Álgebra de Boole

{Bool_r.doc}

Preâmbulo: **Álgebra de Boole**,
de *dois elementos* {0 e 1} e *três operações* {*soma/OR*: **+**, *produto/AND*: • e *negação/NOT*: ¯ }

P1: ***Operações*** da Álgebra de Boole:

OR, **+**	**0**	**1**
0	0	1
1	1	1

NOT	**0**	**1**
	1	0

AND, •	**0**	**1**
0	0	0
1	0	1

Por extenso:
- o AND produz '**0**' se, e só se, *ao menos um* operando for '**0**' – o mesmo é dizer que produz '**1**' se, e só se, *todos* os seus operandos forem '**1**'
- o OR produz '**1**' se, e só se, *ao menos um* operando for '**1**' – o mesmo é dizer que produz '**0** se, e só se, *todos* os seus operandos forem '**0**'

P2: ***Propriedades: Axiomas, Teoremas***...

Axiomas:	Comutatividade	$x+y=y+x$	$x \bullet y = y \bullet x$
	Distributividade	$x+(y \bullet z)=(x+y) \bullet (x+z)$	$x \bullet (y+z)=(x \bullet y)+(x \bullet z)$
	Elemento neutro	$x+0=x$	$x \bullet 1 = x$
	Complementaridade	$x+\bar{x}=1$	$x \bullet \bar{x} = 0$
Teoremas:	Idempotência	$x+x=x$	$x \bullet x = x$
	Elemento absorvente	$1+x=1$	$0 \bullet x = 0$
	Associatividade	$x+(y+z)=(x+y)+z$	$x \bullet (y \bullet z)=(x \bullet y) \bullet z$
	Involução (dupla negação)	$\bar{\bar{0}}=0$	$\bar{\bar{1}}=1$
	Absorção	$x+xy=x$	$x(x+y)=x$
	Redundância	$x+\bar{x}y=x+y$	$x(\bar{x}+y)=x+y$
	Adjacência	$xy+x\bar{y}=x$	$(x+y)(x+\bar{y})=x$
	Leis de De Morgan	$\overline{x \bullet y}=\bar{x}+\bar{y}$	$\overline{x+y}=\bar{x} \bullet \bar{y}$
	Dualidade	De uma coluna chega-se à outra por troca: + ↔ • e 0 ↔ 1	

Representação e Simplificação de Funções Booleanas

1. [10E2.2, 10T3.2] Simplifique algebricamente a expressão da função

$$f(A,B,C,D)=\bar{C}\,\bar{D}+B+\bar{B}\,C\,\bar{D}+A\,B\,C\ .$$

R: $f(A,B,C,D)=\bar{C}\,\bar{D}+\bar{B}\,C\,\bar{D}+B$ (Absorção: $x+xy=x$, no caso, $B+A\,B\,C=B$)

$=\bar{C}\,\bar{D}+C\,\bar{D}+B$ (Simplificação: $x+\bar{x}y=x+y$, no caso, $B+\bar{B}C\bar{D}=B+C\,\bar{D}$

$=\bar{D}+B$ (Distributividade – $\bar{D}$ em evidência -, e Complementaridade: $\bar{C}+C=1$)

Confirmação: um outro desenvolvimento, não tão rápido, será:

$f(A,B,C,D)=\bar{D}(\bar{C}+\bar{B}\,C)+B(1+A\,C)$

$=\bar{D}(\bar{C}+\bar{B})+B$

$=\bar{C}\,\bar{D}+\bar{B}\,\bar{D}+B$

$= \overline{C}\,\overline{D} + \overline{D} + B$

$= \overline{D} + B$

Pormenorizando todos os passos dados na confirmação, um de cada vez:

$f(A,B,C,D) = \overline{C}\,\overline{D} + \overline{B}\,C\,\overline{D} + B + A\,B\,C$ (Comutatividade: Reordenaram-se os termos do OR)

$= \overline{D}\,\overline{C} + \overline{D}\,C\,\overline{B} + B + B\,C\,A$ (Comutatividade: Reordenaram-se os termos dos ANDs)

$= \left[\overline{D}\,\overline{C} + \overline{D}\,C\,\overline{B}\right] + \left[B + B\,C\,A\right]$ (Associatividade)

