1	1	D_i	(Carrega)



Sumário (Aulas 15 e 16)

- Contadores
 - Contador *ripple* vs. síncrono
 - Contadores com *enable* de contagem
 - Contadores com carregamento paralelo
 - Outros contadores
 - Projecto de circuitos sequenciais utilizando contadores
 - Projecto de contadores



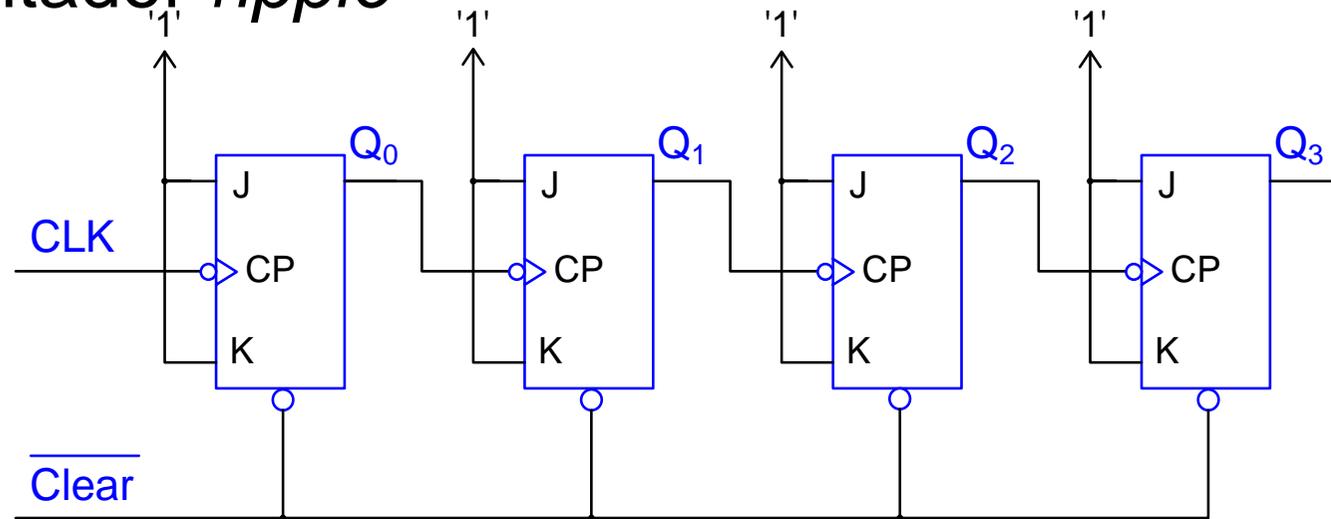
Contadores

- Contador

- Circuito sequencial que segue uma transição de estados pré-programada.
 - Um contador que segue a sequência dos números binários designa-se *Contador binário*
- São geralmente de 2 tipos:
 - *ripple* – a saída de cada *flip-flop* é utilizada como sinal de relógio para accionar outro *flip-flop*
 - *síncrono* – o sinal de relógio é aplicado directamente a todos os *flip-flops*

Contadores

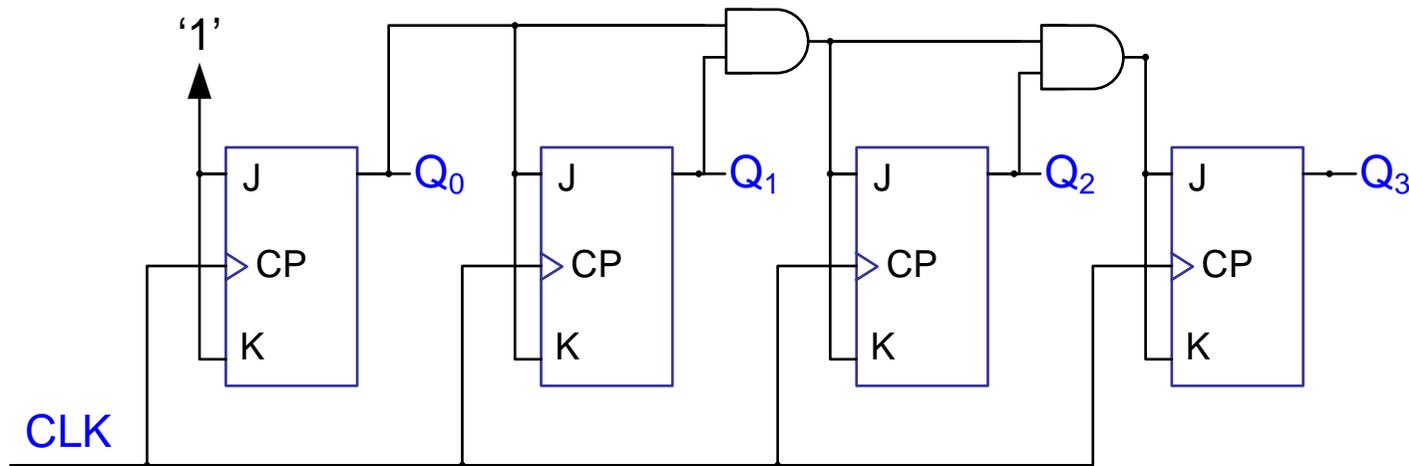
- Contador *ripple*



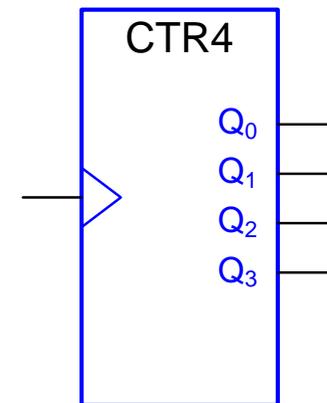
- Por cada 2 impulsos na entrada de relógio (CP) de um *flip-flop*, ocorre um impulso na entrada de relógio do *flip-flop* seguinte
- Os *flip-flops* não reagem todos ao mesmo tempo – existe *escorregamento* do relógio

Contadores

■ Contador síncrono

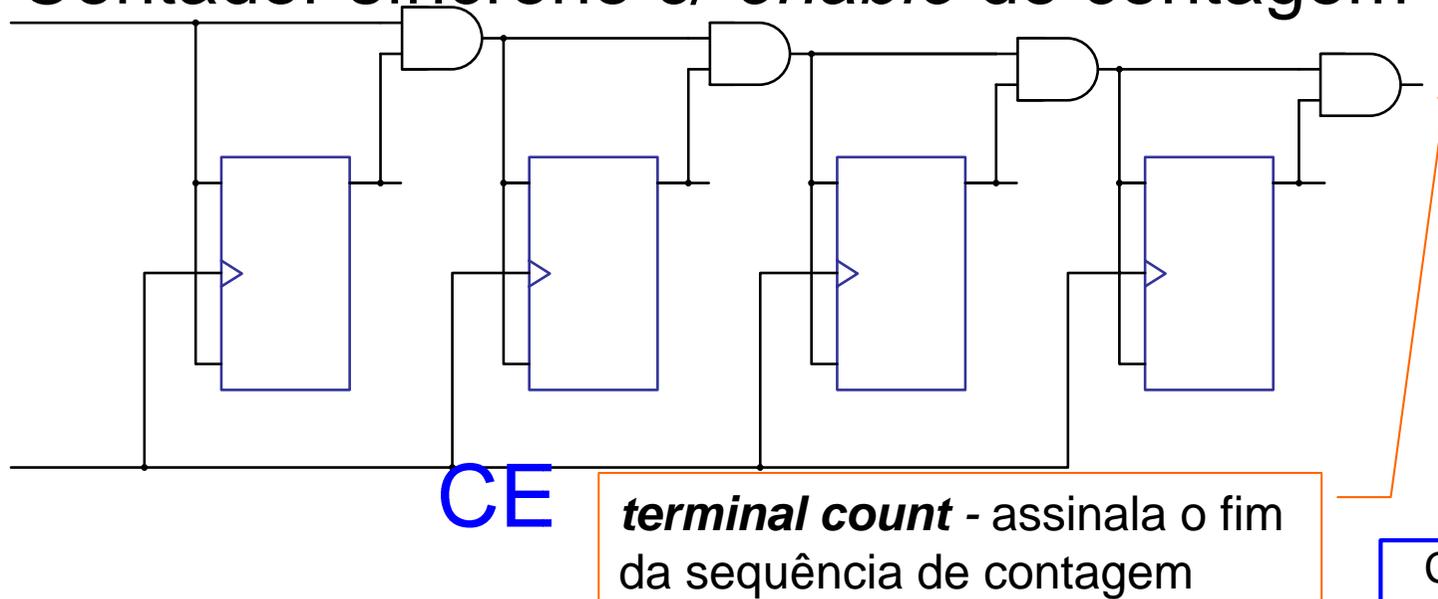


Neste contador todos os flip-flops reagem ao mesmo tempo, o que evita problemas com temporizações

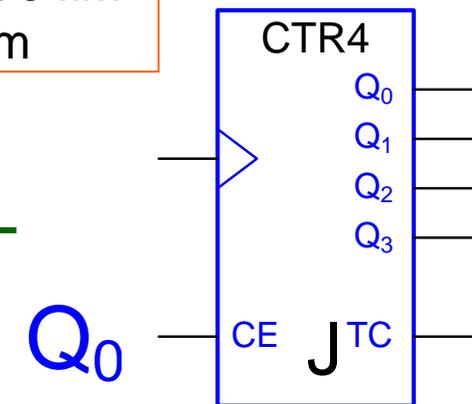


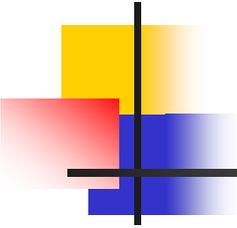
Contadores

- Contador síncrono c/ *enable* de contagem



CE (count enable)	Função
0	Mantém o estado (contagem parada)
1	Conta normalmente



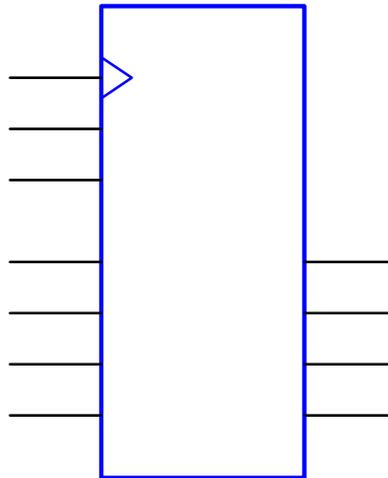


Contadores

- Outros tipos de contadores
 - Contador BCD – sequência de 0 a 9;
 - Contador módulo-N – segue uma sequência de N estados, que podem corresponder a uma sequência binária ou não;
 - Contador *up-down* – permite contagem crescente ou decrescente;
 - Contador com carregamento paralelo – permite o carregamento de um número a partir do qual pode iniciar a contagem.

Contadores

- Contador com carregamento paralelo



Load	CE	Função
0	0	Mantém o estado
0	1	Conta normalmente
1	x	$D \Rightarrow Q$ (Carrega valor)

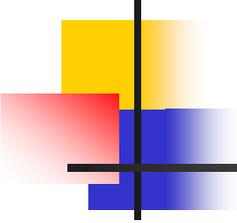
CTR

Load

CE

222

D

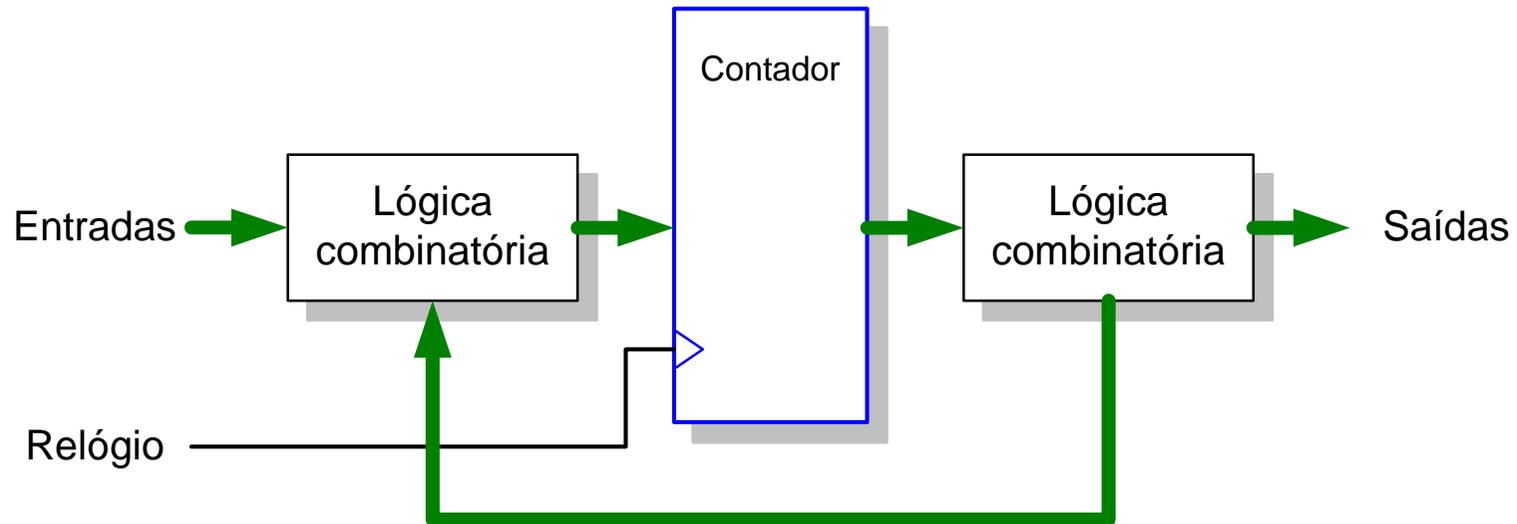


Projecto com contadores

- Existem inúmeras situações em que um circuito sequencial segue sempre (ou quase sempre) a mesma sequência de estados
- Nesses casos é habitual projectar o circuito utilizando um contador como base
- Os estados do circuito são definidos pelas saídas do contador
- Pode-se também tirar partido do carregamento paralelo, *clear*, *count enable*, etc.

Projecto com contadores

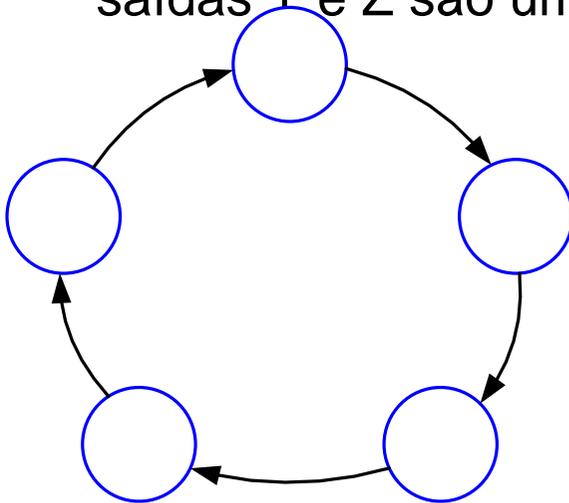
- O circuito poderá seguir a seguinte topologia genérica:



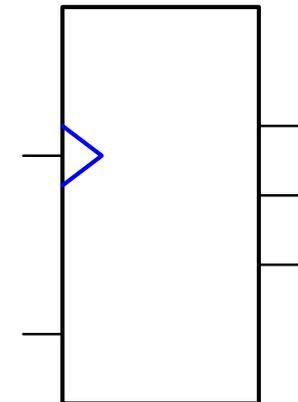
Projecto com contadores

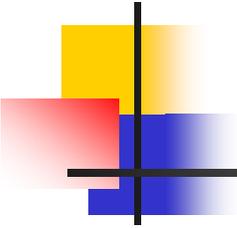
■ Exemplo

- Projectar um circuito sequencial com base num contador de 3 bits com entrada de *clear*
- O circuito deverá seguir o diagrama de estados assinalado. As saídas Y e Z são uma função do estado indicada na tabela



Estado	Saídas	
	Y	Z
0	0	0
1	0	1
2	1	1
3	1	0
4	1	1





Projecto com contadores

- Tabela que descreve o comportamento do circuito

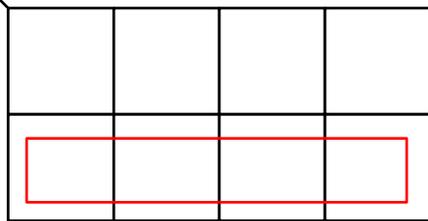
Estado			Saídas		Clear
Q ₂	Q ₁	Q ₀	Y	Z	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	x	x	x

No estado '100' será activada a entrada de *Clear*, para o contador voltar ao estado '000'

Projecto com contadores

- Mapas de Karnaugh

Clear



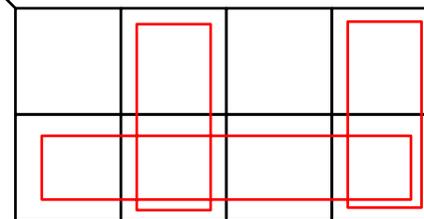
$$Clear = Q_2$$

Z



$$Y = Q_2 + Q_1$$

$Q_1 Q_0$



$$Z = Q_2 + (Q_1 \oplus Q_0)$$

Q_2

00

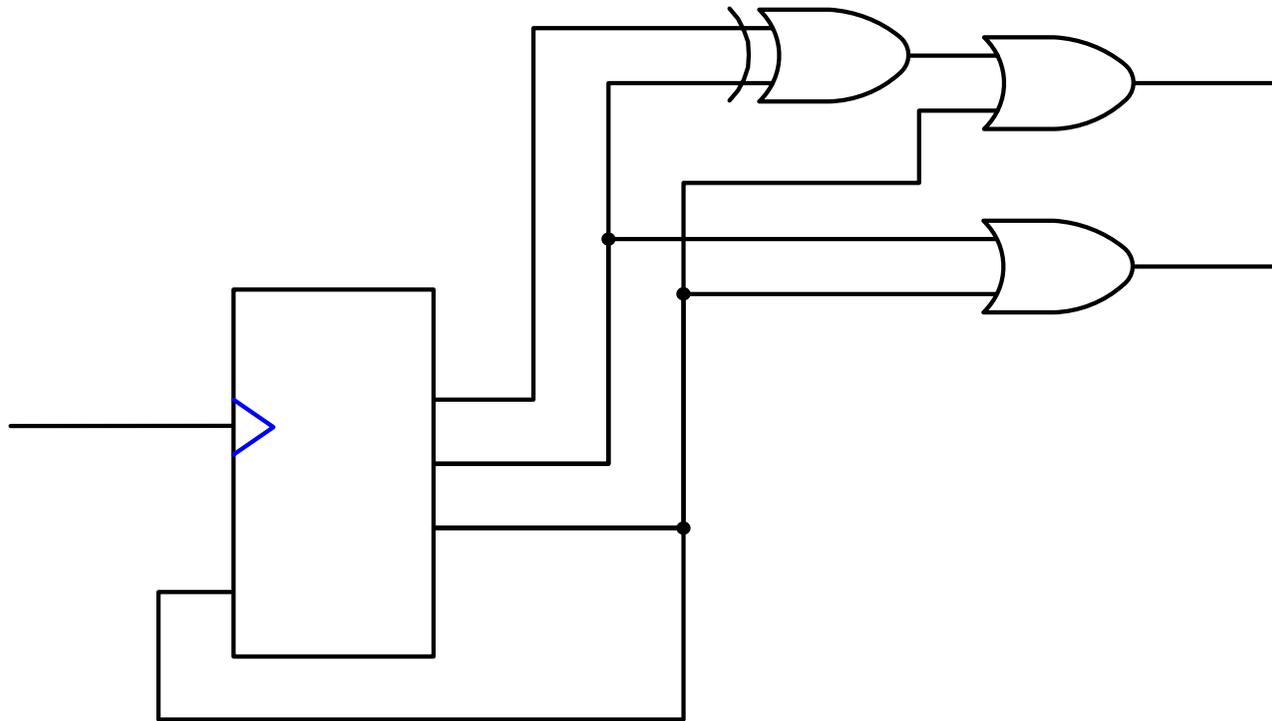
01

11

10

Projecto com contadores

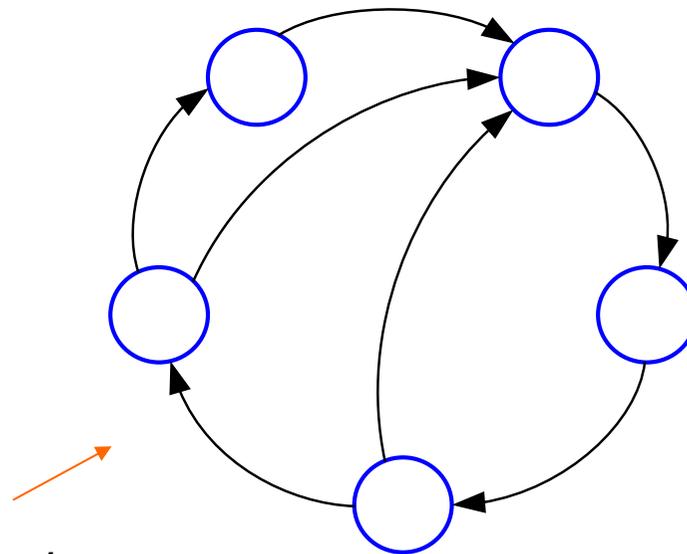
- Circuito resultante



Projecto com contadores

- Outro exemplo:

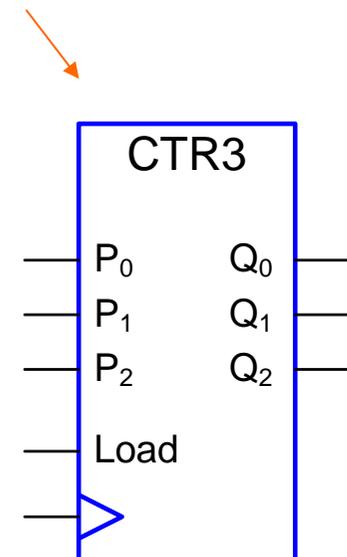
- Implementar um circuito correspondente ao seguinte diagrama de estados, com base num contador binário de 3 bits com carregamento paralelo síncrono



Modelo de *Mealy*

1 entrada (S) e 2 saídas (Y_0 e Y_1)

Contador a utilizar



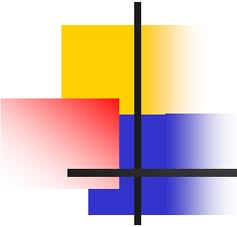
Projecto com contadores

- Tabela de transição de estados

Q_2	Q_1	Q_0	S	Q_2'	Q_1'	Q_0'
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
0	1	1	1	0	0	1
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	x	x	x
1	0	1	1	x	x	x
...

← Contagem normal
 (segue a sequência de contagem)

← Carrega '001'
 ← Carrega '000'
 ← Carrega '001'



Projecto com contadores

- Acrescentando as variáveis Load e P_0 a P_2

Q_2	Q_1	Q_0	S	Q_2'	Q_1'	Q_0'	Load	P_2	P_1	P_0
0	0	0	0	0	0	1	0	x	x	x
0	0	0	1	0	0	1	0	x	x	x
0	0	1	0	0	1	0	0	x	x	x
0	0	1	1	0	1	0	0	x	x	x
0	1	0	0	0	1	1	0	x	x	x
0	1	0	1	0	1	1	0	x	x	x
0	1	1	0	1	0	0	0	x	x	x
0	1	1	1	0	0	1	1	0	0	1
1	0	0	0	0	0	0	1	0	0	0
1	0	0	1	0	0	1	1	0	0	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
...

Projecto com contadores

- Mapas de Karnaugh

$$\text{Load} = Q_2 + Q_1 Q_0 S$$

		$Q_0 S$			
		00	01	11	10
$Q_2 Q_1$	00				
	01			1	
	11	x	x	x	x
	10	1	1	x	x

$$P_0 = S$$

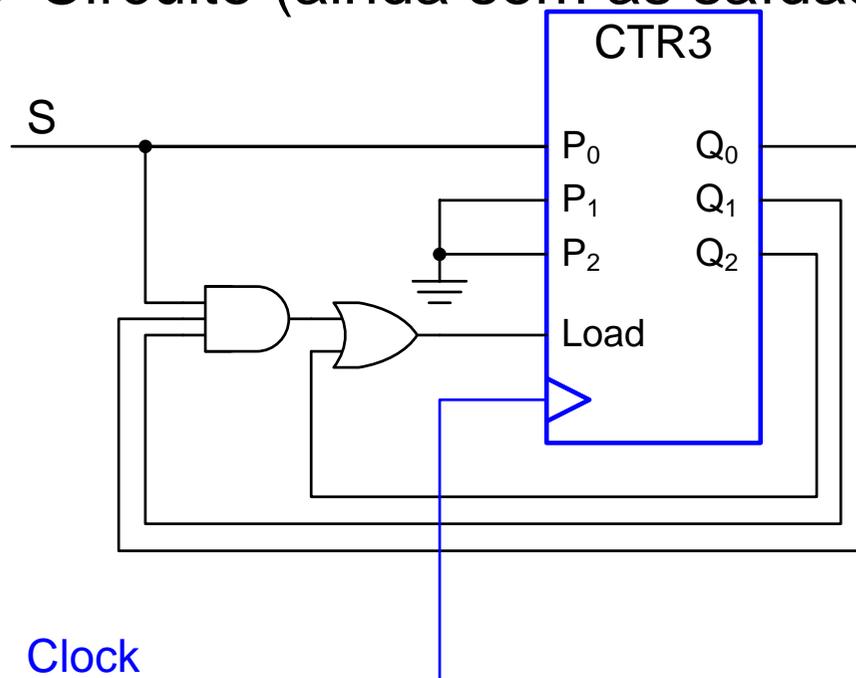
		$Q_0 S$			
		00	01	11	10
$Q_2 Q_1$	00	x	x	x	x
	01	x	x	1	x
	11	x	x	x	x
	10	0	1	x	x

$$P_1 = P_2 = 0$$

(não é necessário mapa)

Projecto com contadores

- Circuito (ainda sem as saídas)



$$Load = Q_2 + Q_1Q_0S$$

$$P_0 = S$$

$$P_1 = P_2 = 0$$

Projecto com contadores

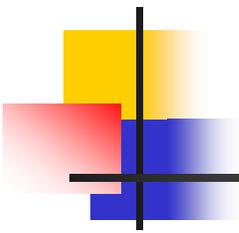
Saídas				Y ₁	Y ₀
Q ₂	Q ₁	Q ₀	S		
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	x	x
1	0	1	1	x	x
...

Q ₂ Q ₁		Q ₀ S			
		00	01	11	10
00				1	
01				1	
11	x	x	x	x	
10	1		x	x	

$$Y_0 = Q_2 \bar{S} + Q_0 \bar{S}$$

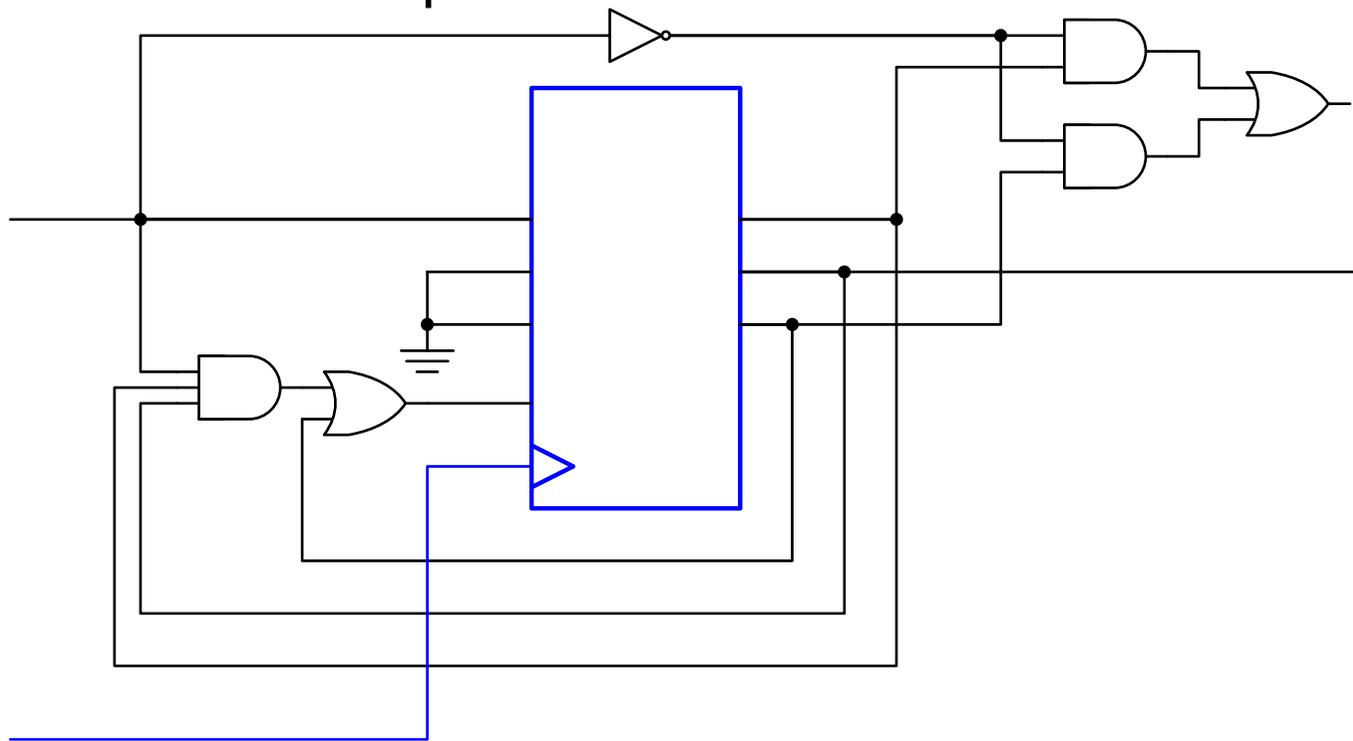
Q ₂ Q ₁		Q ₀ S			
		00	01	11	10
00					
01	1	1	1	1	
11	x	x	x	x	
10			x	x	

$$Y_1 = Q_1$$



Projecto com contadores

- Circuito completo



S

CTR3

P₀
236

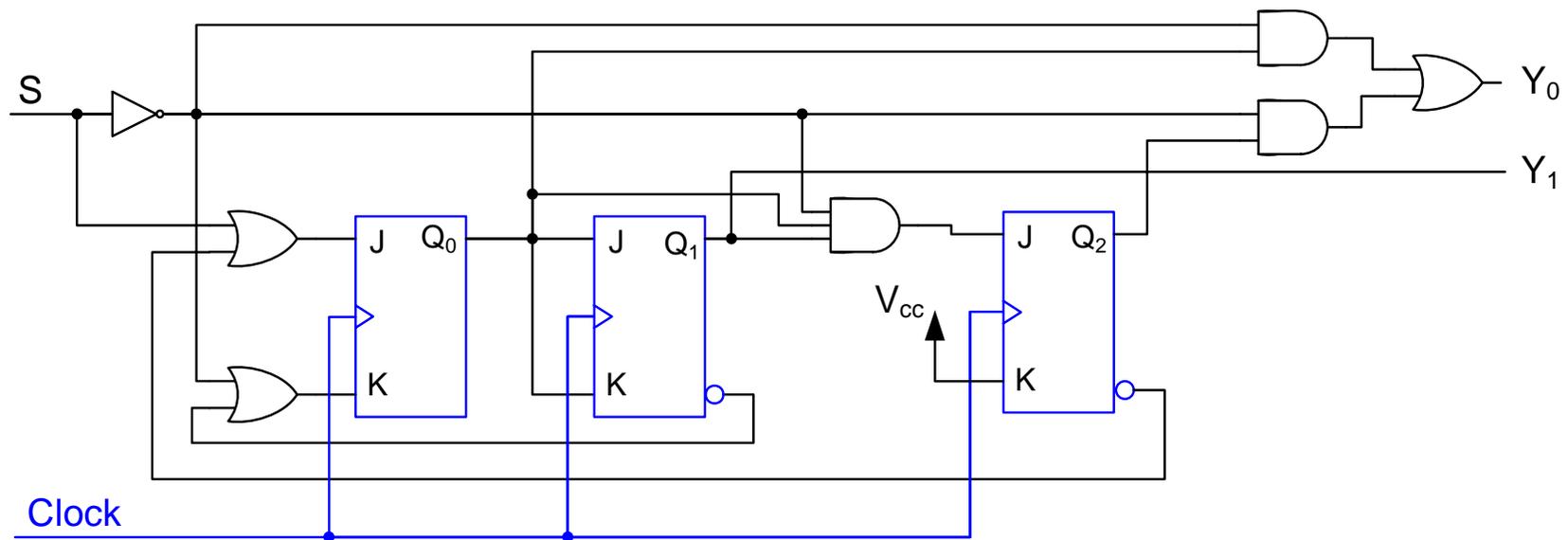
Q

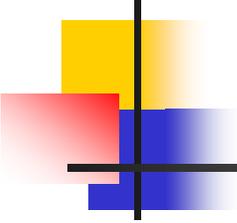
P₁

Q

Projecto com contadores

- Só por curiosidade... Com FFs JK e seguindo o método convencional, o circuito resultante seria:





Projecto de contadores

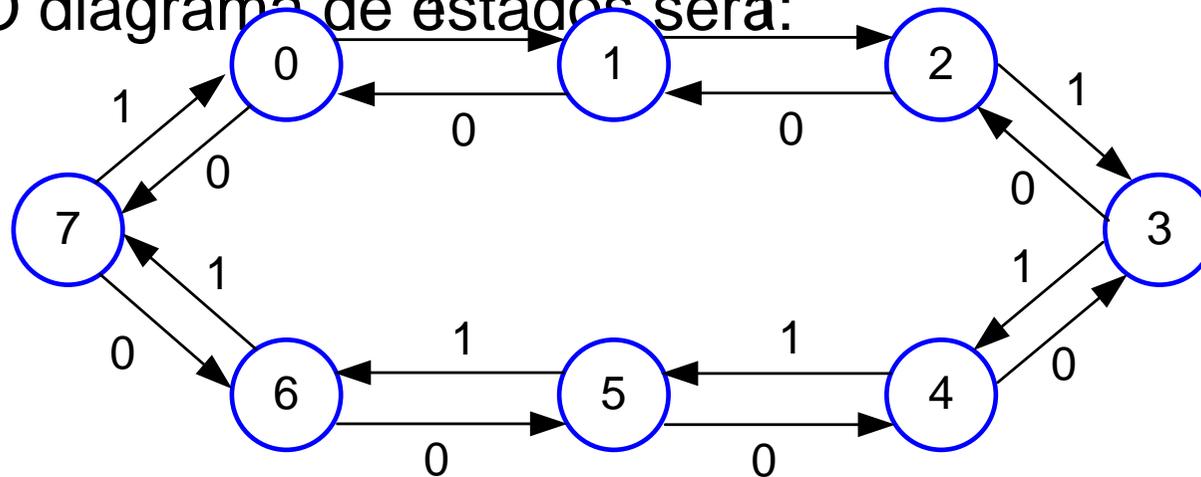
- Projecto de contadores
 - Diagrama de estados da contagem
 - Tabela de transição de estados
 - Equações de entrada nos FFs

- Depois podem ser acrescentadas outras funcionalidades, tais como
 - *Enable* de contagem
 - *Clear* síncrono
 - Carregamento paralelo síncrono
 - Detecção de final da sequência de contagem

Projecto de contadores

- Exemplo:

- Projectar um contador de 3 bits com controlo de direcção de contagem (ascendente/descendente)
- O diagrama de estados será:



Entrada: U = 0 – contagem descendente
U = 1 – contagem ascendente

Projecto de contadores

■ Tabela de transição de estados

Q_2	Q_1	Q_0	U	Q_2'	Q_1'	Q_0'	J_2K_2	J_1K_1	J_0K_0
0	0	0	0	1	1	1	1X	1X	1X
0	0	0	1	0	0	1	0X	0X	1X
0	0	1	0	0	0	0	0X	0X	X1
0	0	1	1	0	1	0	0X	1X	X1
0	1	0	0	0	0	1	0X	X1	1X
0	1	0	1	0	1	1	0X	X0	1X
0	1	1	0	0	1	0	0X	X0	X1
0	1	1	1	1	0	0	1X	X1	X1
1	0	0	0	0	1	1	X1	1X	1X
1	0	0	1	1	0	1	X0	0X	1X
1	0	1	0	1	0	0	X0	0X	X1
1	0	1	1	1	1	0	X0	1X	X1
1	1	0	0	1	0	1	X0	X1	1X
1	1	0	1	1	1	1	X0	X0	1X
1	1	1	0	1	1	0	X0	X0	X1
1	1	1	1	0	0	0	X1	1X	X1

Projecto de contadores

$$J_2 = K_2 = \overline{Q_1} \overline{Q_0} \overline{U} + Q_1 Q_0 U = \overline{Q_1} \oplus U \cdot \overline{Q_0} \oplus U$$

■ Mapas de Karnaugh

	Q ₀ U			
Q ₂ Q ₁	00	01	11	10
00	1			
01			1	
11	X	X	X	X
10	X	X	X	X

	Q ₀ U			
Q ₂ Q ₁	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11			1	
10	1			

$$J_1 = K_1 = \overline{Q_0} \overline{U} + Q_0 U = \overline{Q_0} \oplus U$$

$$J_0 = K_0 = 1$$

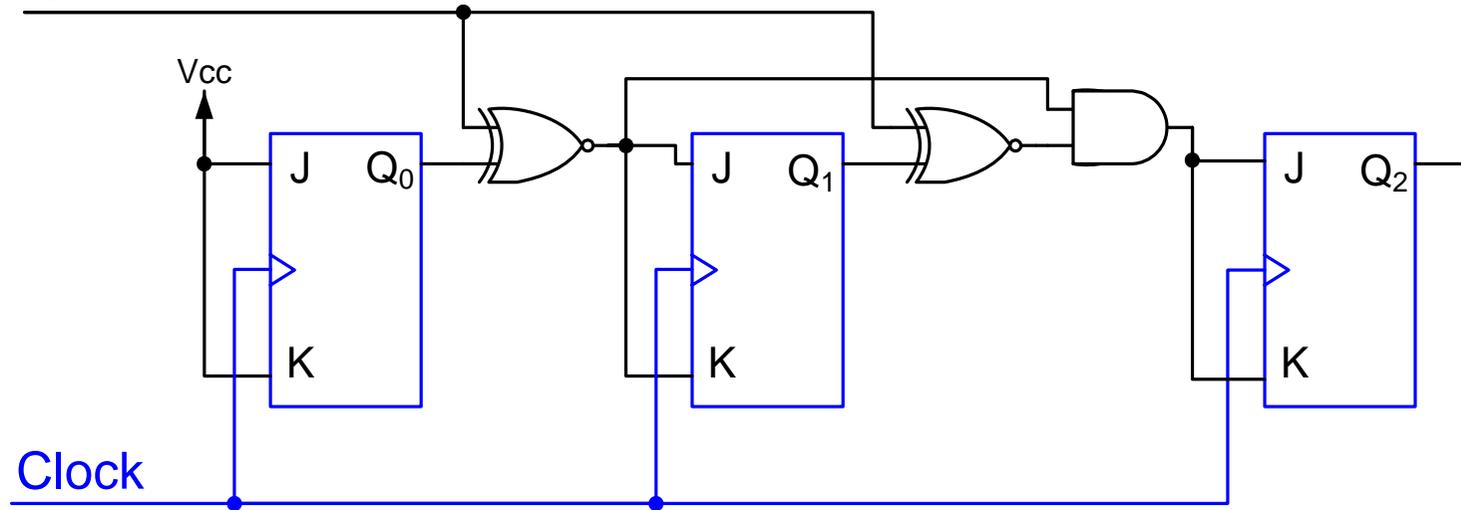
(não são necessários mapas)

	Q ₀ U			
Q ₂ Q ₁	00	01	11	10
00	1		1	
01	X	X	X	X
11	X	X	X	X
10	1		1	

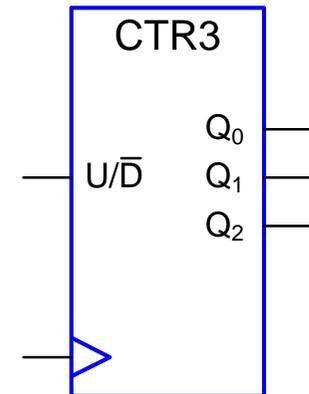
	Q ₀ U			
Q ₂ Q ₁	00	01	11	10
00	X	X	X	X
01	1		1	
11	1		1	
10	X	X	X	X

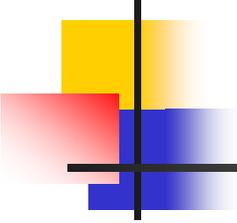
Proyecto de contadores

U ■ Circuito



Símbolo →





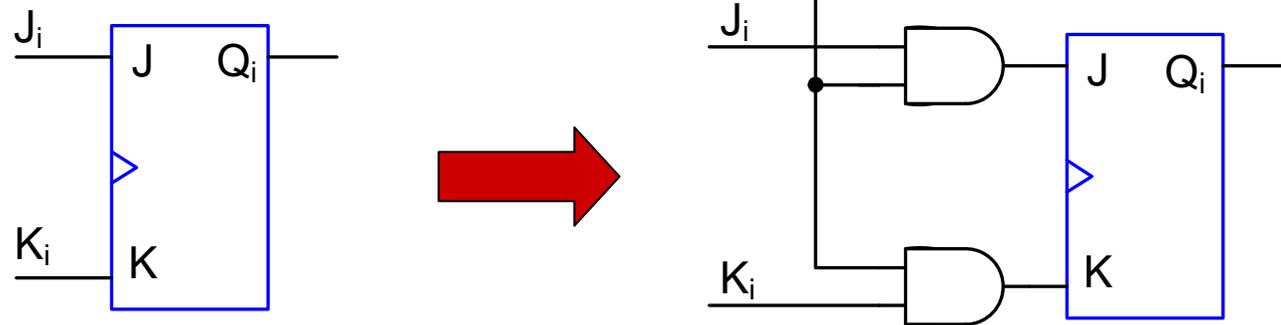
Projecto de contadores

- Outras funcionalidades de um contador:
 - *Enable* de contagem
 - *Clear* (ou Reset) síncrono
 - Carregamento paralelo síncrono

- Podem ser implementadas depois de projectado o contador base, i.e., o conjunto de FFs e lógica que implementam a sequência de contagem desejada
- Com se verá o raciocínio seguido para implementar as funcionalidades referidas é semelhante

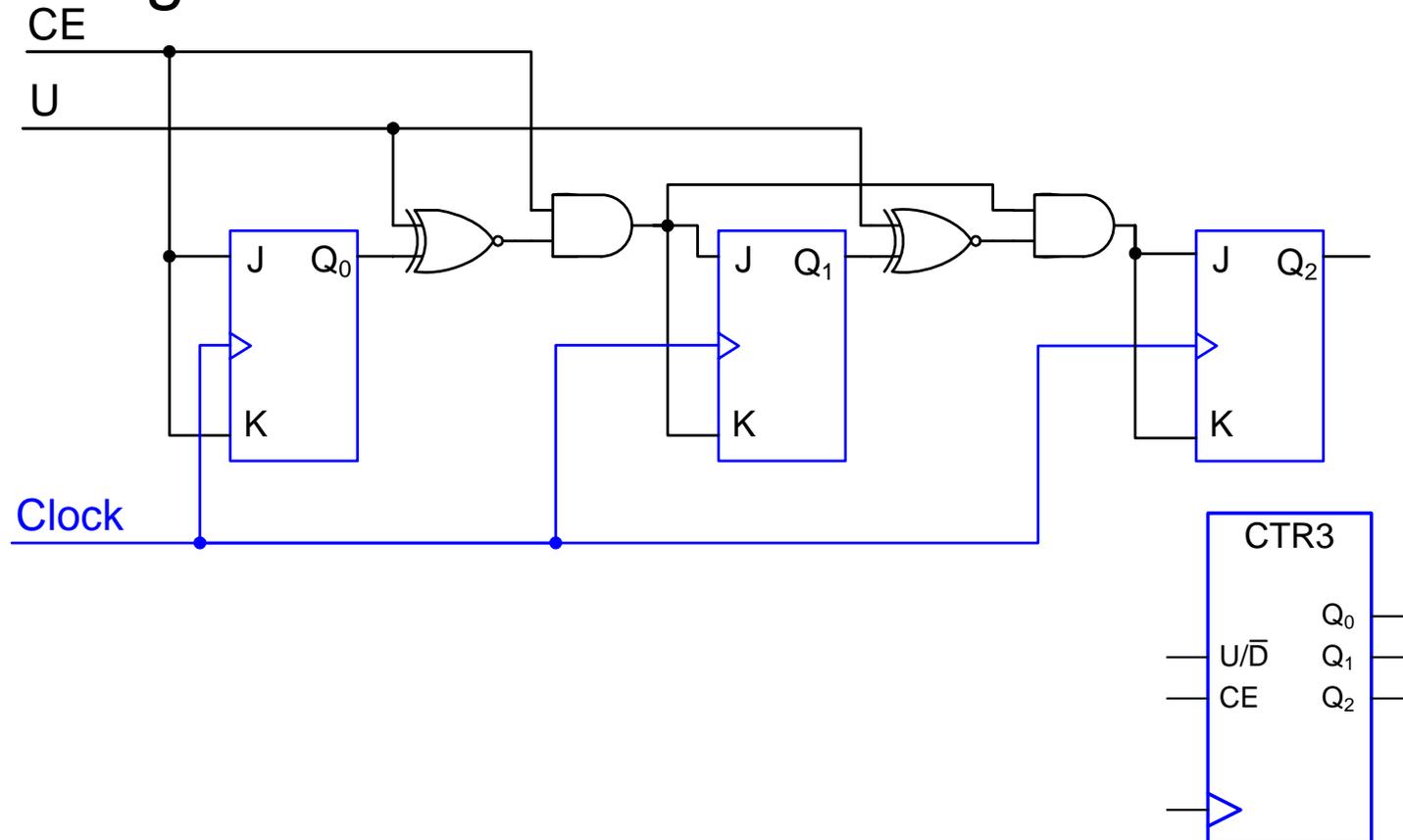
Projecto de contadores

- *Enable* de contagem (CE)
 - Acrescenta-se material às entradas de cada flip-flop de modo a que:
 - com CE desactivado – o FF nunca muda de estado
 - com CE activado – o circuito comporta-se normalmente



Projecto de contadores

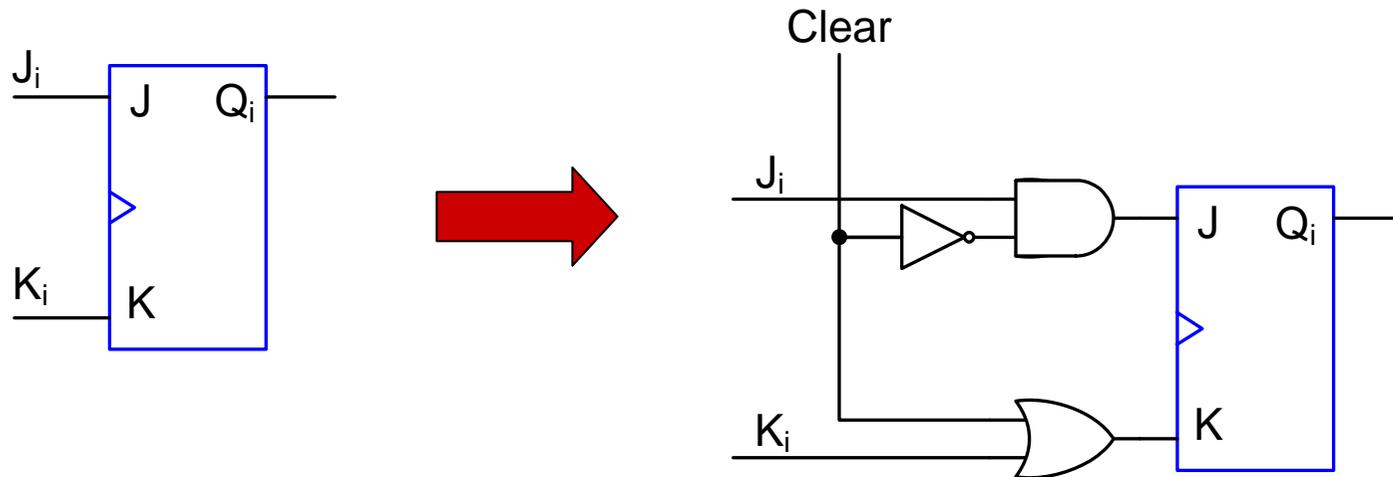
- Exemplo – acrescentar um sinal de Enable de contagem ao contador anterior



Projecto de contadores

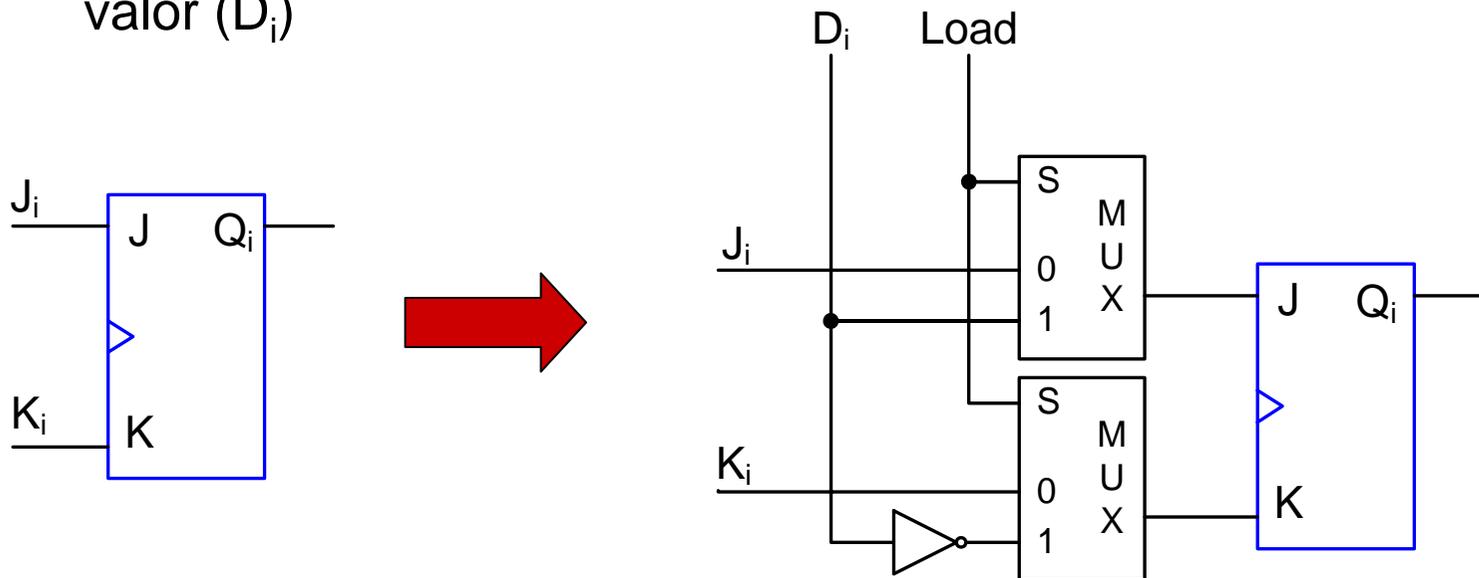
- *Clear* síncrono

- Acrescenta-se material às entradas de cada flip-flop de modo a que:
 - com *Clear* desactivado – o circuito comporta-se normalmente
 - com *Clear* activado – “obriga-se” o FF a fazer *Reset* ($J=0$, $K=1$)



Projecto de contadores

- Carregamento paralelo síncrono (Load)
 - Acrescenta-se material às entradas de cada flip-flop de modo a que:
 - com *Load* desactivado – o circuito comporta-se normalmente
 - com *Load* activado – “obriga-se” o FF a carregar um novo valor (D_i)

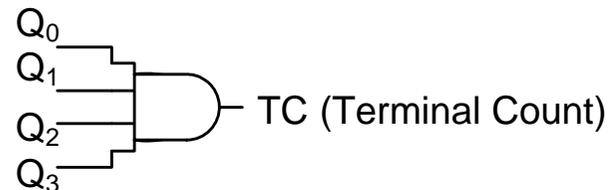


Projecto de contadores

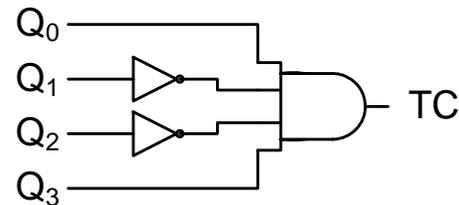
- Detectar o final do ciclo de contagem
 - Acrescenta-se material às saídas do contador
 - um circuito cuja saída vai a '1' no último estado da sequência

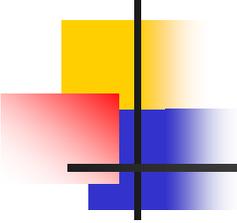
- Exemplos

- Contador binário de 4 bits
o fim é em 15 (1111)



- Contador BCD
o fim é em 9 (1001)





Sumário (Aula 17)

- Transferências entre registos
 - Notação utilizada
 - Noção de barramento (BUS)
 - Variáveis de controlo
 - Selecção de fontes e de destino
 - *Buffers tri-state* e registos bi-direccionais



Sistemas de Computação

Paulo Santos

Transferências entre registos



UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

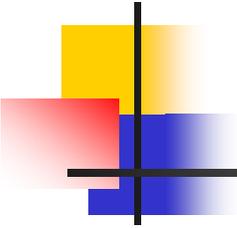
IQF
Instituto para a Qualidade
na Formação, I.P.

Transferências entre registos

■ Notação

Símbolo	Significado	Símbolo	Significado
←	Carregamento	⊕	Xor (bit-a-bit)
+	Adição	<< ou sl	Deslocamento lógico à esquerda
-	Subtracção	>> ou sr	Deslocamento lógico à direita
∧	And (bit-a-bit)	rl	Rotação à esquerda
∨	Or (bit-a-bit)	rr	Rotação à direita
~	Not (bit-a-bit)	asr	Deslocamento aritmético à direita

Designação	Significado	Exemplos
Letras (e números)	Registos	R0; RC; ACC; IR
Parêntesis	Partes de um registo	R1(0); R2(0...3), AC(8..15)
Vírgula	Operações em simultâneo	R1 ←R0, R2 ←R3+R1



Transferências entre registos

- Notação (exemplos)

Notação	Significado
$R0 \leftarrow R1 \vee R2$	Carregar em R0 o resultado de um Or (bit a bit) entre R1 e R2
$R0 \leftarrow R1 + R2$	Carregar em R0 a soma de R1 com R2
$R0 \leftarrow sl(R1)$	Carregar em R0 o resultado do deslocamento lógico para a esquerda de R1
$R1 \leftarrow R1 - R2$	Subtrair R2 a R1 e guardar o resultado de volta em R1
$R0 \leftarrow R0 + 1, R1 \leftarrow R2$	Incrementar R0 e carregar R1 com o conteúdo de R2

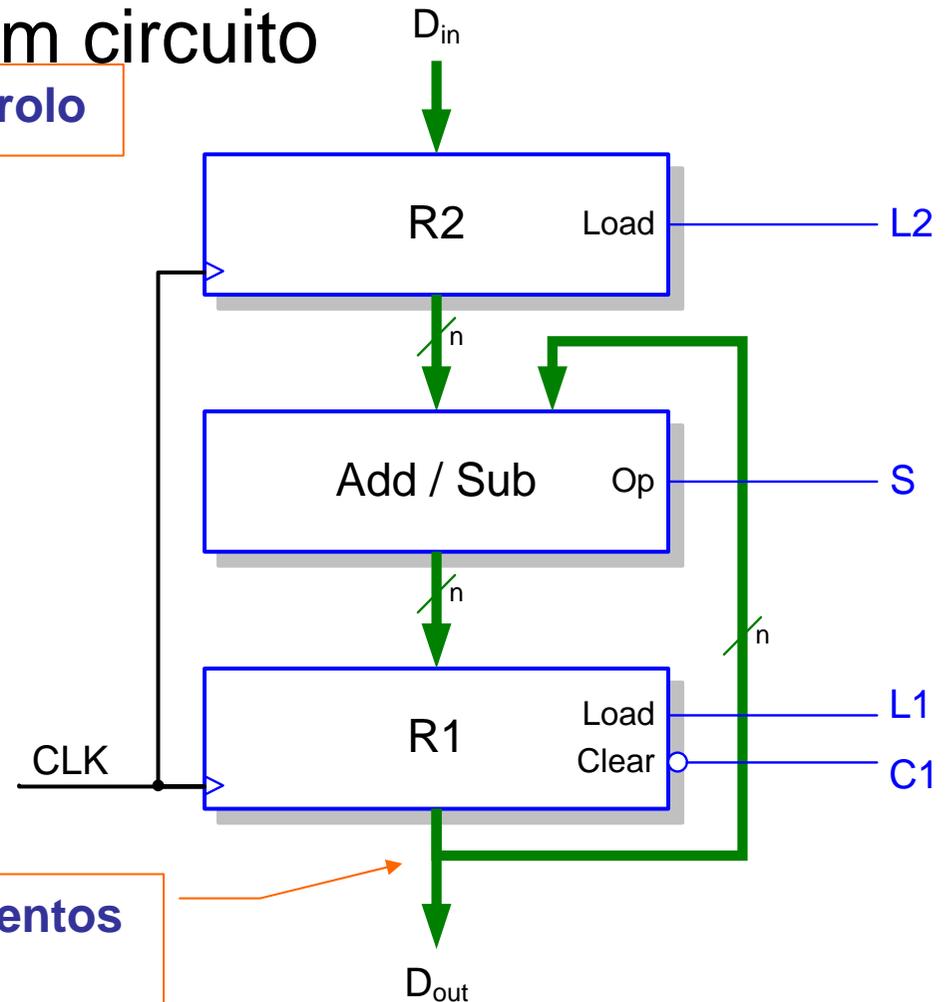
Transferências entre registos

■ Notação aplicada a um circuito

Variáveis de controlo

C1	L1	L2	S	Função
0	x	0	x	$R1 \leftarrow 0$
1	0	0	x	---
1	0	1	x	$R2 \leftarrow D_{in}$
1	1	0	0	$R1 \leftarrow R1 + R2$
1	1	0	1	$R1 \leftarrow R1 - R2$
1	1	1	0	$R1 \leftarrow R1 + R2,$ $R2 \leftarrow D_{in}$
1	1	1	1	$R1 \leftarrow R1 - R2,$ $R2 \leftarrow D_{in}$

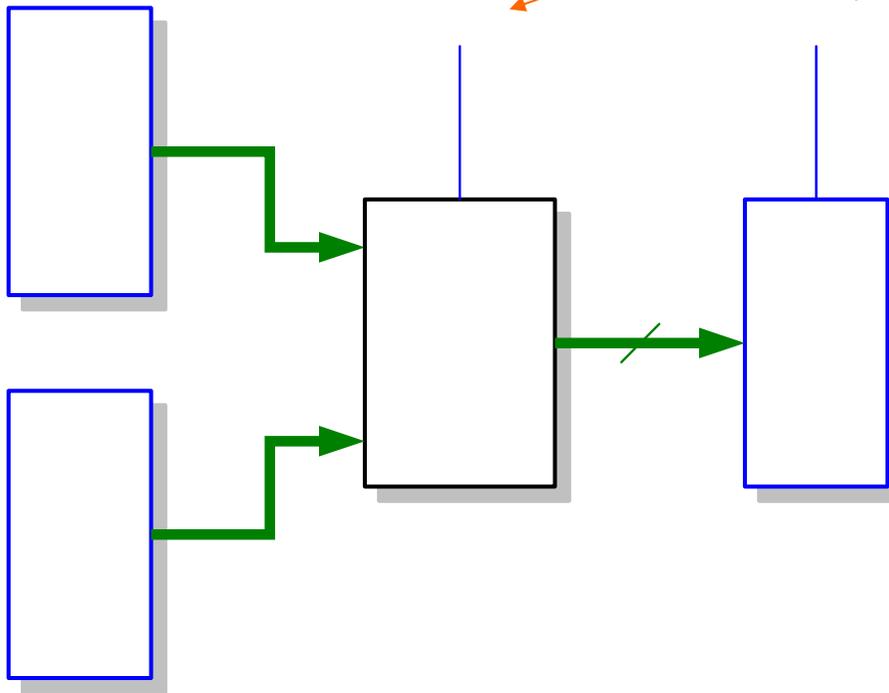
Linhas grossas são **barramentos** (BUS) - várias linhas de bit



Transferências entre registos

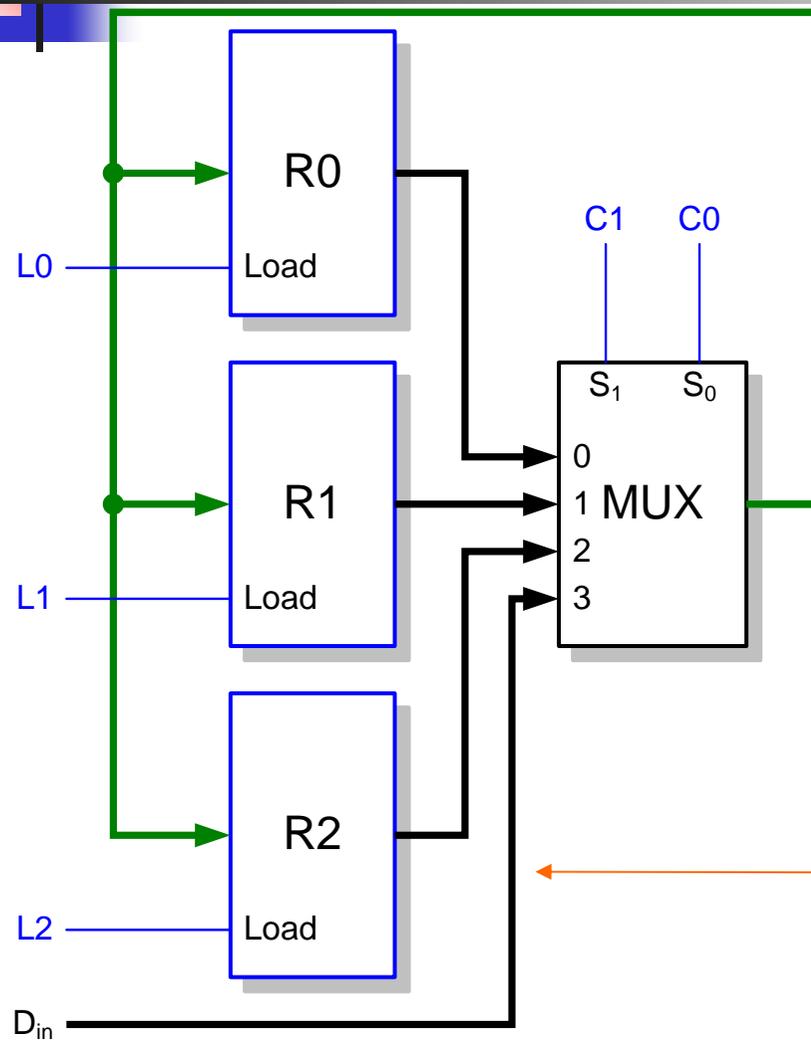
- Utilização de multiplexers
 - Selecção da origem de dados

Variáveis de controlo



C2	C1	Função
0	x	---
1	0	$R0 \leftarrow R1$
1	1	$R0 \leftarrow R2$

Transferências entre registros



Exemplos de operações

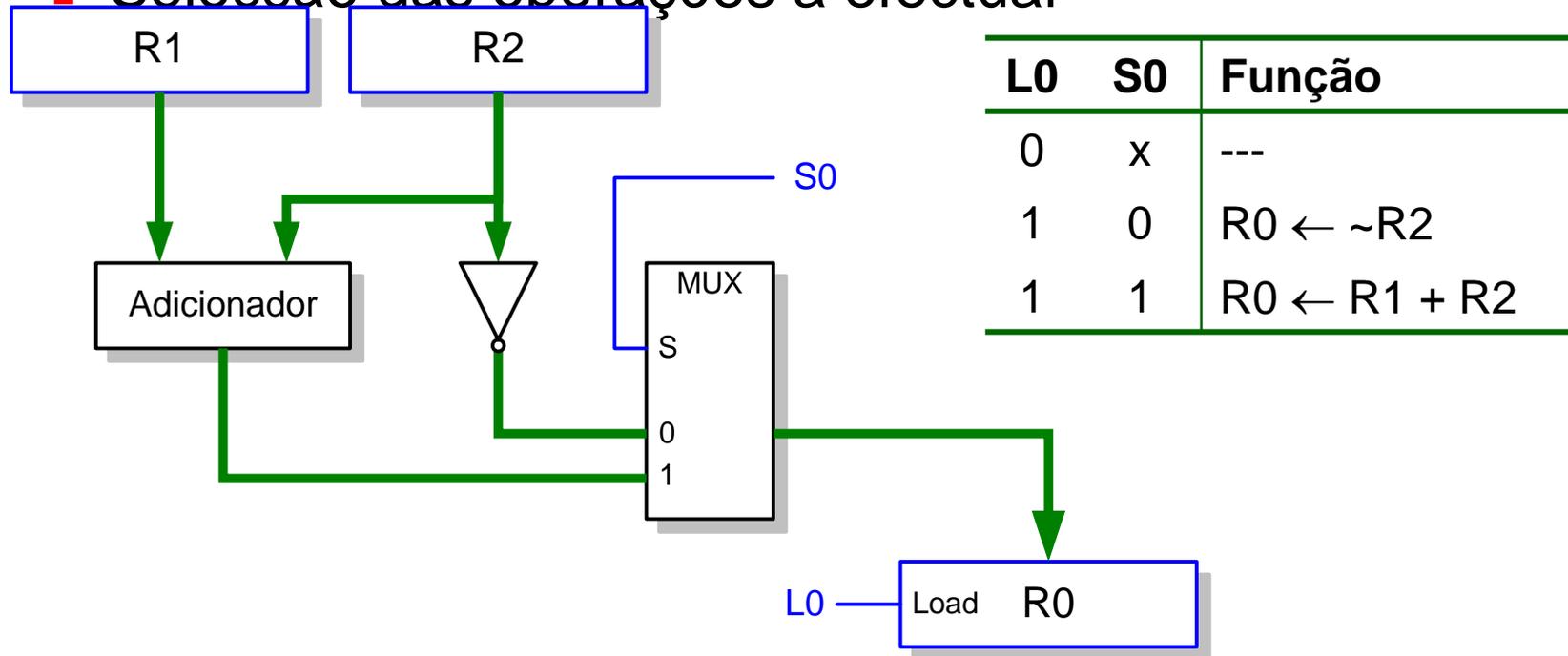
Transferência	C1	C0	L2	L1	L0
$R0 \leftarrow R1$	0	1	0	0	1
$R0 \leftarrow R1, R2 \leftarrow R1$	0	1	1	0	1
$R1 \leftarrow D_{in}$	1	1	0	1	0
$R0 \leftarrow R1, R2 \leftarrow R0$	Impossível				

BUS de dados
externos

Transferências entre registos

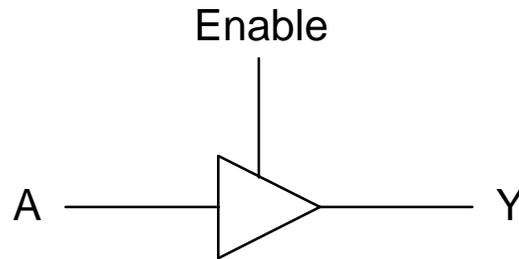
- Utilização de multiplexers

- Seleccão das operações a efectuar



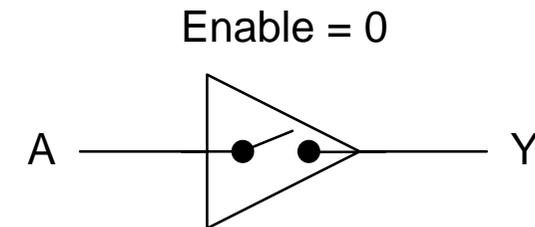
Transferências entre registos

- Utilização de *buffers tri-state*
 - Funcionamento de um *buffer tri-state*

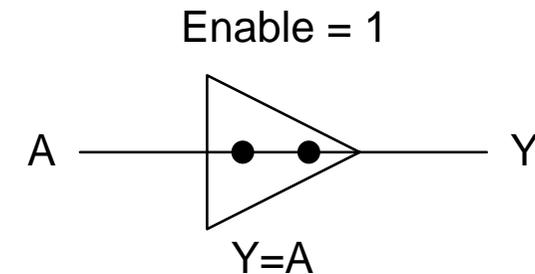


Enable	A	Y
0	x	? (alta impedância)
1	0	0
1	1	1

Modelos:



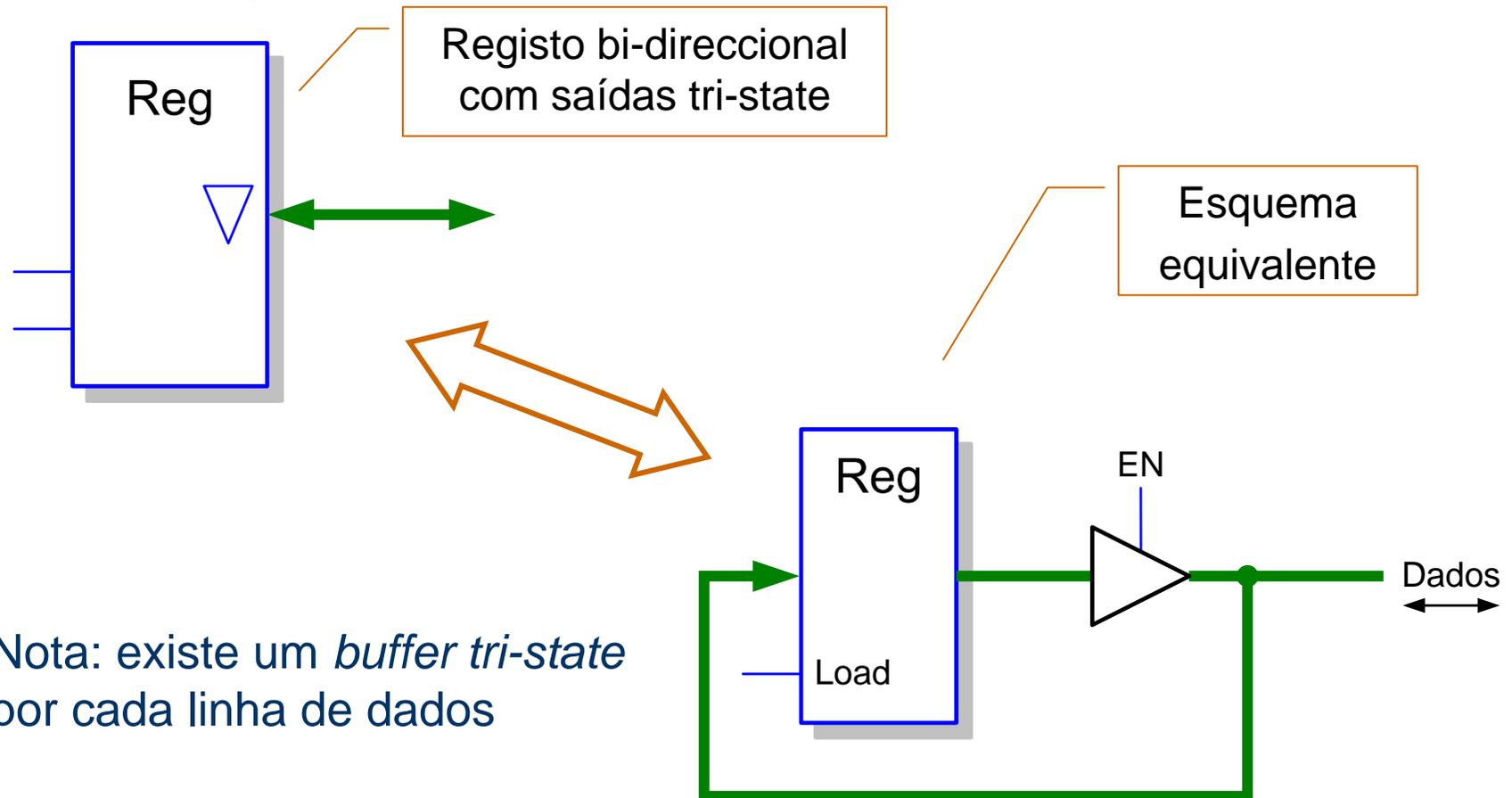
Y fica isolado de A
(alta impedância)



Y=A

Transferências entre registos

- Utilização de *buffers tri-state*



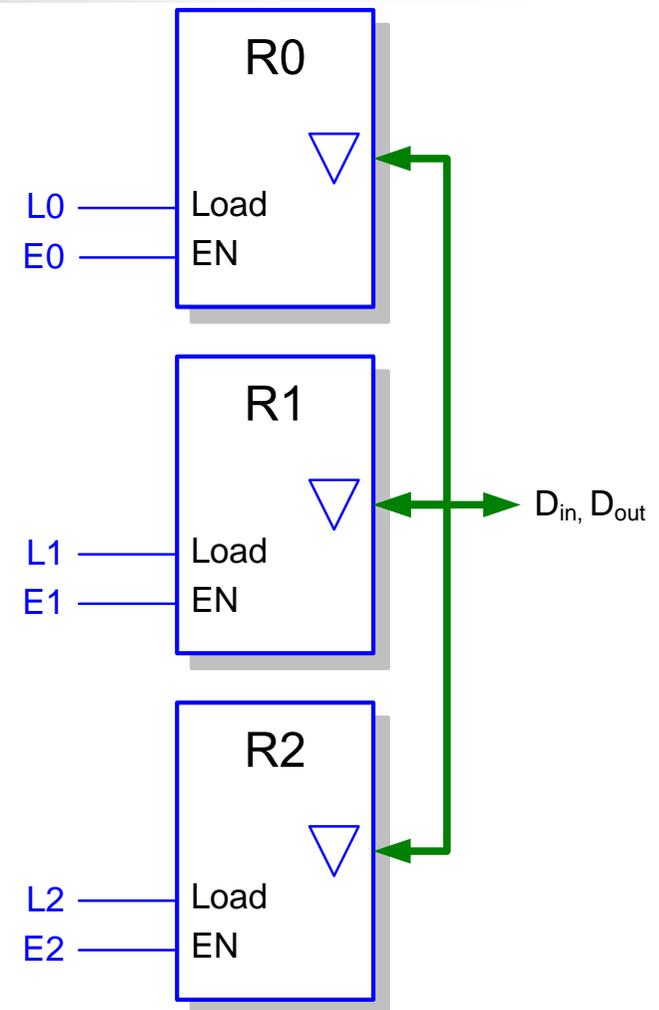
Nota: existe um *buffer tri-state* por cada linha de dados

Transferências entre registos

- Utilização de *buffers tri-state*
 - EN – Controlo da saída
 - Load – Controlo da entrada

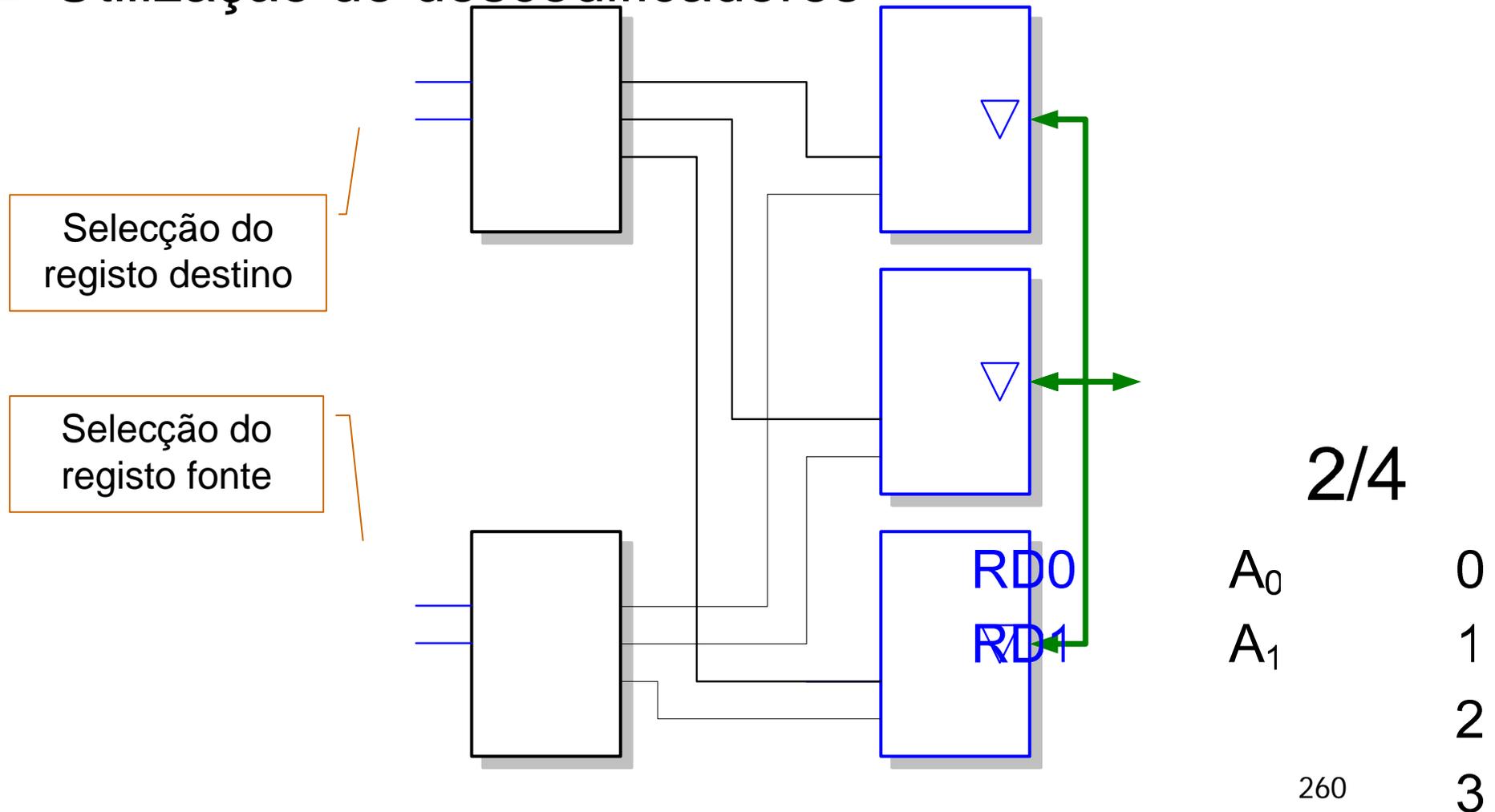
Exemplos

Transferência	E2	E1	E0	L2	L1	L0
R0←R1	0	1	0	0	0	1
R0←R1, R2←R1	0	1	0	1	0	1
R1←R2	1	0	0	0	1	0
R0←R1, R2←R0	Impossível					

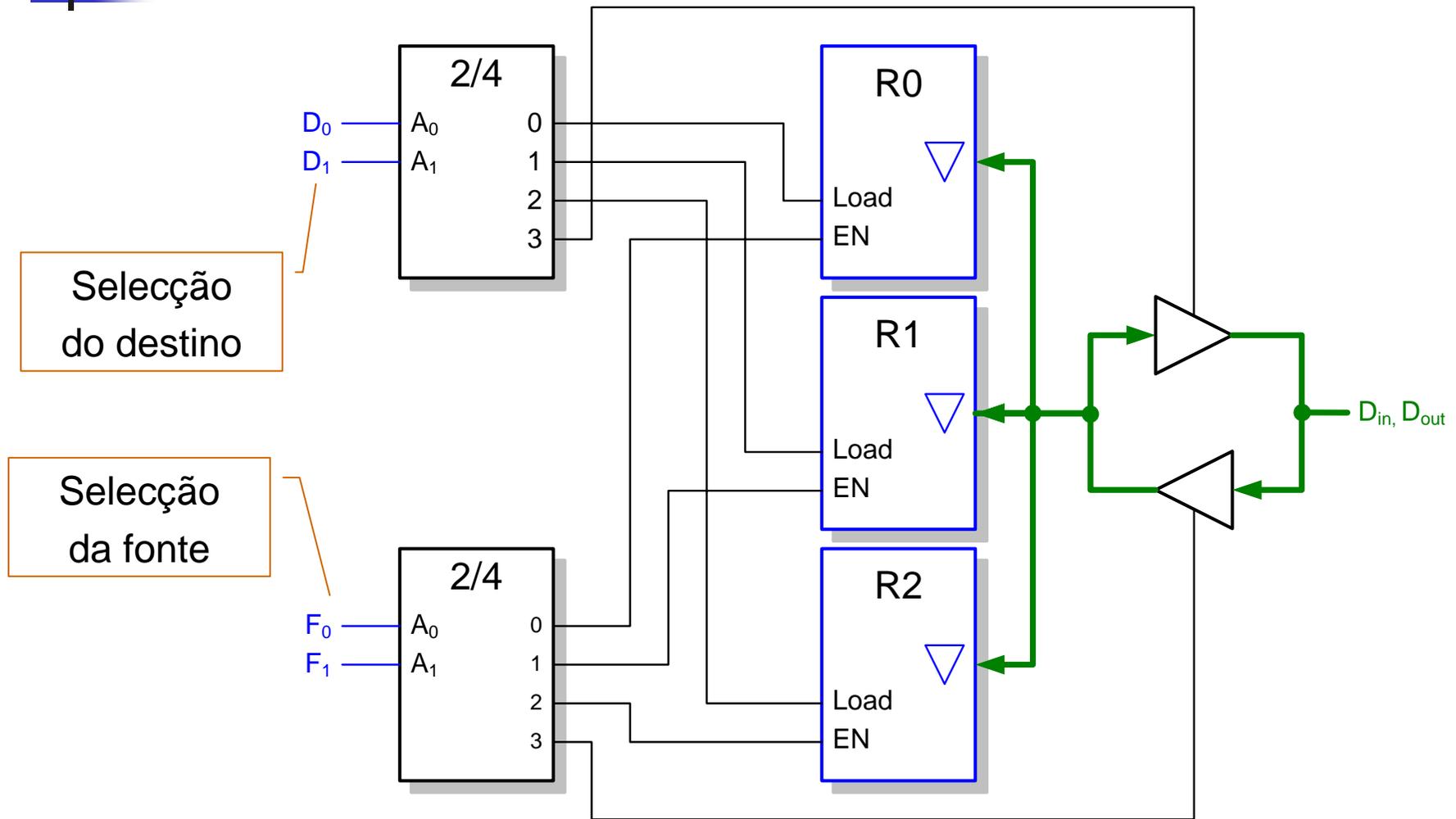


Transferências entre registos

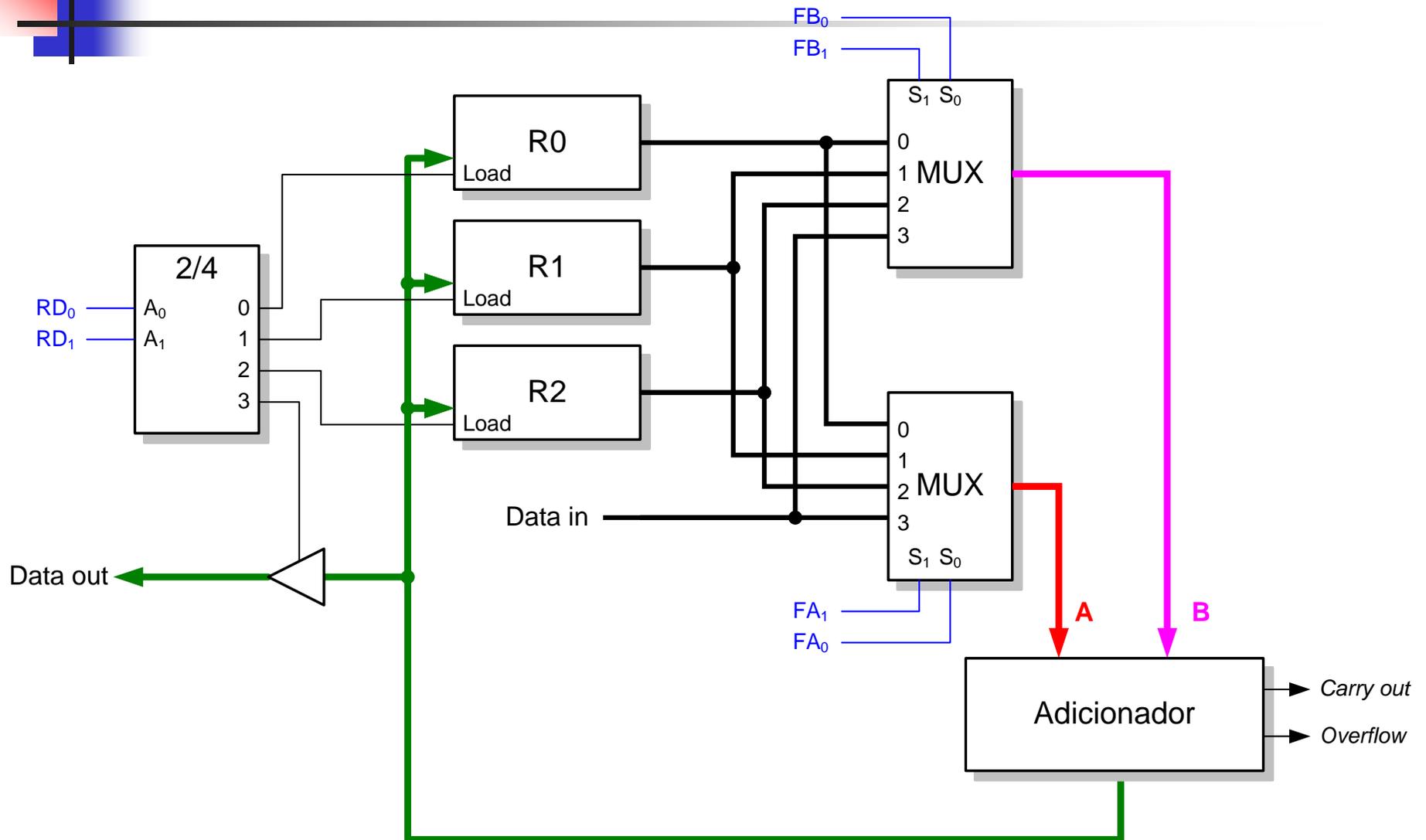
- Utilização de descodificadores

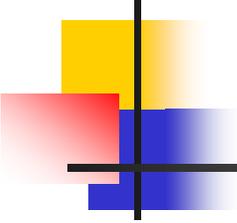


Transferências entre registos



Transferências entre registros





Sumário (Aulas 18 e 19)

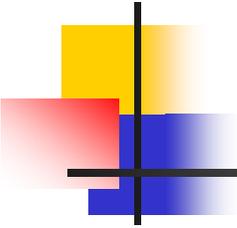
- Multiplicação binária
 - Multiplicação sem sinal
 - Utilizando vários somadores
 - Utilizando um único somador
- Controlo
 - Controlador para o multiplicador
 - Projecto com um contador
 - Projecto com FFs e com contador para implementação de ciclos



Sistemas de Computação

Paulo Santos

Multiplicação binária



Multiplicação binária

- Multiplicação (sem sinal)

$$\begin{array}{r} 1101 \text{ multiplicando (A)} \\ \times 1010 \text{ multiplicador (B)} \\ \hline 0000 \\ 1101 \\ 0000 \\ 1101 \\ \hline 10000010 \text{ produto (P)} \end{array}$$

Quando se multiplicam dois números de **n bits** (sem sinal), o resultado terá, no máximo, **2n bits**.

$$1101 (13) \times 1010 (10) = 10000010 (130)$$

Multiplicação binária

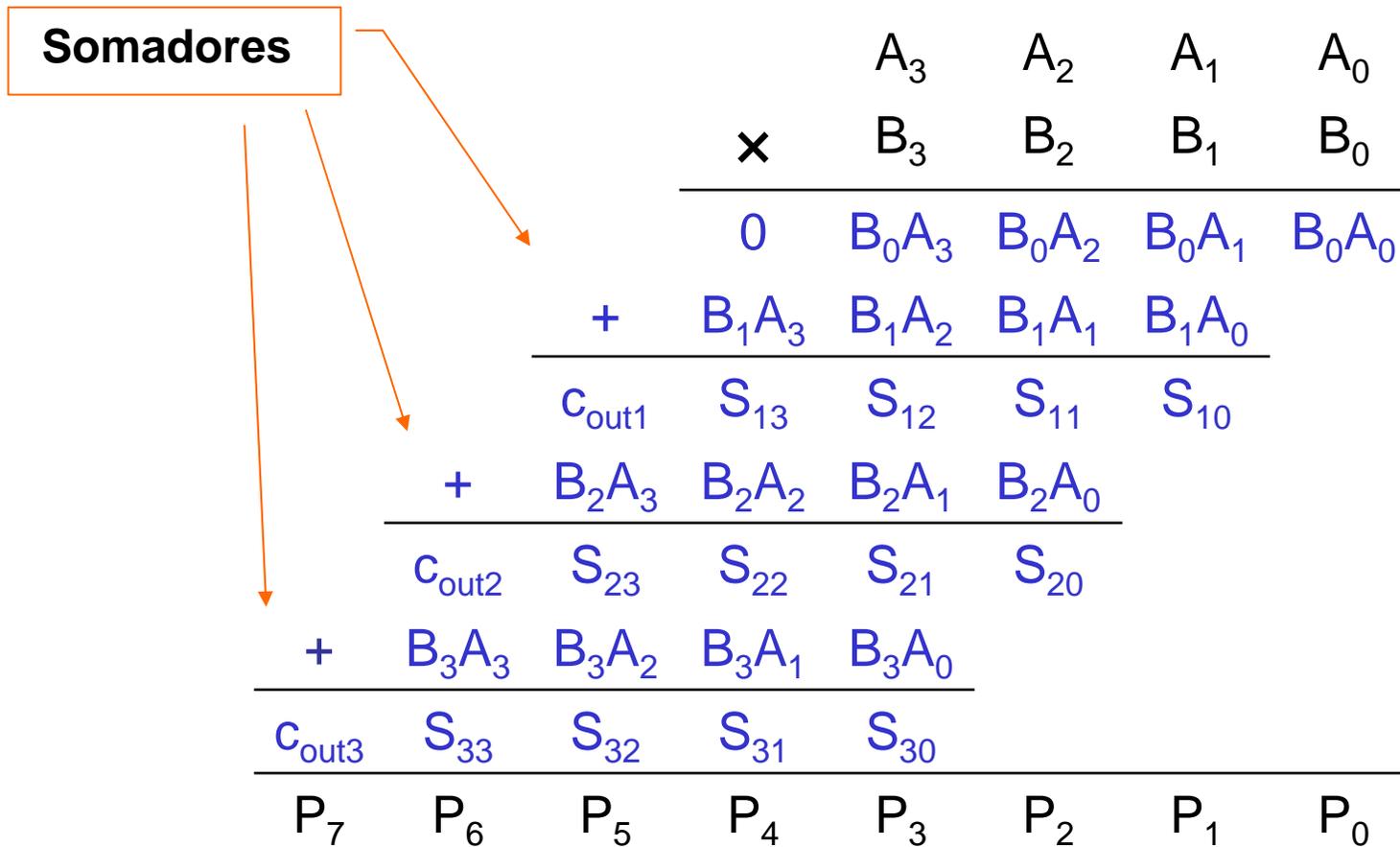
- Multiplicação (sem sinal)

				A_3	A_2	A_1	A_0
			\times	B_3	B_2	B_1	B_0
				B_0A_3	B_0A_2	B_0A_1	B_0A_0
			B_1A_3	B_1A_2	B_1A_1	B_1A_0	
		B_2A_3	B_2A_2	B_2A_1	B_2A_0		
	B_3A_3	B_3A_2	B_3A_1	B_3A_0			
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

ANDs entre os bits de A e os bits de B

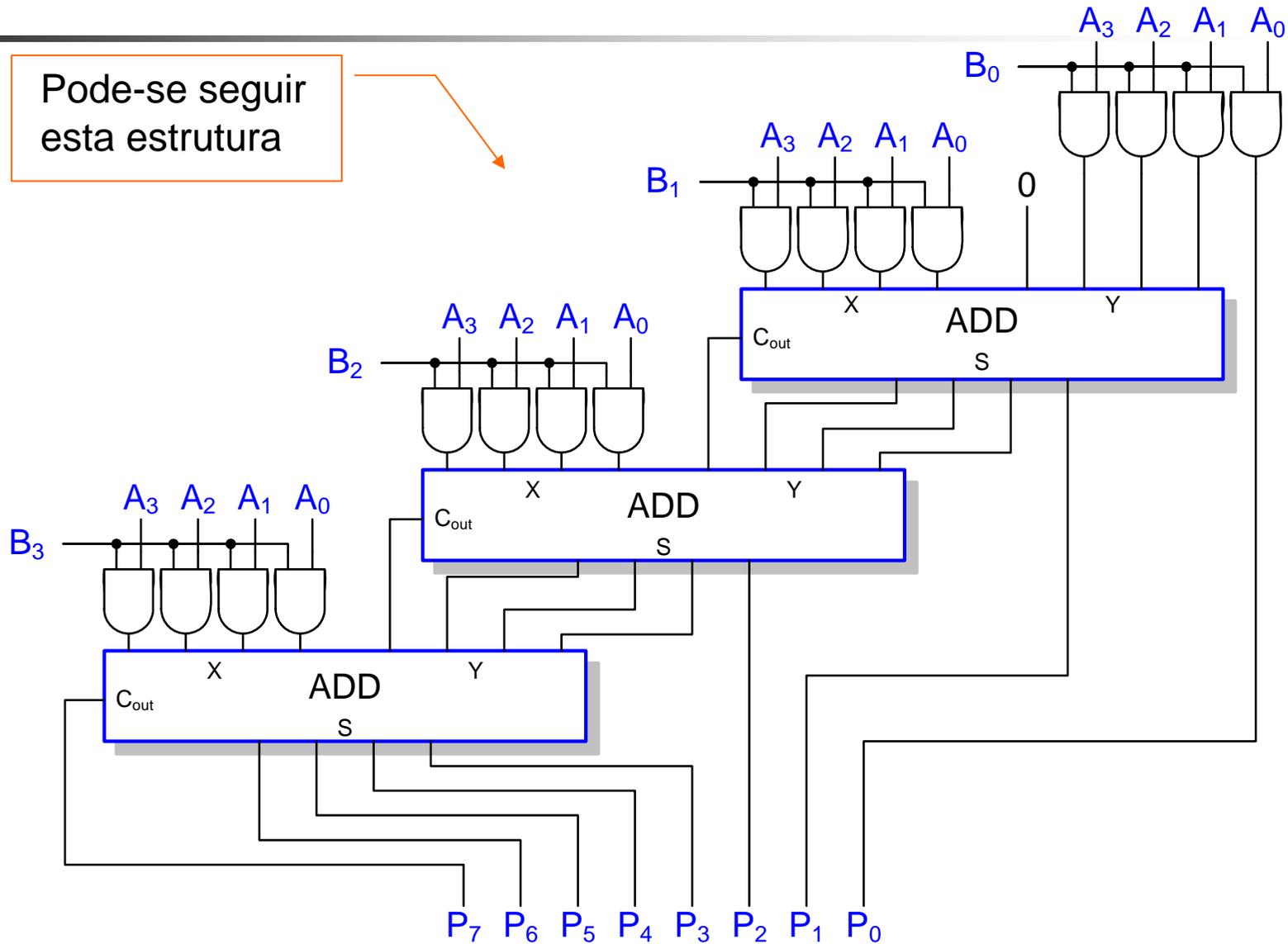
Multiplicação binária

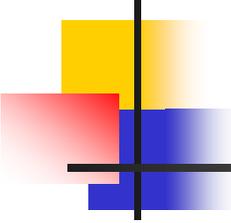
- Utilizando vários adicionadores...



Multiplicação binária

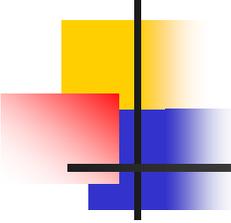
Pode-se seguir esta estrutura





Multiplicação binária

- Utilização de vários adicionadores
 - Se os números a multiplicar são compostos por n bits...
 - ...então são necessários $n-1$ adicionadores de n bits cada um
- Eventual problema: demasiado material...
 - Por exemplo, para multiplicar dois números de 32 bits seriam necessários 31 somadores de 32 bits (...para não falar das 1024 portas AND necessárias para combinar os bits de A e de B)



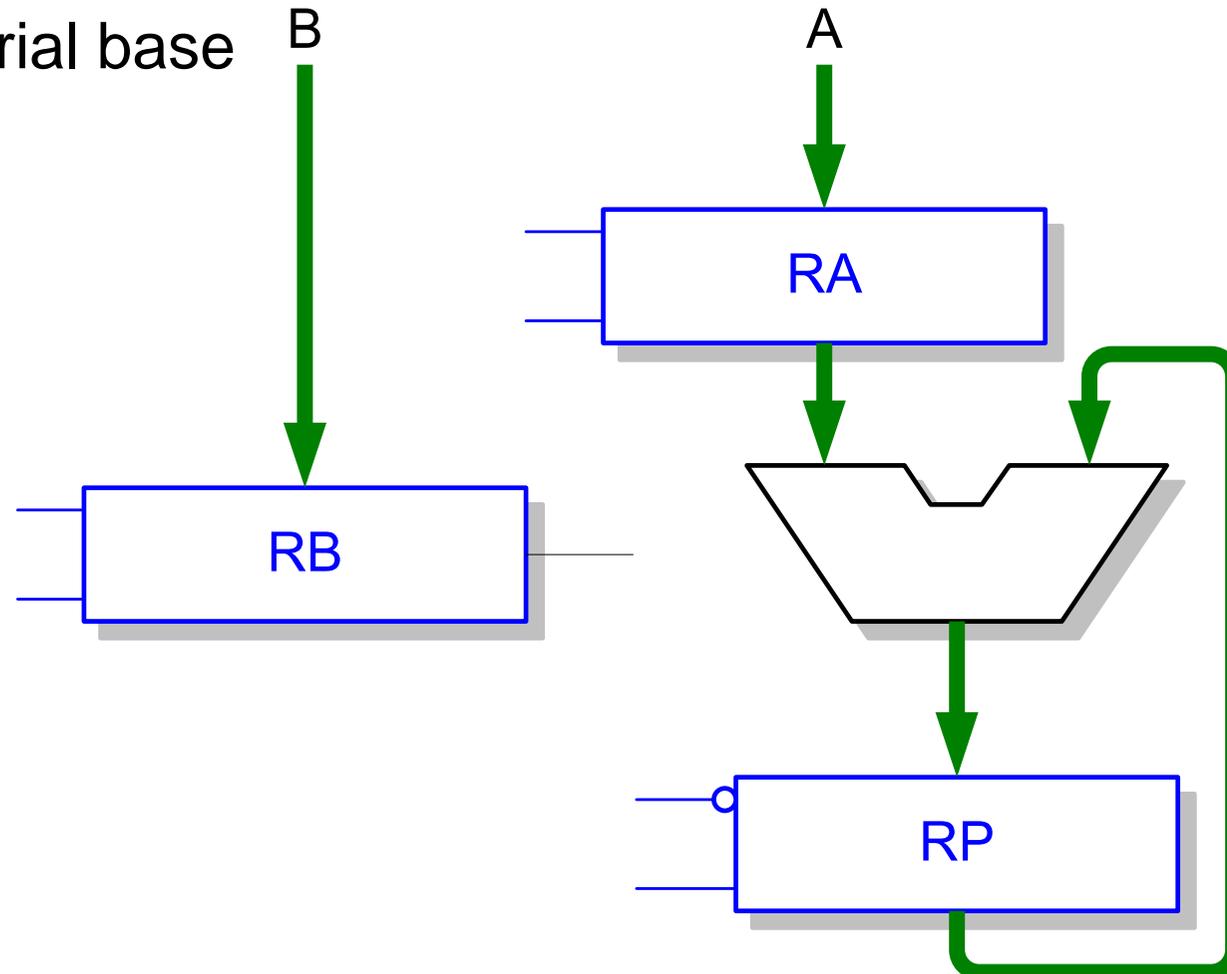
Multiplicação binária

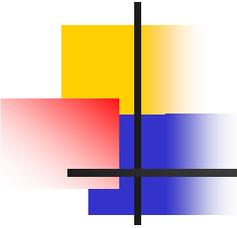
- Uma possível alternativa
 - Utilizar um adicionador e registos
 - O adicionador efectua todas as adições necessárias em n ciclos
 - Um registo RP que acumula os resultados das somas
 - 2 registos de deslocamento
 - Um registo RA que vai deslocando A para a esquerda
 - Outro registo RB que vai deslocando B para a direita, de forma a serem obtidos os seus bits individualmente
 - RP e RA são registos de $2n$ bits; RB é um registo de n bits

Multiplicação binária

- Utilização de um adicionador e registos

- Material base





Multiplicação binária

- Algoritmo

- Inicialização:

- $RP \leftarrow 0, RA \leftarrow A, RB \leftarrow B$

- Ciclo (n iterações)

- se ($S_{out} == 1$)

- $RP \leftarrow RP + RA$

- $RA \leftarrow sl(RA), RB \leftarrow sr(RB)$

- Ao fim de n iterações o resultado correcto fica guardado em RP

Multiplicação binária

■ Exemplo – multiplicar 1010 por 1001

Inicialização:

RP \leftarrow 0000 0000
RA \leftarrow 0000 1010
RB \leftarrow 1001

Ciclo 3 ($S_{out} = 0$)

RP \leftarrow 0000 1010
RA \leftarrow 0101 0000
RB \leftarrow 0001

Ciclo 1 ($S_{out} = 1$)

RP \leftarrow 0000 1010
RA \leftarrow 0001 0100
RB \leftarrow 0100

Ciclo 4 ($S_{out} = 1$)

RP \leftarrow 0101 1010
RA \leftarrow 1010 0000
RB \leftarrow 0000

Ciclo 2 ($S_{out} = 0$)

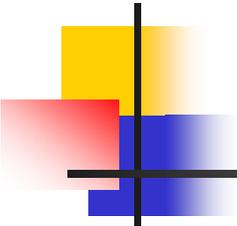
RP \leftarrow 0000 1010
RA \leftarrow 0010 1000
RB \leftarrow 0010

0000 0000
+ 0000 1010

0000 1010

0000 1010
+ 0101 0000

0101 1010



Multiplicação binária

- Sequência de sinais de controlo para multiplicar dois números de 4 bits

Estado	LD_A	SL_A	LD_B	SR_B	LD_P	CLR_P
Init	1	x	1	x	x	0
Ciclo 1	0	1	0	1	S _{out}	1
Ciclo 2	0	1	0	1	S _{out}	1
Ciclo 3	0	1	0	1	S _{out}	1
Ciclo 4	0	1	0	1	S _{out}	1
Fim	x	x	x	x	0	1

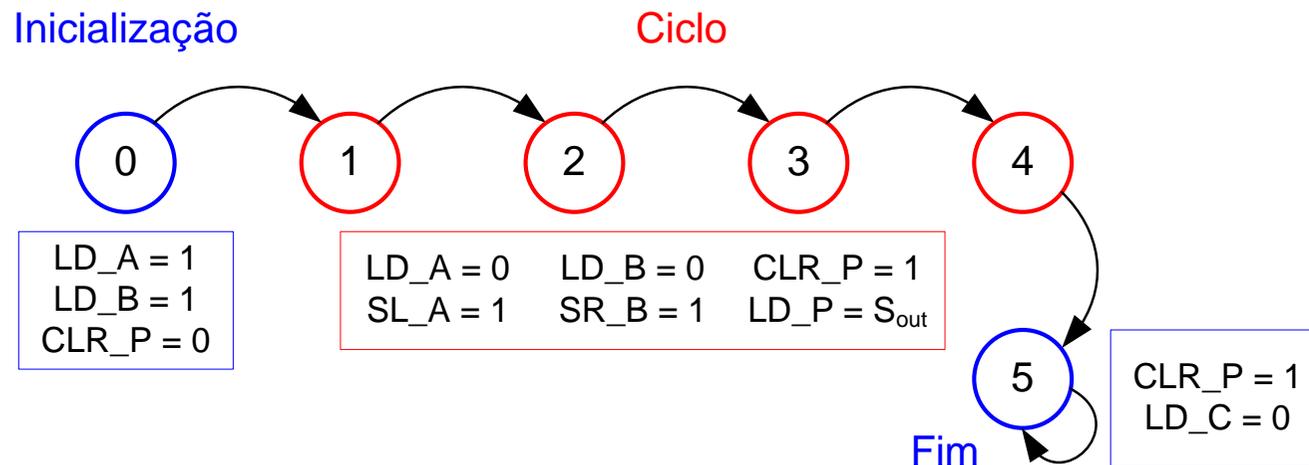
Admite-se que, quando activados em simultâneo, CLEAR prevalece sobre LOAD, e LOAD prevalece sobre SHIFT.

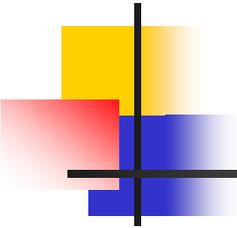
Controlador

■ Projecto de um controlador para o multiplicador

■ Controlador

- Circuito sequencial que gera a sequência de sinais de controlo para se obter a finalidade desejada
- As saídas do controlador são os sinais de controlo dos vários elementos do circuito a controlar
- O controlador para o multiplicador de 4 bits poderia seguir a sequência de estados:





Controlador

■ Tabela de transições de estados

$Q_2 Q_1 Q_0$	$Q_2' Q_1' Q_0'$	LD_A	SL_A	LD_B	SR_B	LD_P	CLR_P
0 0 0	0 0 1	1	x	1	x	x	0
0 0 1	0 1 0	0	1	0	1	S_{out}	1
0 1 0	0 1 1	0	1	0	1	S_{out}	1
0 1 1	1 0 0	0	1	0	1	S_{out}	1
1 0 0	1 0 1	0	1	0	1	S_{out}	1
1 0 1	1 0 1	x	x	x	x	0	1

A sequência seguida sugere a utilização de um contador...

Controlador

- Projecto com base num contador de 3 bits

- Sinais de controlo

$$LD_A = LD_B = \overline{Q_2 + Q_1 + Q_0}$$

$$CLR_P = Q_2 + Q_1 + Q_0$$

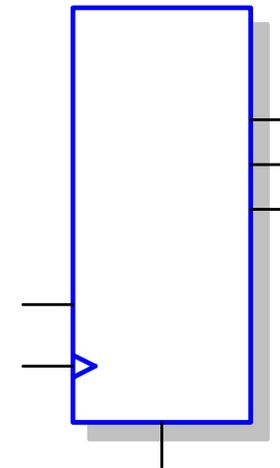
$$LD_P = S_{out} \overline{Q_2 Q_0}$$

$$SL_A = SR_B = 1$$

- Sinais aplicados ao contador

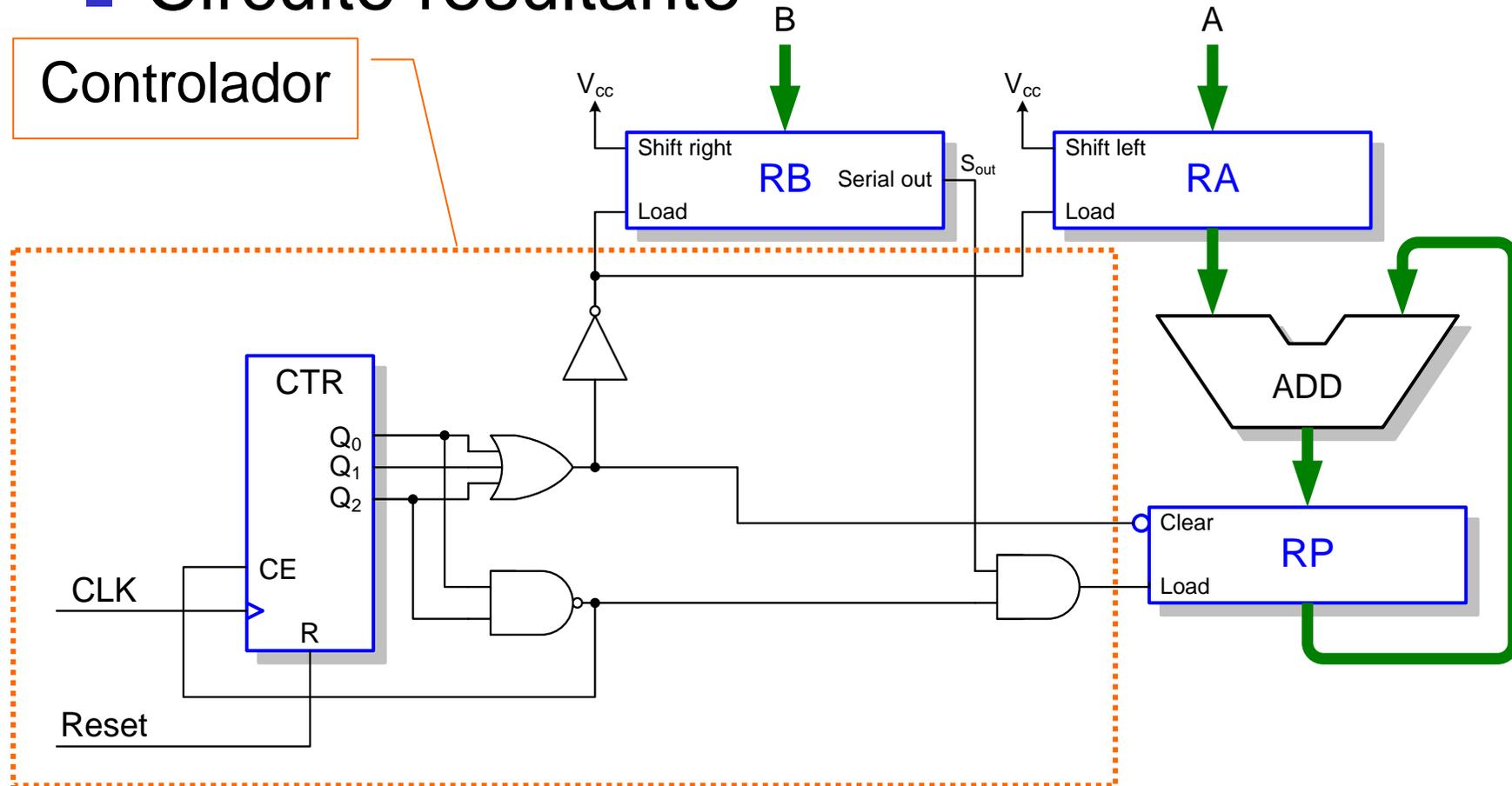
$$CE = Q_2 Q_0$$

Para parar no estado 101



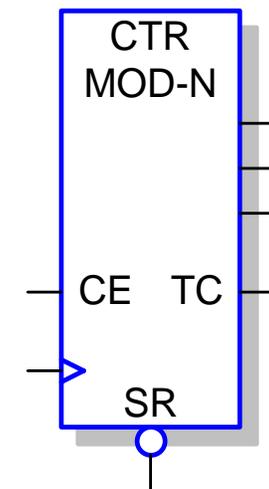
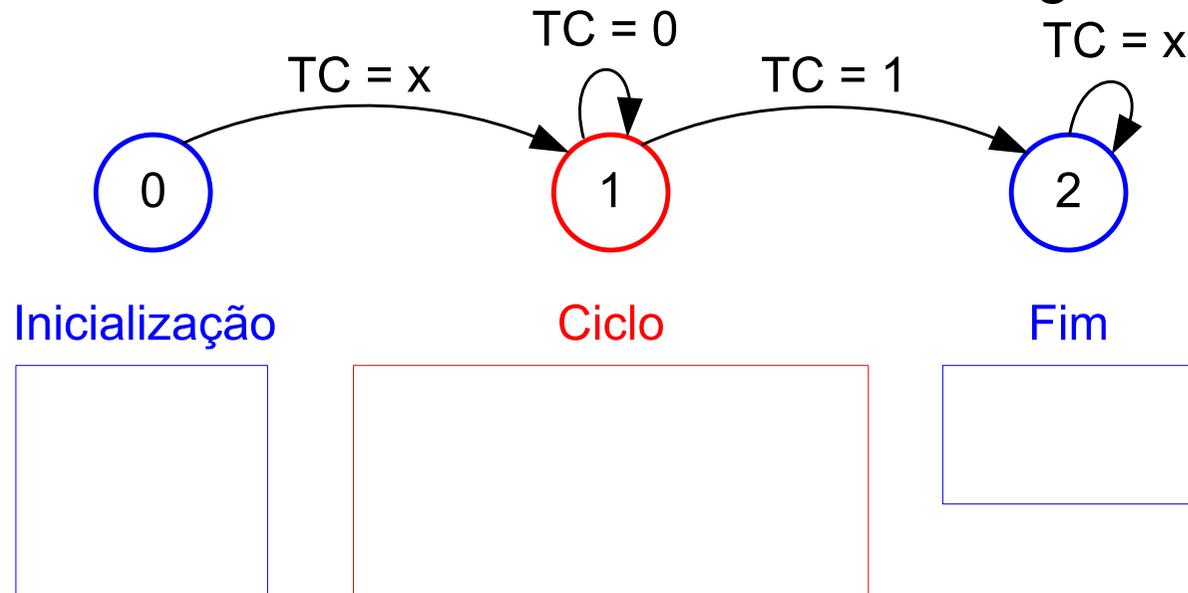
Controlador

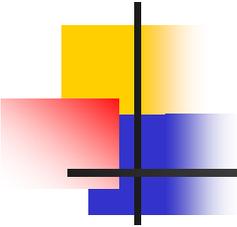
■ Circuito resultante



Controlador

- Outra abordagem (mais flexível)
 - Podia ser utilizado um contador módulo-N para contar o número de iterações do ciclo
 - O sinal TC assinala o fim de contagem





Controlador

- Transições de estados

Q_1	Q_0	T_C	Q_1'	Q_0'	J_1	K_1	J_0	K_0
0	0	0	0	1	0	x	1	x
0	0	1	0	1	0	x	1	x
0	1	0	0	1	0	x	x	0
0	1	1	1	0	1	x	x	1
1	0	0	1	0	x	0	0	x
1	0	1	1	0	x	0	0	x

$$J_1 = T_C Q_0$$

$$J_0 = \overline{Q_1}$$

$$K_1 = 0$$

$$K_0 = T_C$$

Controlador

■ Saídas (sinais de controlo)

$Q_1 Q_0$	CE	SR	LD_A	SL_A	LD_B	SR_B	LD_P	CLR_P
0 0	x	0	1	x	1	x	x	0
0 1	1	1	0	1	0	1	S_{out}	1
1 0	x	x	x	x	x	x	0	1

Sinais a aplicar ao contador:

CE – *count enable* (H)

SR – *synchronous reset* (L)

$$LD_A = LD_B = \overline{Q_0}$$

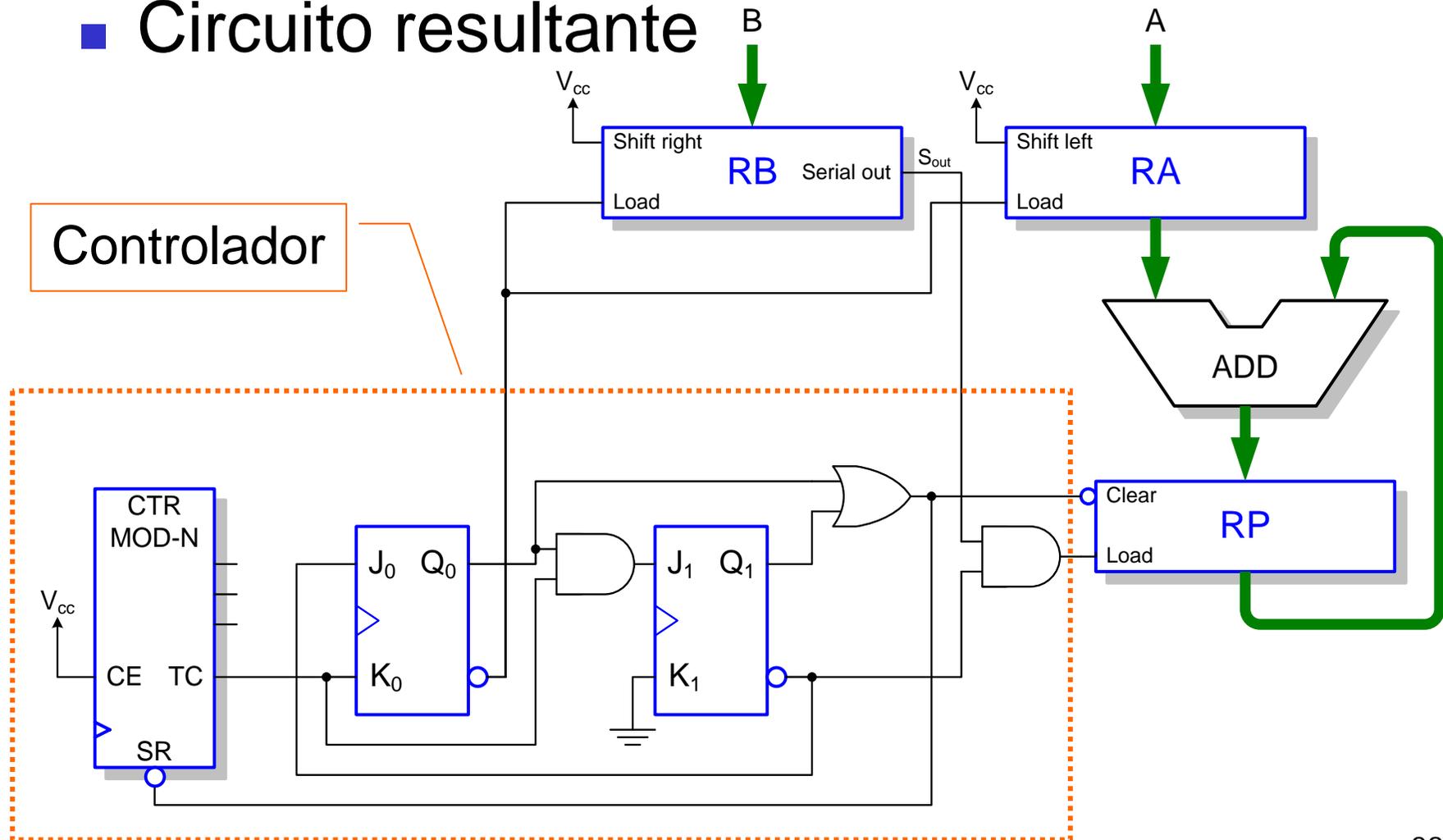
$$SL_A = SR_B = CE = 1$$

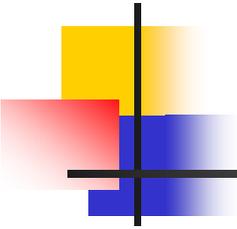
$$LD_P = S_{out} \overline{Q_1}$$

$$CLR_P = SR = Q_1 + Q_0$$

Controlador

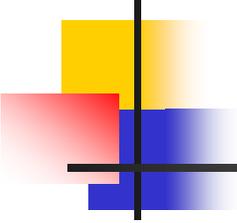
■ Circuito resultante





Controlador

- Passos para projecto de um controlador
 - Determinar quais os sinais de controlo a gerar (as saídas do controlador)
 - Elaborar o diagrama de estados
 - Obter tabela de transição de estados
 - Projectar o circuito
 - utilizando FFs
 - utilizando contadores
 - utilizando FFs e contadores para sequências de estados repetitivas (ciclos)



Sumário (Aulas 20 e 21)

- Memórias voláteis
 - Memórias SRAM e DRAM
 - Estrutura interna uma RAM
 - Ciclos de escrita e leitura
 - Organização da memória RAM
- Memórias não voláteis
 - ROM E PROM
 - EPROM, EEPROM e FLASH EEPROM



Sistemas de Computação

Paulo Santos

Memórias

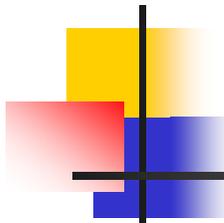


UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

IQF
Instituto para a Qualidade
na Formação, I.P.

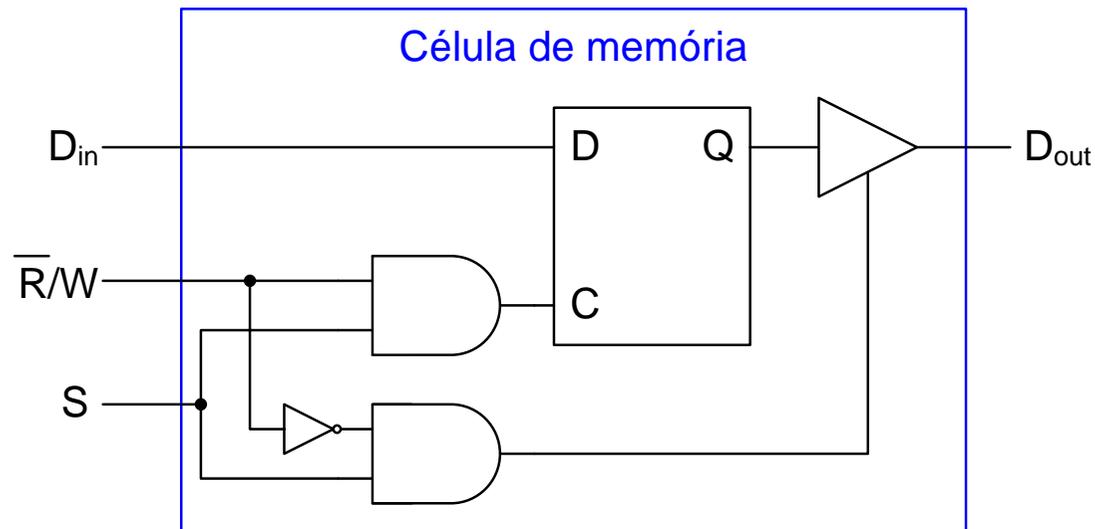


Tipos de Memórias

- Memórias do tipo RAM *Random-access memory*
 - Memórias **voláteis** – a informação perde-se quando se deixa de fornecer energia eléctrica
 - Utilizadas para leitura e escrita da informação
- Memórias do tipo ROM *Read-only memory*
 - Memórias **não-voláteis** – a informação continua armazenada quando se deixa de fornecer energia eléctrica
 - No princípio utilizadas apenas para leitura da informação guardada
 - Mas actualmente existem memórias derivadas da ROM que são programáveis, algumas delas utilizadas tanto para leitura como para escrita

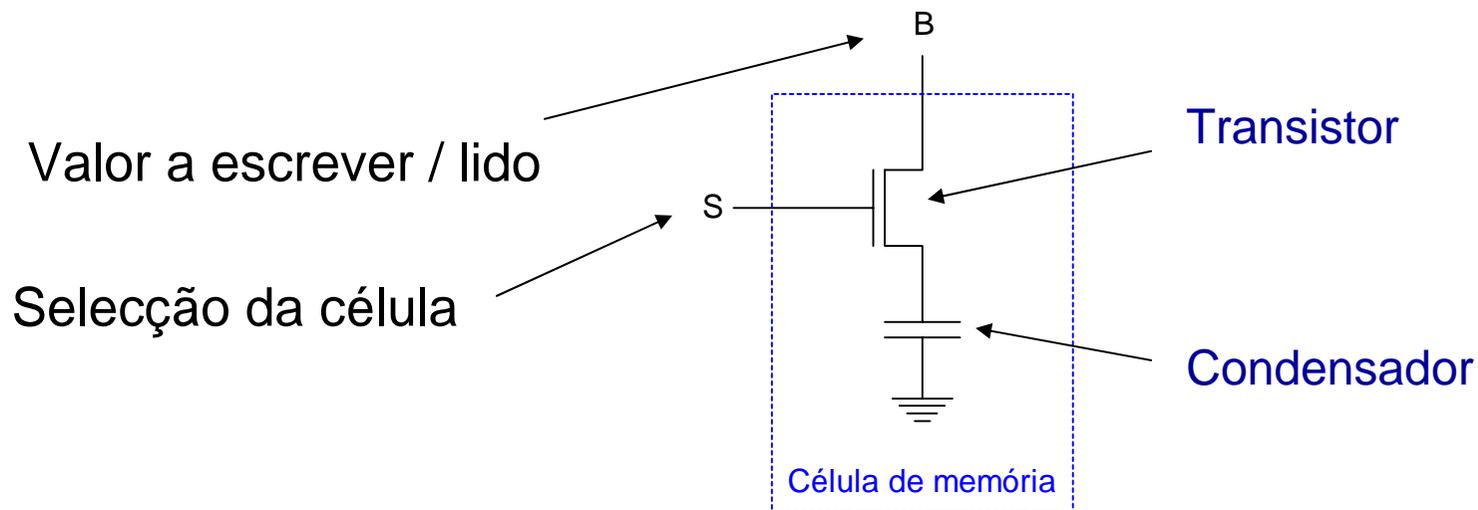
Memórias RAM

- Estáticas – SRAM (*Static* RAM)
 - Células de memória:
 - *latches* ou *flip-flops*
 - Rápidas – tempos de acesso baixos para leitura e para escrita
 - **Utilizadas tipicamente como memórias *cache*** (associadas ao processador)



Memórias RAM

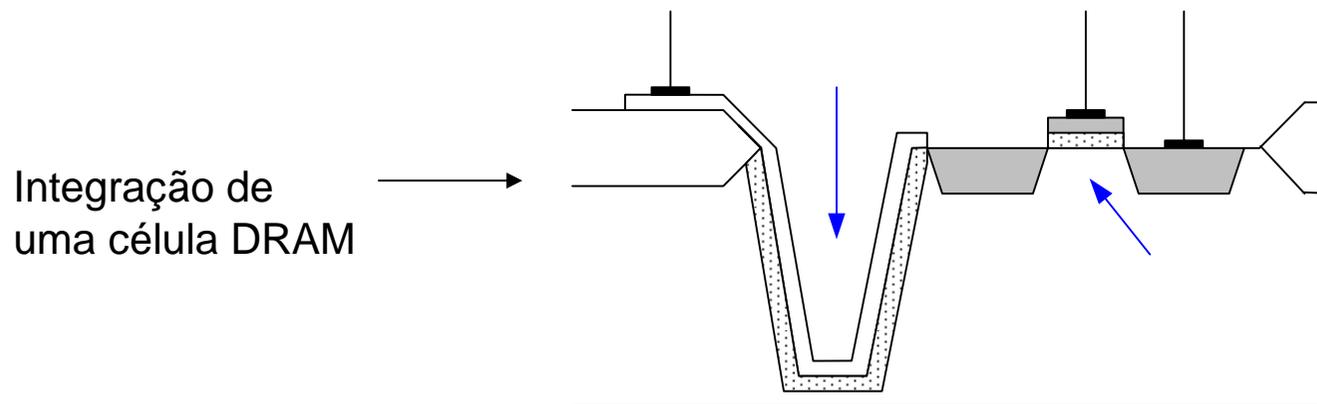
- Dinâmicas – DRAM (*Dynamic RAM*)
 - Células de memória:
 - **Pares transistor-condensador**, que conseguem manter o nível lógico armazenado durante curtos espaços de tempo
 - Necessitam por isso de ciclos de refrescamento periódicos para reposição dos níveis lógicos nos condensadores



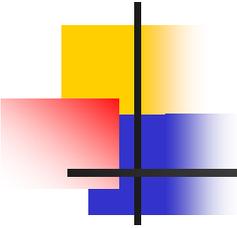
Memórias RAM

■ Dinâmicas – DRAM (cont.)

- Mais lentas que as SRAMs, essencialmente devido à amplificação dos níveis de tensão nos condensadores e ao refrescamento
- Comparando com as memórias estáticas, consegue-se maior capacidade de armazenamento com menor custo
 - Consegue-se maior integração do que nas SRAM a menor custo



- **Utilizadas como memória principal de um computador**

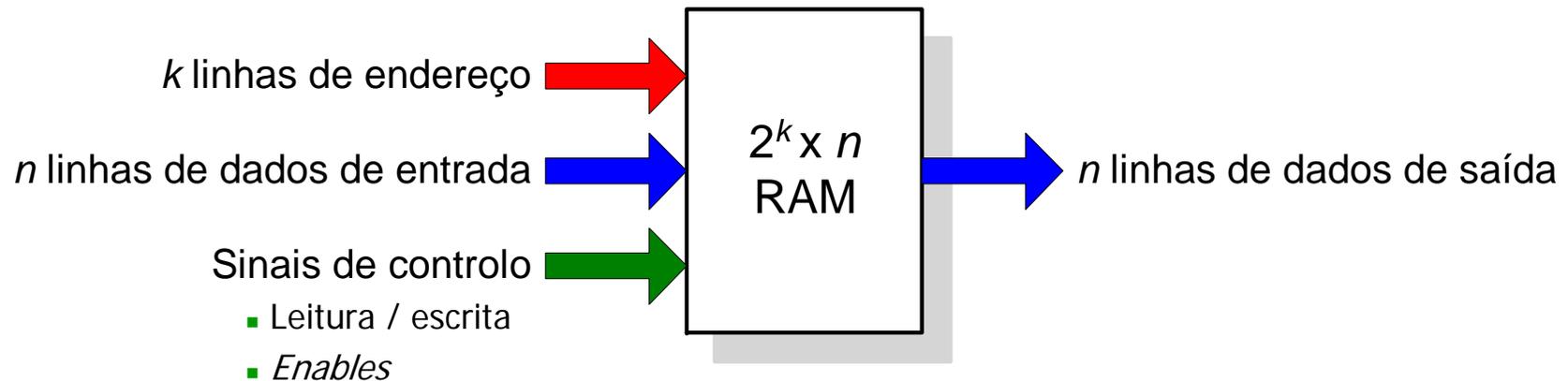


Memórias RAM

- Principais tipos de DRAM
 - SDRAM (*Synchronous DRAM*)
 - Síncronas com o relógio de sistema
 - DDR-SDRAM (*Double Data Rate SDRAM*)
 - Síncronas com o relógio de sistema e reagem a ambos os flancos do relógio
 - Muito utilizadas actualmente
 - RDRAM ou RAMbus
 - Eficientes em leituras e escritas por blocos
 - Têm esta designação porque os módulos de memória estão ligadas em série num barramento (*bus*) próprio
 - Pouco utilizadas, embora sejam rápidas

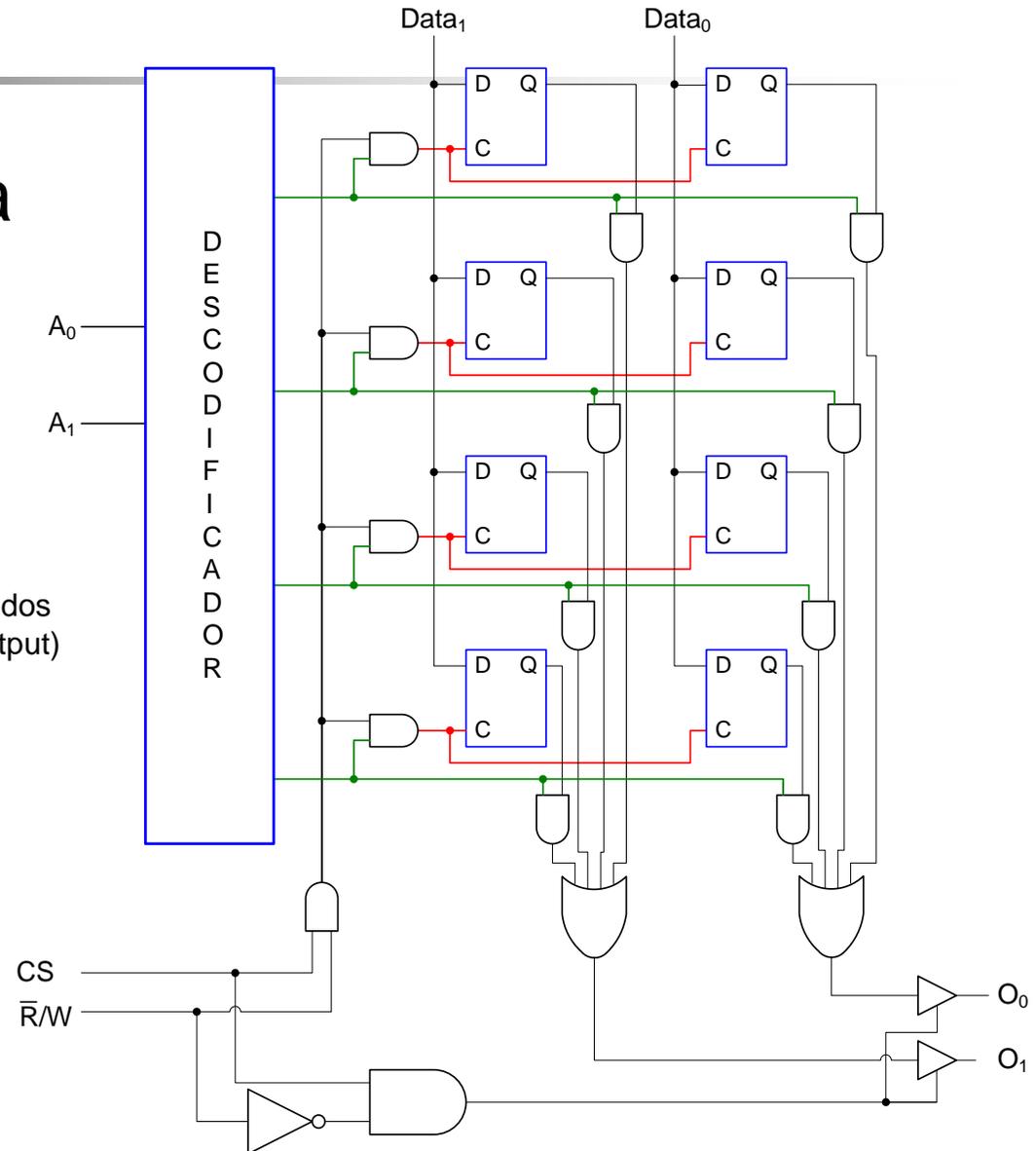
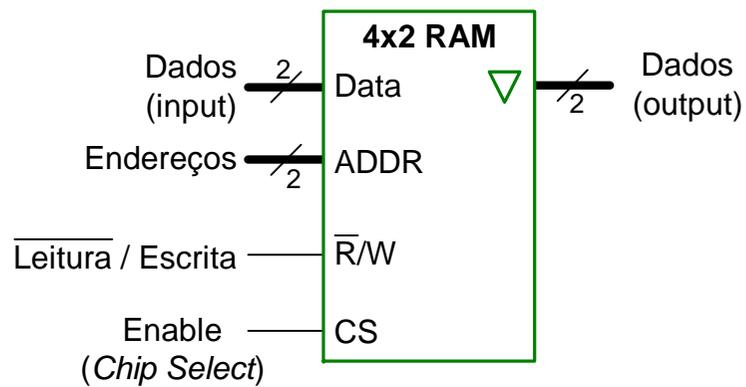
Memórias RAM

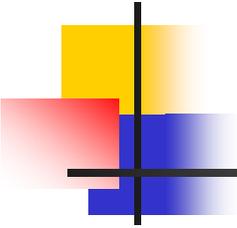
- Acesso e capacidade de uma RAM
 - k linhas de endereço c/ n bits por endereço
 - 2^k endereços ou palavras
 - 1 palavra = n bits
 - Capacidade = 2^k palavras = $2^k \times n$ bits



Memórias RAM

- Estrutura interna (SRAM)



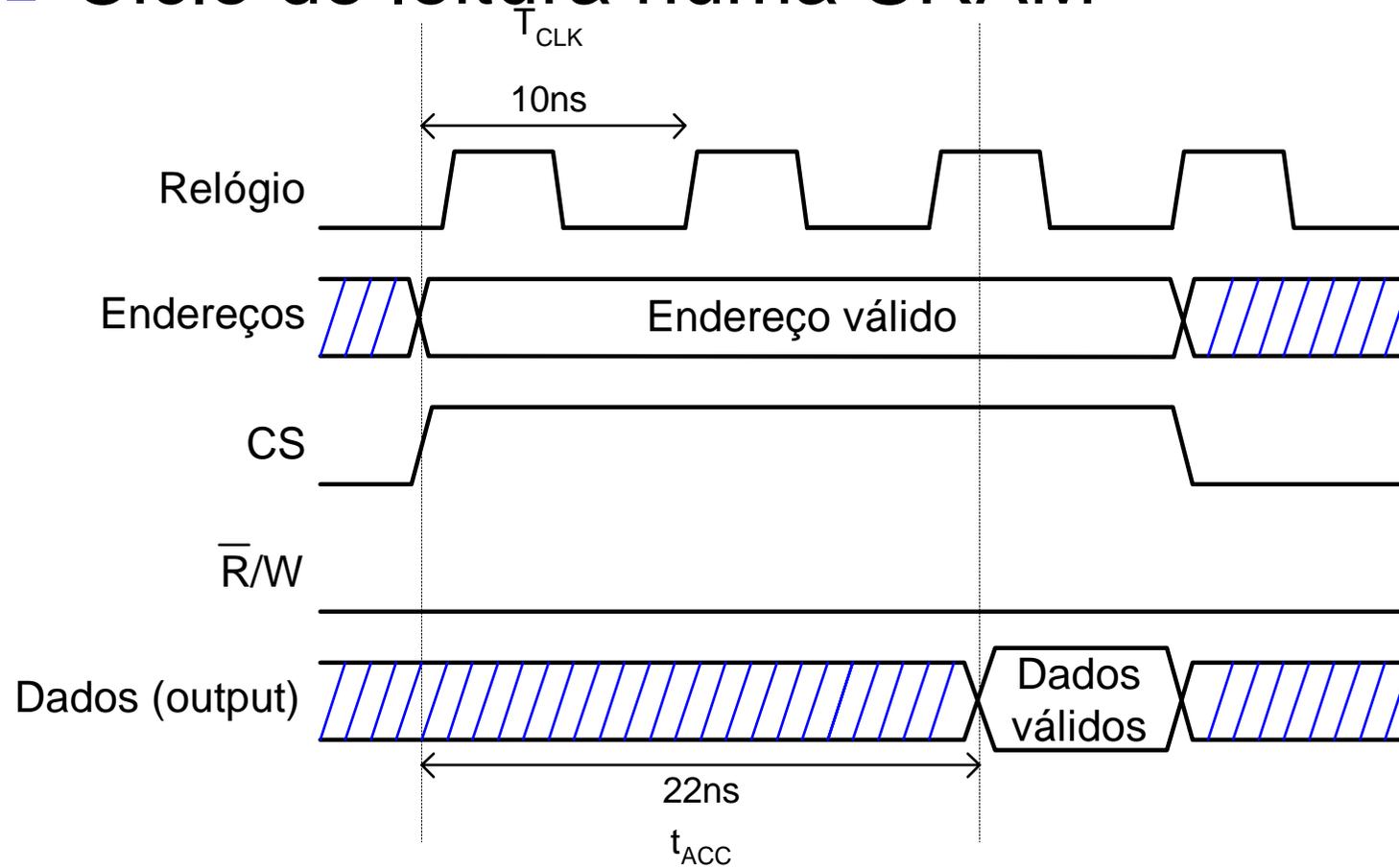


Memórias RAM

- Ciclos de escrita e leitura numa SRAM
 - Temporizações mais importantes:
 - Tempo de acesso para leitura
Tempo que demora entre ser apresentado o endereço a ler até os dados nele contidos aparecerem na saída
 - Tempo de escrita
Tempo que demora entre ser apresentado o endereço a escrever até os dados serem de facto escritos na memória
 - T_{HOLD}
Tempo mínimo para o qual é necessário manter estáveis os endereços (ou dados) após os dados terem sido escritos na memória

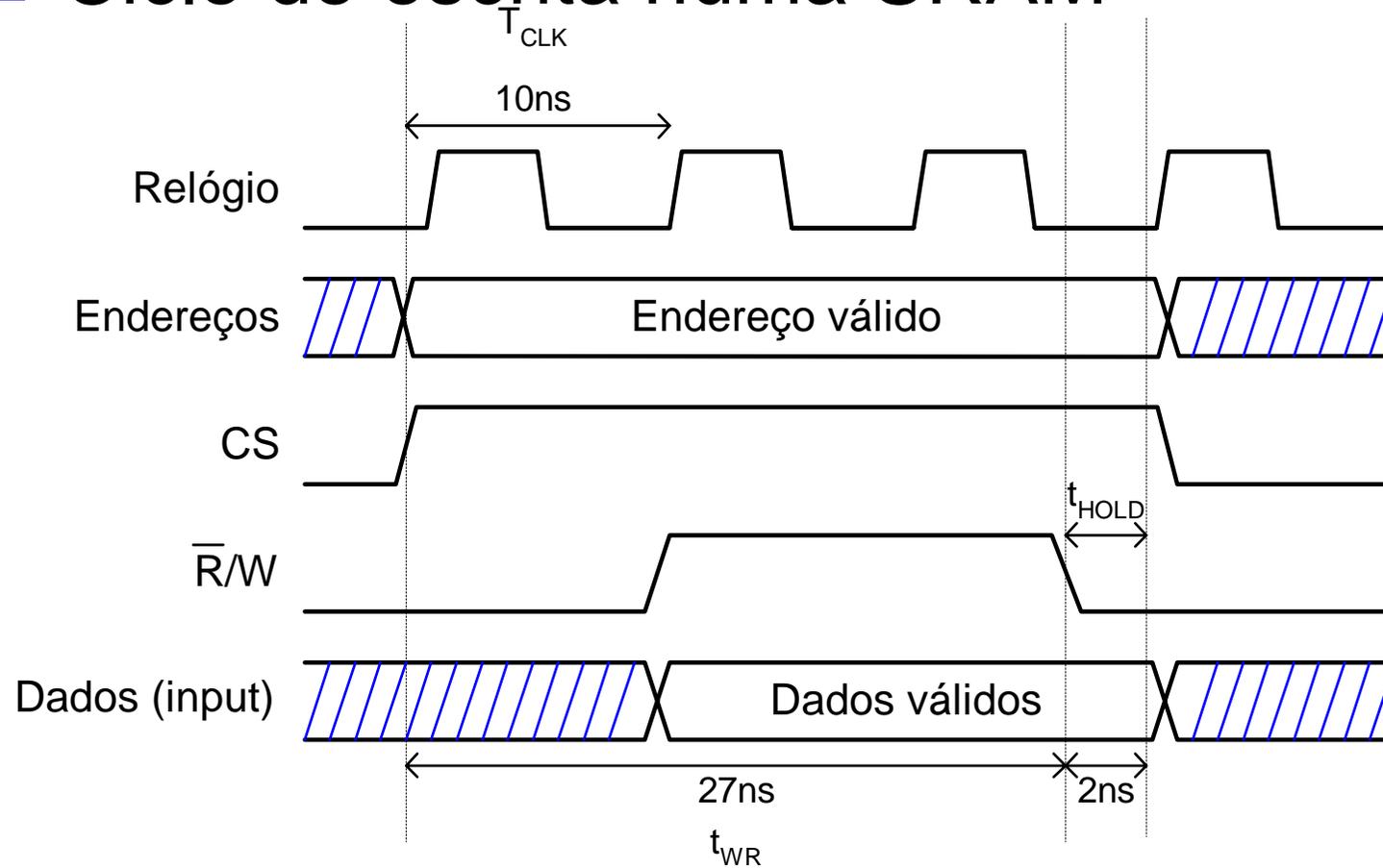
Memórias RAM

■ Ciclo de leitura numa SRAM



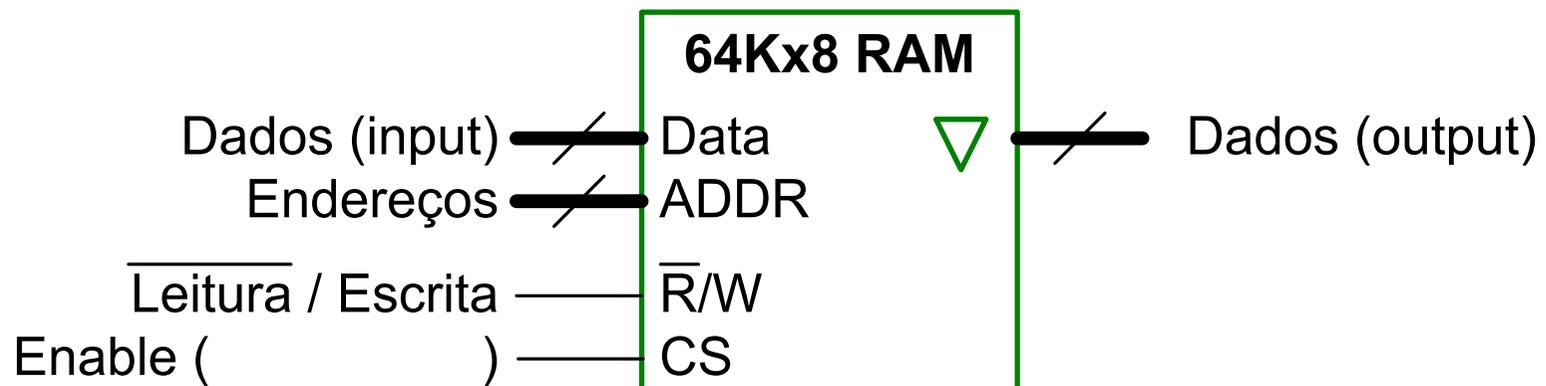
Memórias RAM

■ Ciclo de escrita numa SRAM



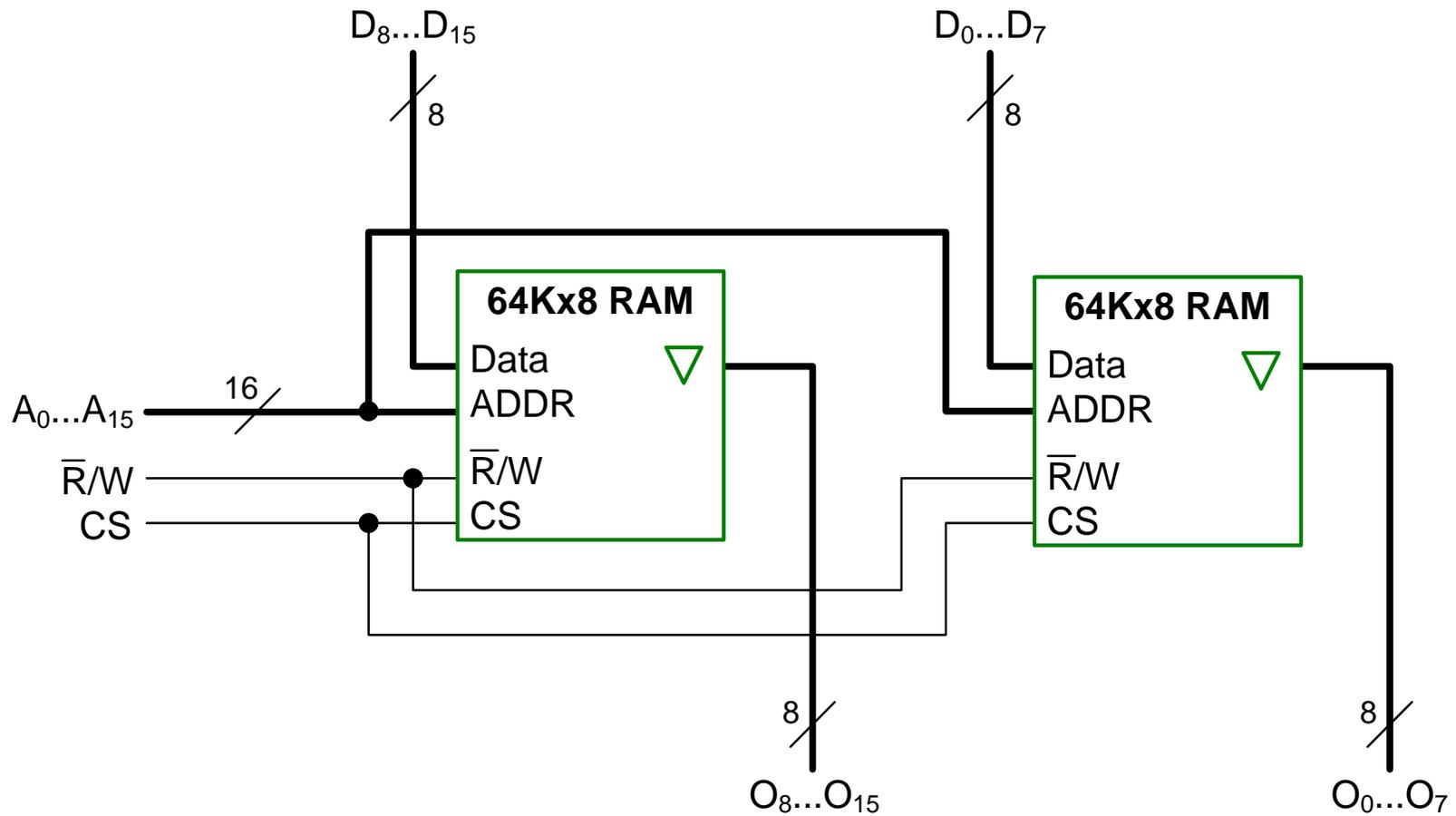
Memórias RAM

- Organização da memória
 - É possível projectar memórias com maior capacidade associando *chips* de memória.
 - Exemplos: projectar uma RAM 64K x 16 e outra 256K x 8 a partir de RAMs 64K x 8



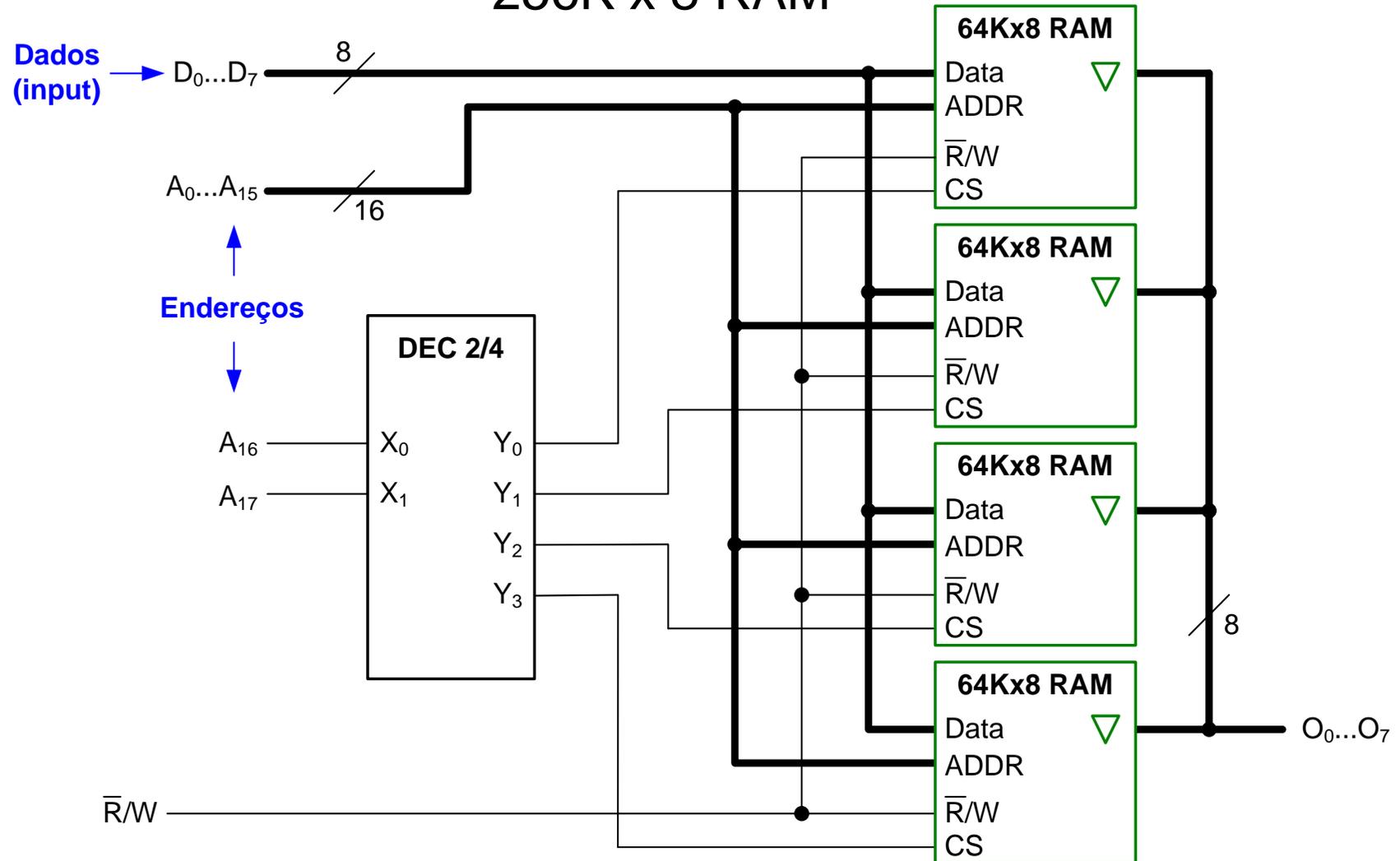
Memórias RAM

64K x 16 RAM



Memórias RAM

256K x 8 RAM

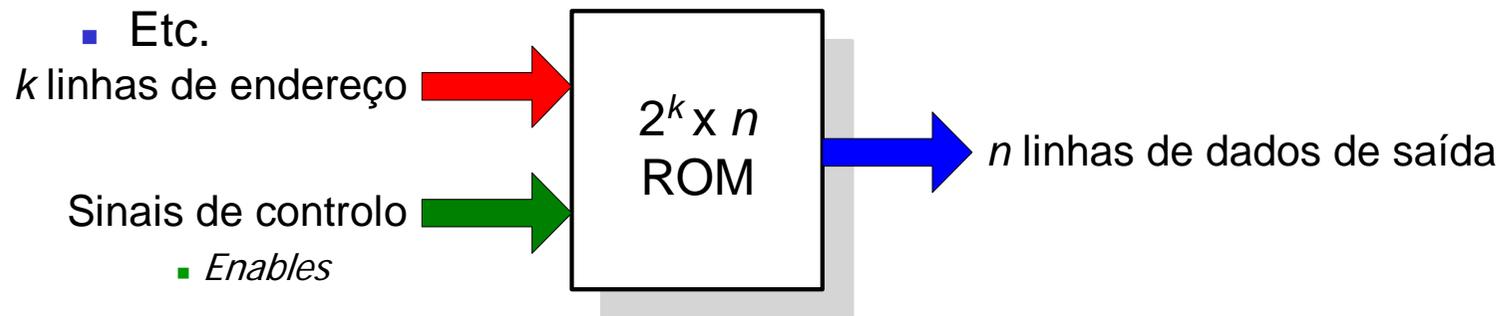


Memórias ROM

■ ROM

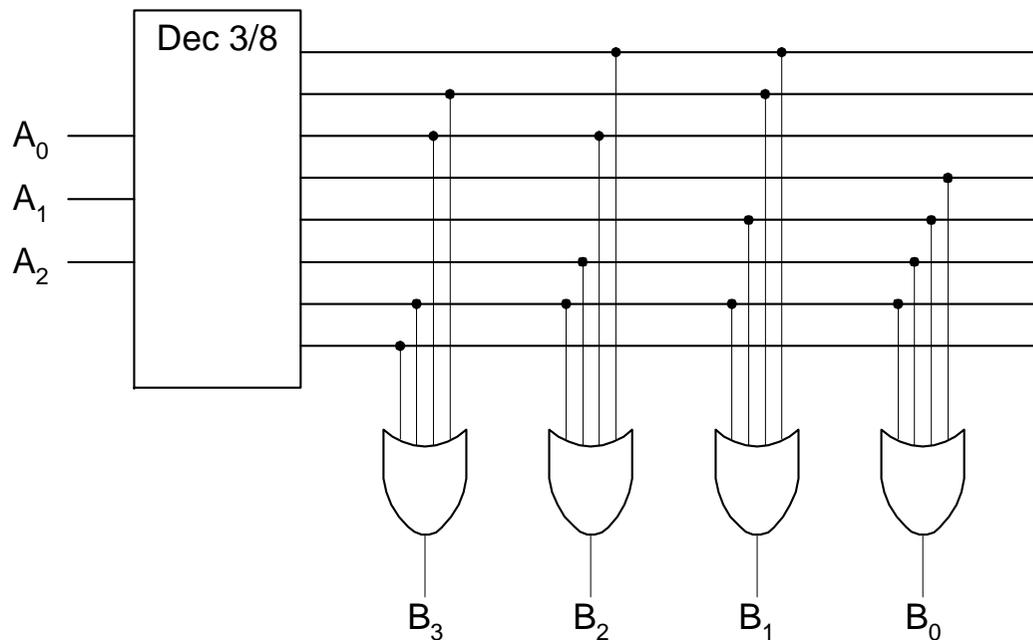
Read-only memory

- Construídas pelo fabricante, mediante especificação fornecida pelo cliente.
- Sem flexibilidade para alteração do conteúdo – só permite leitura da informação armazenada.
- Utilização
 - Guardar informação necessária ao arranque de sistemas
 - Tabelação de conversão de códigos (e.g. binário natural -> BCD)
 - Tabelação de operações aritméticas (e.g. logaritmos, divisões)
 - Etc.

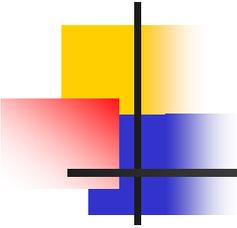


Memórias ROM

- ROM (exemplo)



Endereço	Conteúdo
000	0110
001	1010
010	1100
011	0001
100	0011
101	0101
110	1111
111	1000



Memórias ROM

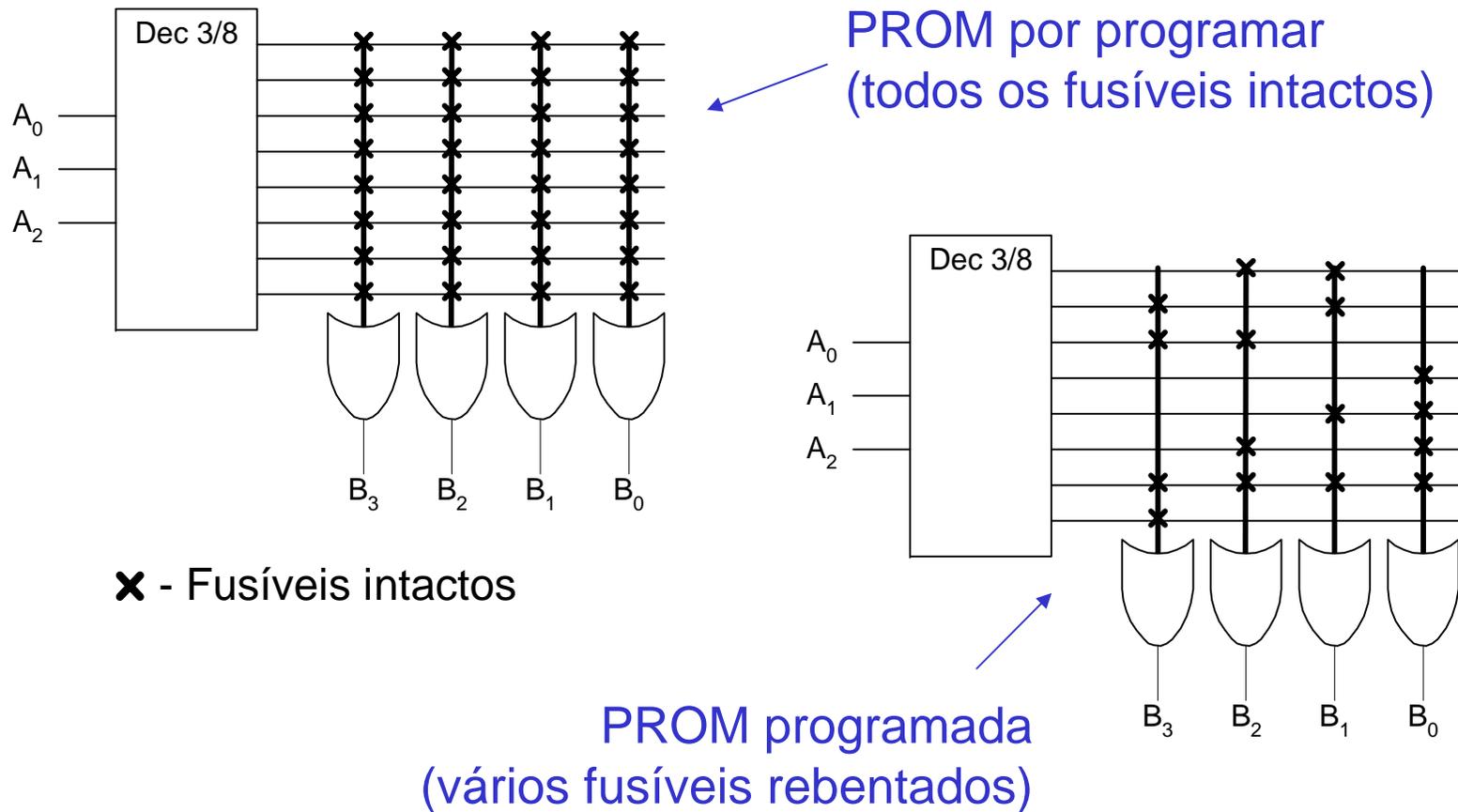
- PROM

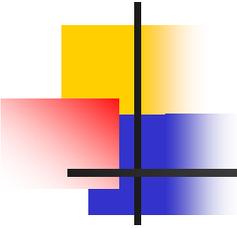
- Programmable read-only memory*

- Semelhante a uma ROM, mas permite uma única programação
 - Permite a especificação de ROMs do lado do utilizador
 - Pouca flexibilidade – uma única programação
 - A programação é geralmente feita através de rebentamento de fusíveis nas ligações entre as linhas de endereços descodificados e as linhas de saída
 - Daí a razão para só poder ser programada uma vez – uma vez rebentados os fusíveis, as ligações são quebradas permanentemente

Memórias ROM

- PROM (exemplo de programação)



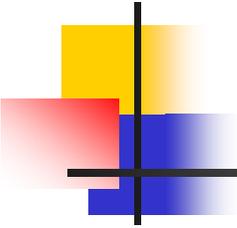


Memórias ROM

- EPROM

Erasable programmable read-only memory

- Permite **múltiplas programações**
- Para que seja possível reprogramar o dispositivo, este deve ser previamente submetido a radiação ultra-violeta durante algumas dezenas de minutos – esta operação apaga o conteúdo armazenado.
- A reprogramação é feita através de impulsos eléctricos
- Custo significativamente mais elevado que uma ROM
- Maior flexibilidade na utilização de ROM durante o desenvolvimento e teste de um sistema digital

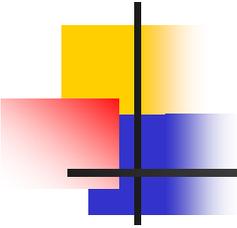


Memórias ROM

- EEPROM

Electrically erasable programmable read-only memory

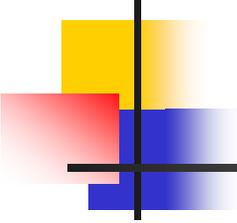
- Utilização idêntica à EPROM, mas consegue-se apagar o conteúdo através de impulsos eléctricos
- Maior flexibilidade por reunir as funcionalidade de uma RAM e uma ROM simultaneamente
- Comparando com uma RAM:
 - Operações de escrita muito mais lentas (devido às operações de apagar e reprogramar)
 - As operações de leitura podem ser da mesma ordem de grandeza



Memórias ROM

■ FLASH EEPROM

- Variantes de memórias EEPROM, habitualmente utilizadas em fotografia digital, pen-disks, PDAs, etc.
- Incluem toda a lógica necessária para reprogramação, e esta é muito mais rápida do que numa EEPROM convencional
 - Mas, mesmo assim, as operações de escrita são muito mais lentas do que numa RAM
- O tempo de vida dos dados armazenados é superior a 10 anos, e o nº máximo de reprogramações é da ordem dos milhões
 - O que não é problemático para a maioria das aplicações
- Actualmente começam já a substituir discos e CD-ROMs (e.g. pen-disks)



Sumário (Aulas 22 e 23)

- Barramentos
 - Definições
 - Tipos de barramentos
 - Descodificação de endereços
- Processador
 - Arquitectura básica:
Máquina de *von Newmann*
 - *Datapath* e unidade de controlo



Sistemas de Computação

Paulo Santos

Processador e Barramentos

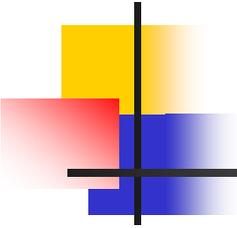


UNIÃO EUROPEIA

Fundo Social Europeu

prime
Programa de Incentivos à
Modernização da Economia

IQF
Instituto para a Qualidade
na Formação, I.P.

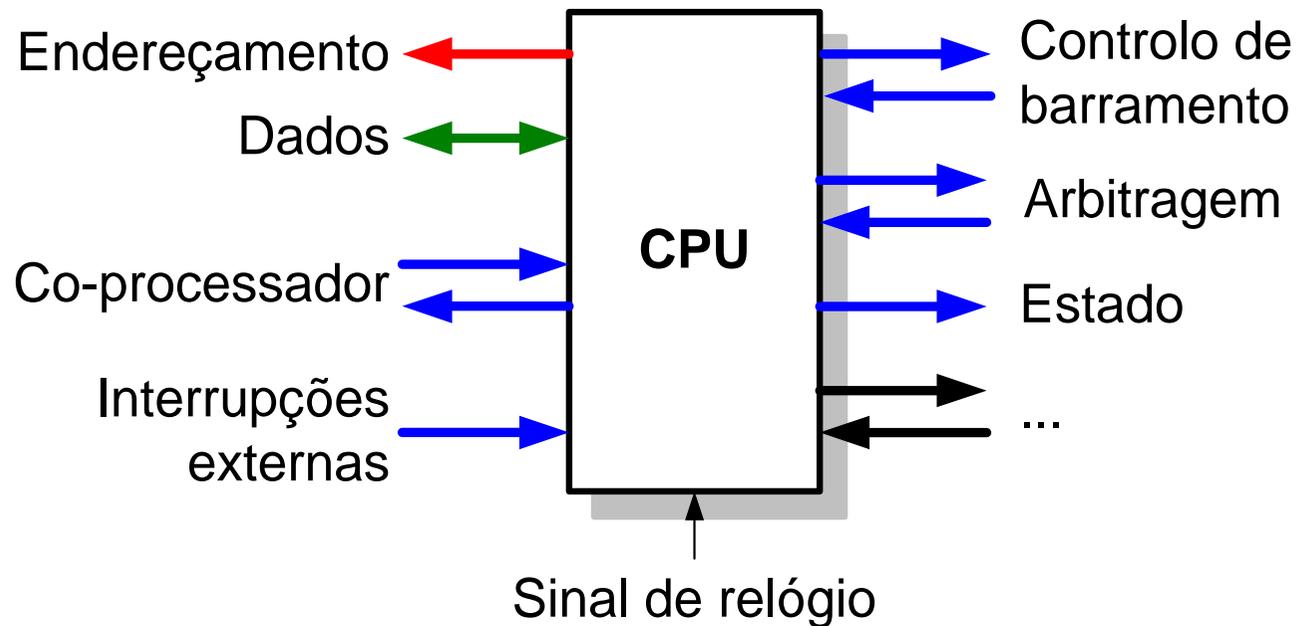


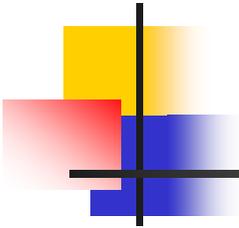
CPU

- CPU (*Central Processing Unit*)
 - É a principal unidade responsável pela actividade de um computador
 - Executa sequências de instruções definidas em programas
 - Comunica com os restantes elementos do sistema através dos seus pinos, ligados a **barramentos** externos

CPU

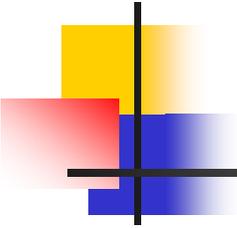
- Existem vários tipos de pinos
 - Dados: m bits – D_0 a D_{m-1}
 - Endereços: n bits – A_0 a A_{n-1}
 - Controlo: para diversas finalidades (ver figura)





Barramentos

- Um **barramento** (ou *Bus*) é um conjunto de linhas partilhado por vários dispositivos
- Num computador existem vários tipos de barramentos
 - Cada barramento é caracterizado pelas suas características eléctricas e físicas
 - O ritmo da transferência de dados varia consoante o tipo barramento
 - Cada barramento obedece a um dado conjunto de regras de comunicação – **protocolo**

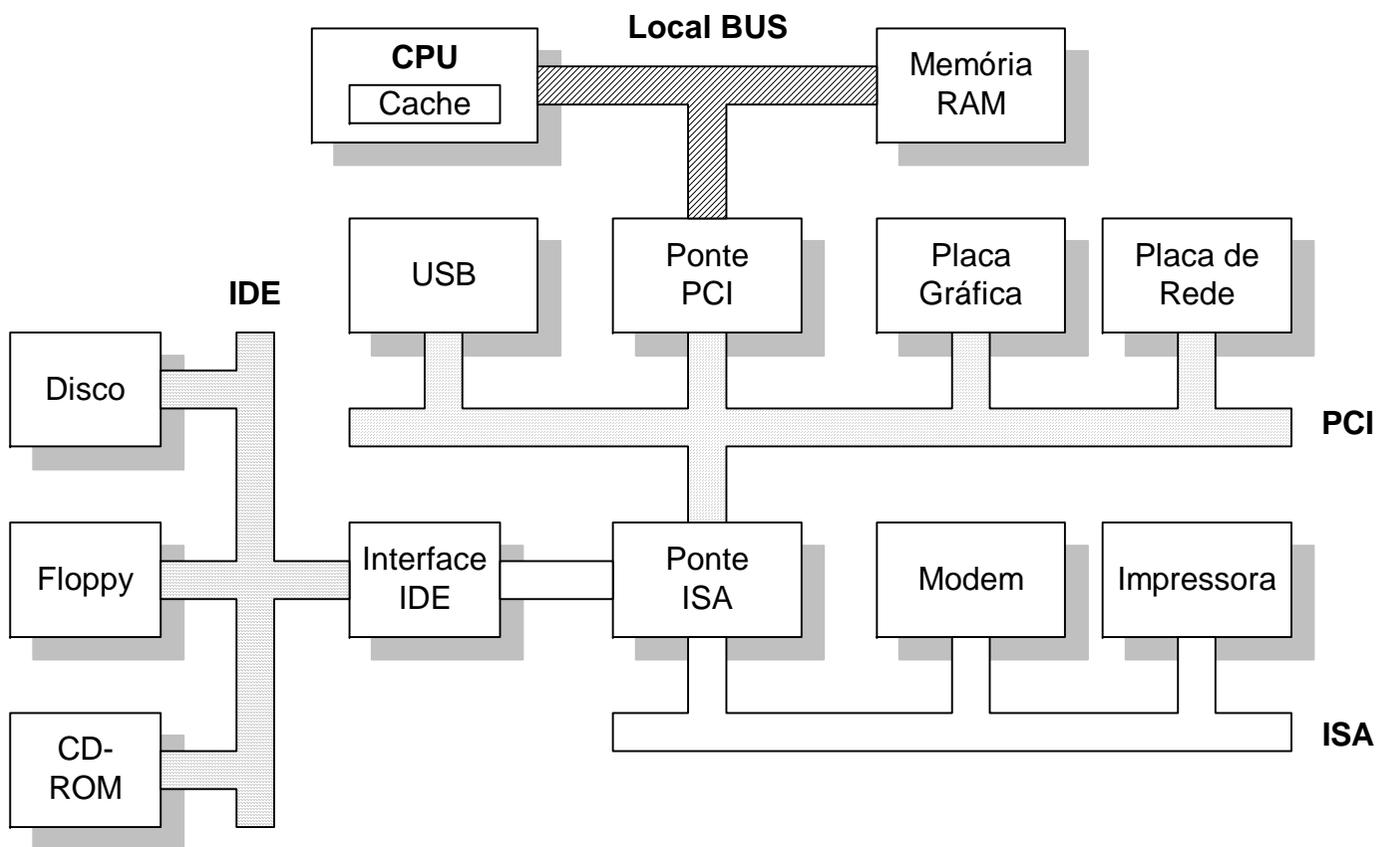


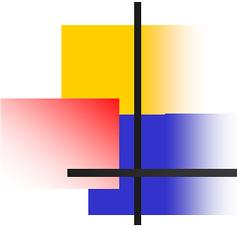
Barramentos

- Alguns exemplos:
 - ISA (Industry Standard Architecture)
 - Periféricos lentos (impressora, modem)
 - PCI (Peripheral Component Interconnect)
 - Periféricos rápidos (placa gráfica, placa de rede)
 - IDE (Integrated Drive Electronics) e SCSI (Small Computer System Interface)
 - Discos, CD-ROMS, DVDs
 - USB (Universal Serial Bus)
 - Periféricos externos
 - FireWire
 - Eletrónica de consumo

Barramentos

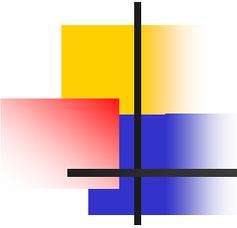
- Possível arquitectura:





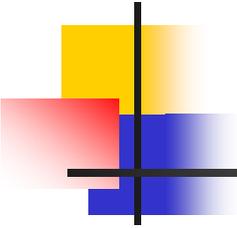
Barramentos

- Temporizações
 - Barramentos síncronos
 - Existe um sinal de referência – o relógio do barramento
 - Vantagem – maior compatibilidade
 - Desvantagem – pode-se perder eficiência
 - Barramentos assíncronos
 - Os dispositivos reagem através da variação dos níveis lógicos em linhas específicas
 - Vantagem – maior eficiência



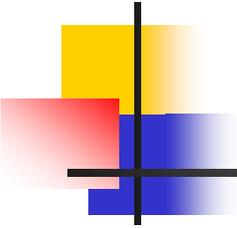
Barramentos

- Transferências
 - Paralelo
 - Uma linha por bit
 - Exige mais linhas no barramento
 - Série
 - A mesma linha serve para transmitir vários bits
 - Exige menos linhas



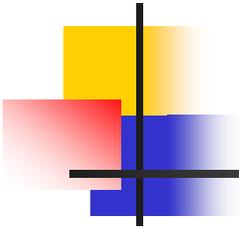
Barramentos

- **Master e Slave** de um barramento
 - **Master** – dispositivo que requisita o barramento tomando a iniciativa numa transferência de dados
 - **Slave** – dispositivo que serve o pedido
- A maioria dos dispositivos tanto pode ser *master* como ser *slave*
 - excepto a memória que é sempre *slave*



Barramentos

- Ligação de vários dispositivos periféricos
 - Vários dispositivos podem partilhar o mesmo barramento. Tal é possível devido a
 - Utilização de **buffers tri-state**
 - Permite que apenas os dispositivos que fazem a transferência fiquem em contacto com o BUS
 - Existência de **arbitragem** no barramento
 - O “árbitro” impede que dois dispositivos diferentes sejam *master* simultaneamente.
 - As linhas de dados dos barramentos são geralmente bidireccionais
 - O que significa que entradas e saídas de um dispositivo se encontram ligadas ao mesmo ponto



Descodificação de Endereços

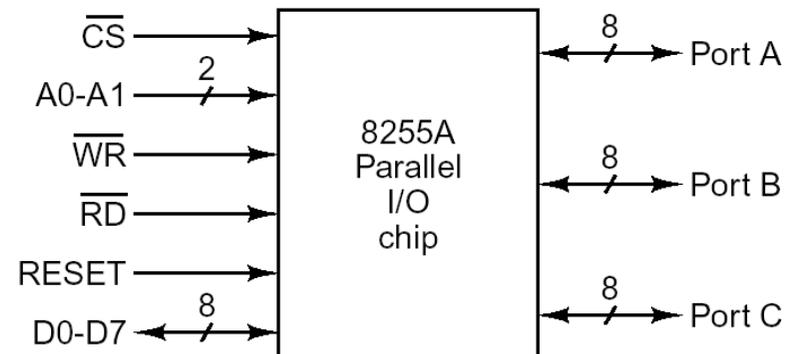
- Os periféricos podem ser mapeados para posições de memória (*memory-mapped I/O*)
 - Desta maneira poupam-se linhas de barramento dedicadas para cada periférico
- As transferências de dados podem ser vistas como operações de leitura / escrita em memória
- Põe-se então um problema: como activar os *chips* correspondentes ao controlo destes periféricos ?
 - Utiliza-se descodificação de endereços
 - Desta maneira a cada periférico fica associado um conjunto de endereços

Descodificação de Endereços

- Exemplo:

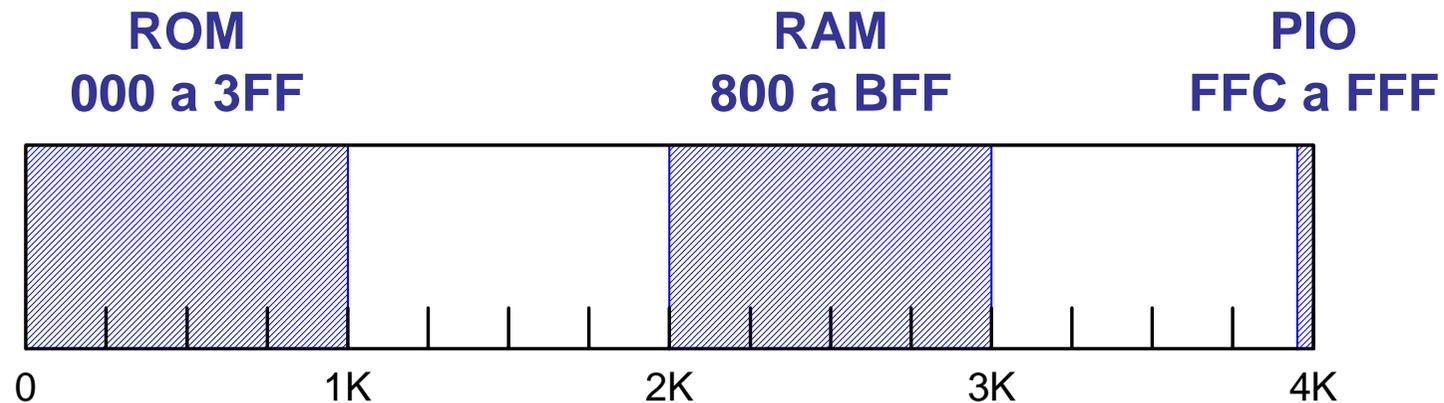
Suponha que se pretende construir um pequeno sistema com um CPU de 12 bits de endereçamento e 8 bits de dados e que contenha:

- uma ROM 1Kx8
- uma RAM 1Kx8
- um controlador de I/O 8255A (Parallel I/O) com acesso a 3 portos de 8 bits (a cada porto poderá estar associado um periférico como teclado, impressora, terminal de texto)



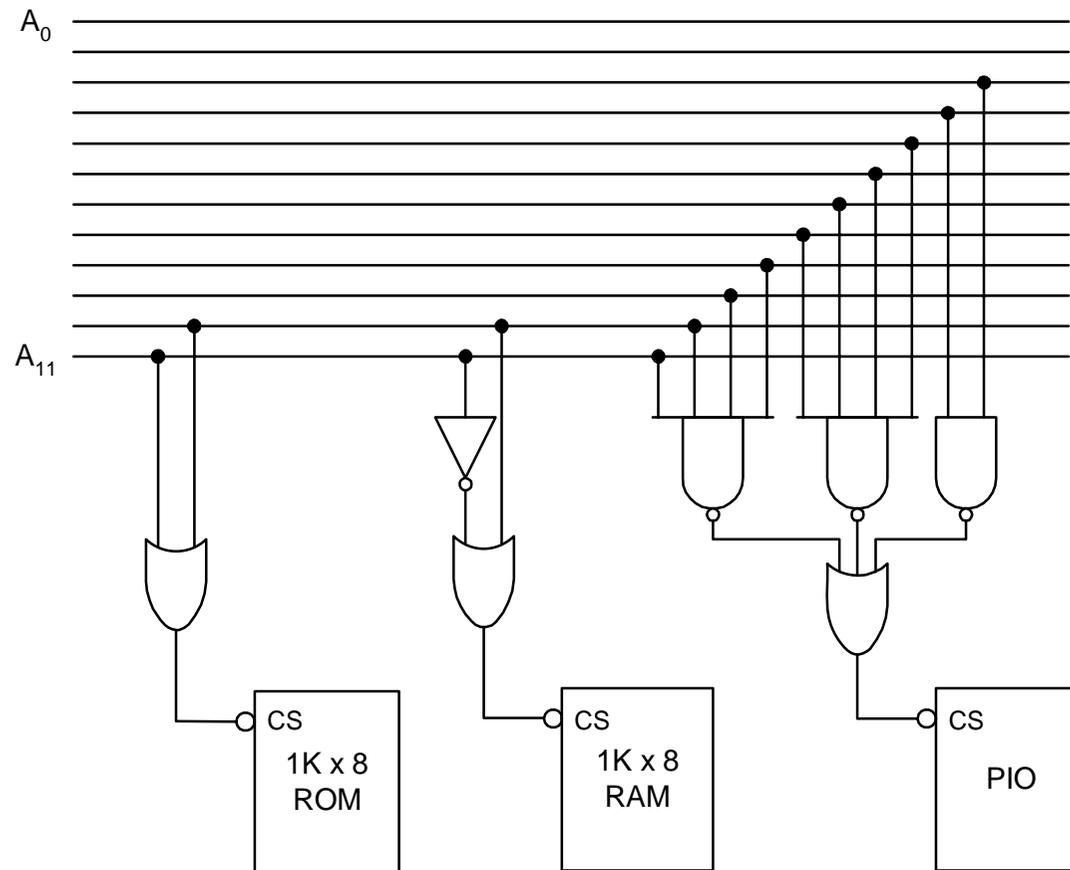
Descodificação de Endereços

- Exemplo de mapeamento do espaço de endereçamento



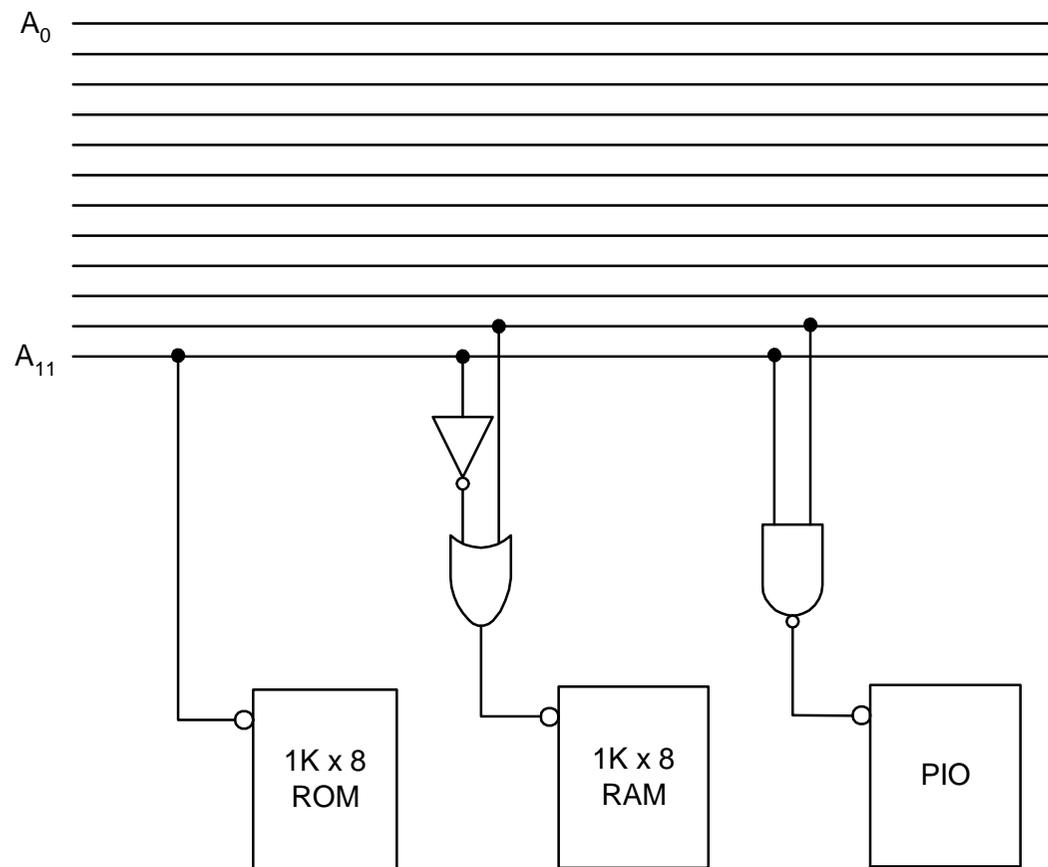
Descodificação de Endereços

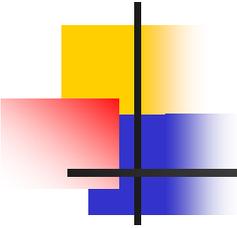
- Descodificação completa



Descodificação de Endereços

- Descodificação parcial





Descodificação de Endereços

- Descodificação completa vs. parcial
 - Completa
 - Utiliza-se mais lógica
 - Ajustada à dimensão do espaço de endereçamento de cada dispositivo
 - Parcial
 - Utiliza-se menos lógica
 - Cada dispositivo poderá ocupar uma porção do espaço de endereçamento superior ao necessário



Sistemas de Computação

Paulo Santos

Processadores

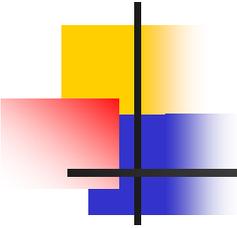


UNIÃO EUROPEIA

Fundo Social Europeu

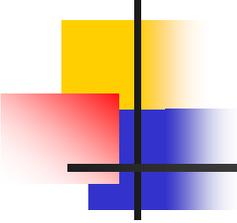
prime
Programa de Incentivos à
Modernização da Economia





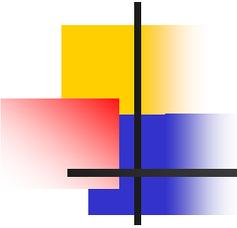
CPU

- Processador ou CPU
Central Processing Unit
 - Como já foi referido na aula anterior, é o órgão responsável pela actividade de um computador
 - Nesta aula iremos estudar a sua arquitectura (Na sua variante mais simples)



A máquina de *von Neumann*

- John Von Neumann (1903-1957)
 - Propôs uma arquitectura de computadores conhecida posteriormente como *Máquina de von Neumann*
 - Essa arquitectura é a precursora dos actuais processadores
 - *Von Neumann* trabalhou em projectos importantes:
 - ENIAC (um dos principais computadores de 1ª geração)
 - Manhattan Project (bomba atómica)
 - ...

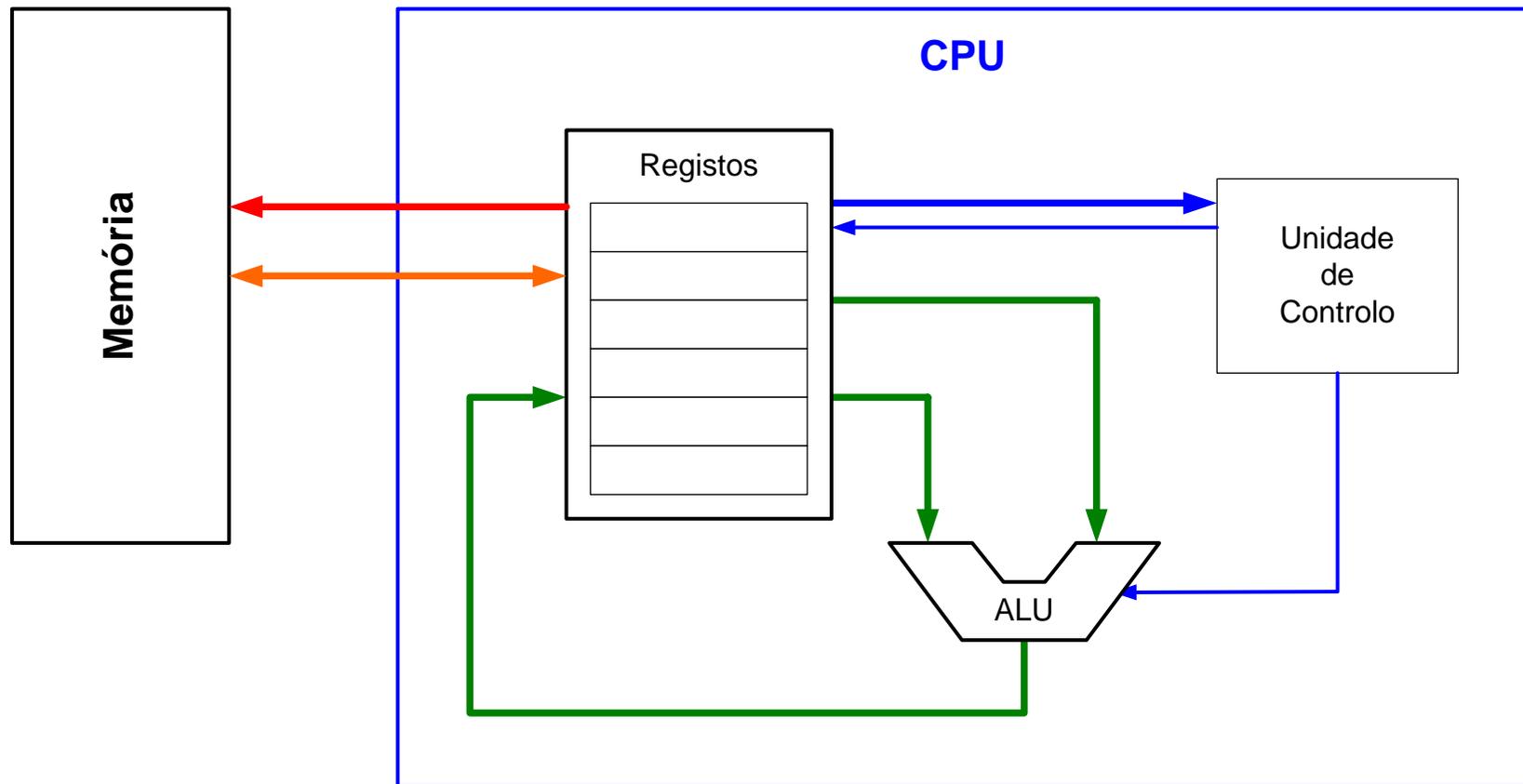


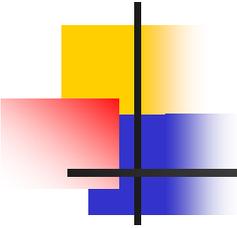
A máquina de *von Neumann*

- Funcionamento
 - CPU composto por 3 unidades principais
 - Conjunto de registos
 - ALU – Unidade aritmética-lógica
 - Unidade de controlo
 - O CPU executa um conjunto de instruções carregadas em memória

A máquina de *von Neumann*

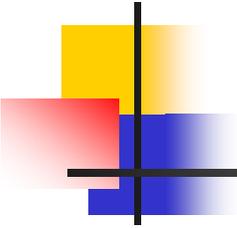
■ Arquitectura





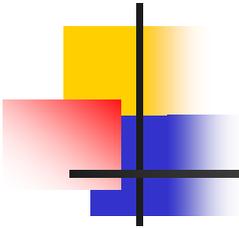
Execução de programas

- **Programa** – sequência de instruções em código-máquina obtido através de
 - Compilação (C/C++)
 - Interpretação (Java)
 - *Assembler* (Assembly)
- **Código-máquina** – linguagem que a CPU percebe
 - Muito baixo nível – ‘0’s e ‘1’ (!)
- ***Assembly*** – linguagem próxima do código-máquina, mas mais fácil para um humano perceber
 - A cada instrução *assembly* corresponde uma instrução em código-máquina
 - Baixo nível



Execução de programas

- Para executar cada instrução carregada em memória, há que:
 - Ler da memória a instrução a executar – **fetch**
 - A instrução é carregada para um registo especial do CPU
 - A **Unidade de controlo** descodifica a instrução...
 - ...e gera um conjunto de sinais que controlam os **registos** e a **ALU**, de modo a serem efectuadas as operações definidas pela instrução

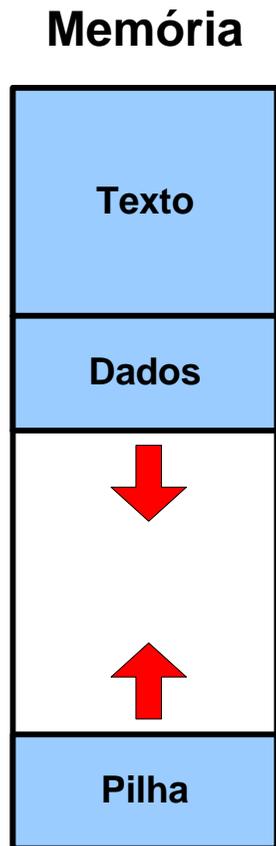


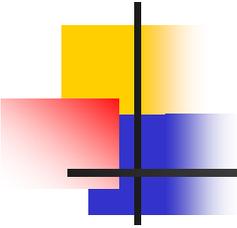
Execução de programas

- Exemplos de instruções:
 - Adicionar ao conteúdo de um registo um valor armazenado em memória (Add)
 - Copiar os dados de um registo para outro registo (Move)
 - Carregar um registo com dados armazenados na memória (Load)
 - Escrever na memória os dados guardados num registo (Store)
 - Chamar um procedimento (Call)
 - Retornar de um procedimento (Return)

Execução de programas

- Espaço de endereçamento de um programa
 - Segmento de texto
 - Contém o programa
 - Segmento de dados (Heap)
 - Variáveis globais
 - Constantes
 - Blocos de dados criados durante a execução do programa
 - Segmento da pilha (Stack)
 - Parâmetros passados a procedimentos
 - Variáveis locais (podem conter referências para dados na Heap)
 - Endereços de retorno dos procedimentos



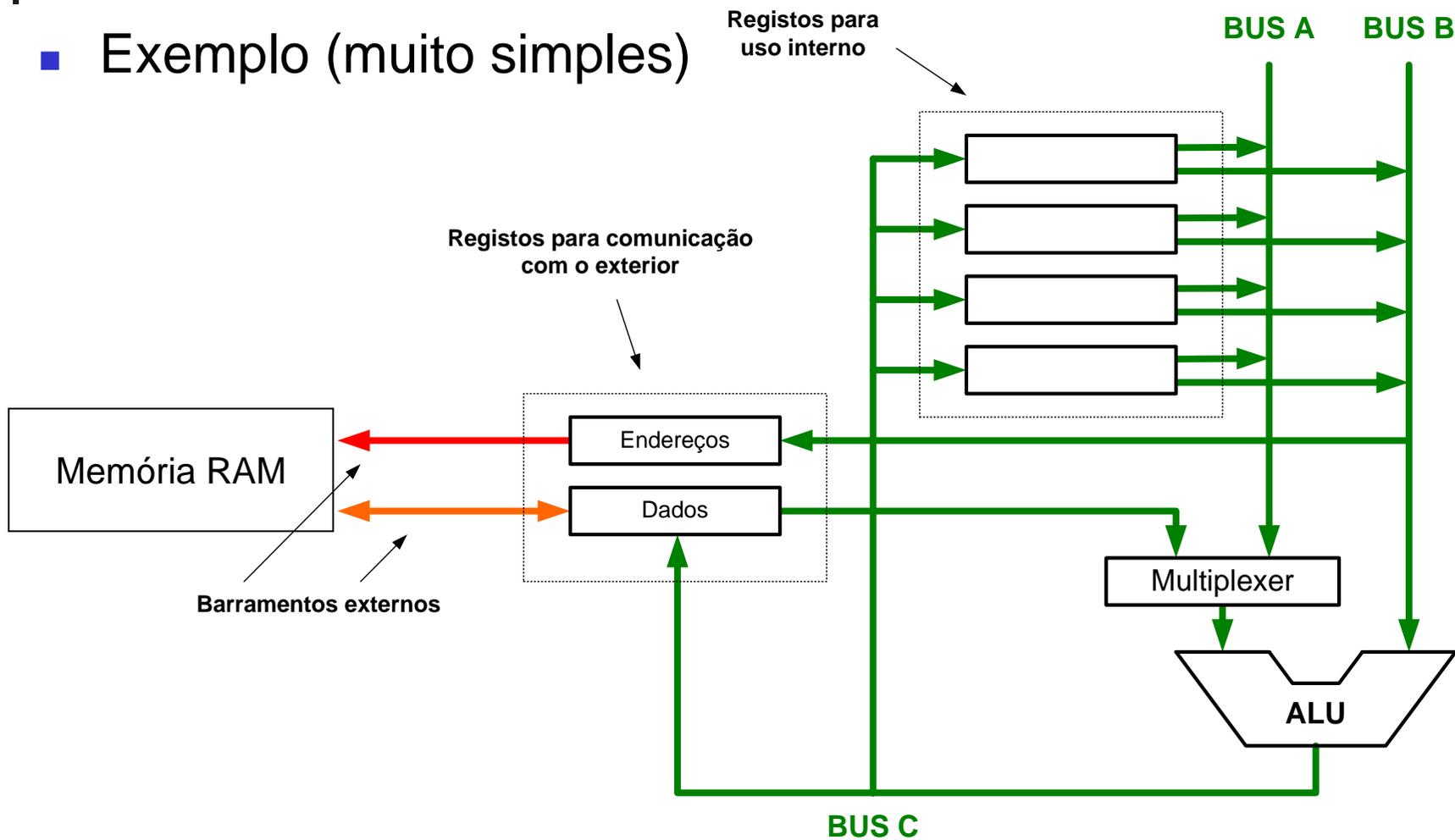


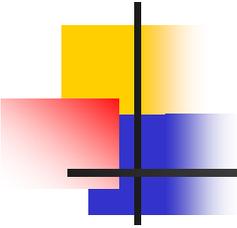
Data Path

- *Data Path* – conjunto formado por
 - Registos
 - ALU
 - Barramentos internos do CPU
- No *Data Path* circulam dados e endereços
- As operações são feitas no *Data Path* de acordo com os sinais gerados pela unidade de controlo

Data Path

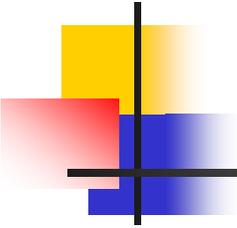
- Exemplo (muito simples)





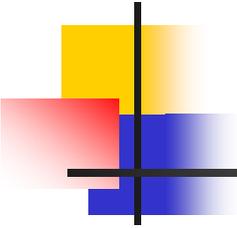
Data Path

- Registos típicos num CPU
 - Uso interno
 - PC (*Program Counter*)
 - Guarda o endereço da próxima instrução a executar
 - SP (*Stack Pointer*)
 - Guarda o endereço do topo da pilha
 - AC (*Accumulator*)
 - Guarda o resultado da última operação na ALU
 - IR (*Instruction Register*)
 - Guarda a instrução (código-máquina) a executar
 - Registos de uso geral



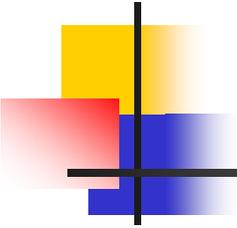
Data Path

- Registos
 - Comunicação com o exterior
 - MAR (*Memory Address Register*)
 - Endereço da posição de memória a ler ou escrever
 - MDR (*Memory Data Register*)
 - Dados lidos ou a escrever na memória



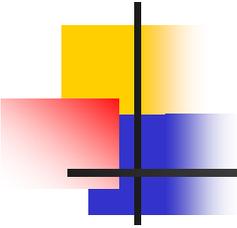
Data Path

- ALU – Unidade Aritmética-Lógica
 - Operações Aritméticas
 - $A + B$
 - $A \times B$
 - ...
 - Operações lógicas
 - $A \wedge B$
 - $\sim A$
 - ...
 - Testes
 - $A = 0 ?$
 - $A > 0 ?$
 - ...



Unidade de controlo

- A unidade de controlo tem a função de gerar sinais de controlo de acordo com a instrução a executar
 - Sinais de controlo para os registos
 - Sinais de controlo para a ALU
- As suas entradas consistem
 - Instrução a executar
 - Sinais resultantes dos testes efectuados na ALU



Unidade de controlo

- Modelos

- Unidade de controlo **uni-ciclo**

- A unidade de controlo é um circuito combinatório
- Exige que o programa e os dados estejam em memórias separadas (pois não se pode aceder à mesma 2 vezes no mesmo ciclo)

- Unidade de controlo **multi-ciclo**

- A unidade de controlo é um circuito sequencial
- Contem uma ROM com sequências de micro-instruções
- Para efectuar uma instrução do programa (macro-instrução) são necessárias várias micro-instruções
 - Consequentemente, vários ciclos de relógio
- É o modelo mais comum...