$= \overline{D}\,(\overline{C} + C\,\overline{B}) + B\,(1 + C\,A)$ (Distributividade: $\overline{D}$ e B em evidência, e 1 é neutro do AND)

$= \overline{D}\,(\overline{C} + C\,\overline{B}) + B\,(1)$ (Absorção: 1 é absorvente da adição)

$= \overline{D}\,(\overline{C} + C\,\overline{B}) + B$ (1 é o neutro do AND)

$= \overline{D}\left[(\overline{C} + C)\,(\overline{C} + \overline{B})\right] + B$ (Distributividade)

$= \overline{D}\left[(1)\,(\overline{C} + \overline{B})\right] + B$ (Complementaridade)

$= \overline{D}\,(\overline{C} + \overline{B}) + B$ (1 é o neutro do AND)

$= \overline{D}\,\overline{C} + \overline{D}\,\overline{B} + B$ (Distributividade)

$= \overline{D}\,\overline{C} + \left(\overline{D} + B\right)\left(\overline{B} + B\right)$ (Distributividade)

$= \overline{D}\,\overline{C} + \left(\overline{D} + B\right)(1)$ (Complementaridade)

$= \overline{D}\,\overline{C} + \overline{D} + B$ (1 é o neutro do AND)

$= \overline{D}\left(\overline{C} + 1\right) + B$ (Distributividade: $\overline{D}$ em evidência, e 1 é neutro do AND)

$= \overline{D}\,(1) + B$ (Absorção: 1 é absorvente da adição)

$= \overline{D} + B$ (1 é o neutro do AND)

2ª Confirmação: nem sempre acontece, como foi este o caso, haver dois caminhos diferentes para chegar ao resultado final… Mas resta *em geral* a possibilidade de aplicar o método de Karnaugh, para confirmar que o resultado está correcto:

1. A primeira questão é: qual será a *tabela de verdade* da função, $f(A,B,C,D) = \overline{C}\,\overline{D} + B + \overline{B}\,C\,\overline{D} + A\,B\,C$?

Ela é um OR – uma operação que, *vidé Preâmbulo-2*, produz '**1**' *se, e só se, ao menos um dos seus termos for '1'*! A função será então '**1**' se (e só se) for '**1**' *ao menos um dos seus termos*: $\overline{C}\,\overline{D}$ ***ou*** B ***ou*** $\overline{B}\,C\,\overline{D}$ ***ou*** $A\,B\,C$.

Afora B, os restantes termos são AND – uma operação que, *vidé Preâmbulo-2*, produz '**1**' *se, e só se, todos os seus termos forem '1'*! Considerando termo a termo,

$\overline{C}\,\overline{D}$ será '**1**' se, e só se, $\overline{C}$ ***e*** $\overline{D}$ forem, *ambos,* '**1**' – o mesmo é dizer se $C = 0$ ***e***, *simultâneamente*, $D = 0$;

$\overline{B}\,C\,\overline{D}$ será '**1**' se, e só se, $\overline{B}$ ***e*** C ***e*** $\overline{D}$ forem, *todos*, '**1**' – ou seja: se, *simultâneamente*, $B = 0$ ***e*** $C = 1$ ***e*** $D = 0$;

$A\,B\,C$ será '**1**' se, e só se, A ***e*** B ***e*** C forem, *todos*, '**1**': se, *simultâneamente*, $A = 1$ ***e*** $B = 1$ ***e*** $C = 1$.

A construção da tabela de verdade, já sob a forma de quadro de Karnaugh, ir-se-á desenrolando da seguinte maneira:

1ª etapa: desenho da configuração da tabela de Karnaugh (*vidé* fig Karnaugh01.1, onde, para suportar referências adiante, as células surgem *rotuladas* {0, …, 15})

		0	0	1	1	B
		0	1	1	0	A
0	0	0	1	3	2	
0	1	4	5	7	6	
1	1	12	13	15	14	
1	0	8	9	11	10	
D	C					Karnaugh01.1

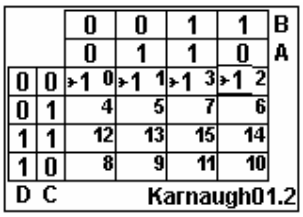
2ª etapa: inscrição dos '**1**' produzidos pela função:

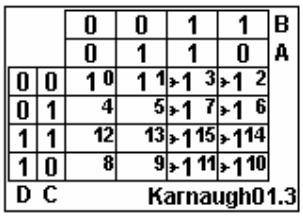
- inscrição dos '**1**' produzidos por $\overline{C}\,\overline{D}$, isto é, na linha em que $D = 0$ e $C = 0$; na prática (*vidé* Karnaugh01.2), inscrição de '**1**'s nas células {0, 1, 2 e 3}

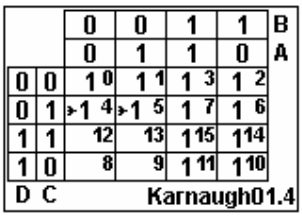
- inscrição dos '**1**' produzidos por B, isto é, nas colunas em que $B = 1$; na prática (*vidé* Karnaugh01.3), inscrição de '**1**'s nas células {2, 3, 6, 7, 10, 11, 14 e 15}

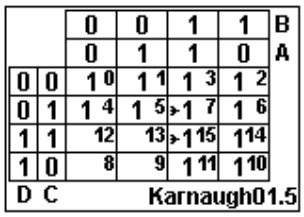
- inscrição dos '**1**' produzidos por $\overline{B}\,C\,\overline{D}$, isto é, nas células em que, $D = 0$, $C = 1$ e $B = 0$; na prática (*vidé* Karnaugh01.4), inscrição de '**1**'s nas células {4, 5}

- inscrição dos '**1**' produzidos por $A\,B\,C$, isto é, nas células em que $C=1$, $B=1$ e $A=1$; na prática (*vidé* Karnaugh01.5), inscrição de '**1**'s nas células {7, 15}

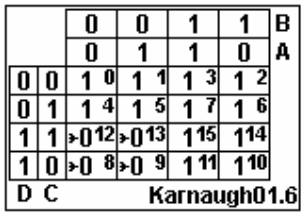


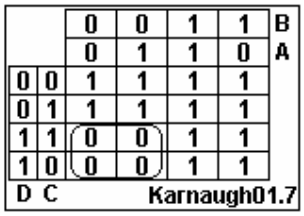






3ª etapa: inscrição dos '**0**' nas restantes células; na prática (*vidé* Karnaugh01.6), inscrição de '**0**'s nas células {8, 9, 12 e 13}





4ª etapa: simplificação usando o método de Karnaugh; Existindo duas alternativas (agrupamento de '**0**'s ou de '**1**'s, no caso presente, é tentador usar a segunda, pois somente há 4 '**0**'s (*vidé* Karnaugh01.7).

A função tem '**0**'s em só quatro células {8, 9, 12 e 13}– e, nelas, $B=0$ e $D=1$; pelo que

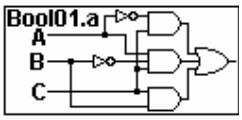
$$f(A,B,C,D)=\overline{D}+B$$

Notas a propósito da asserção "resta ***em geral*** a possibilidade de aplicar o método de Karnaugh…":

- como ocorre com $(D+\overline{B}\,(\overline{C}+A)+C\,A)\,(\overline{D}+B\,(C+\overline{A})+\overline{C}\,\overline{A})$, pode haver mais que uma forma de simplificar: pode, *algebricamente*, chegar-se a uma delas, e, pelo *método de Karnaugh*, chegar-se à outra…

- como ocorre com $XY+\overline{Y}\,\overline{Z}+WX\overline{Z}$, pode acontecer que, *algebricamente*, não se saiba "por onde pegar nisto" – e todavia o *método de Karnaugh* mostre que é mesmo possível simplificar a expressão: verifique o leitor isso mesmo – e descubra o que fazer para chegar à forma mais simples por meios algébricos…

2. [10E1.1] Considere o circuito da figura Bool01.a:



2. 1. Indique qual a expressão lógica de F.
2. 2. Encontre a expressão lógica que implementa o circuito da figura anterior utilizando unicamente portas NAND e NOT.
2. 3. Os circuitos concretizados utilizando *gates* básicas directamente a partir das expressões anteriores necessitam de pelo menos 3 circuitos integrados. Desenhe o logigrama de um circuito que implemente a mesma função mas que utilize no máximo 1 circuito integrado.

R1: $F(C,B,A)=C\,\overline{A}+C\,\overline{B}\,A+C\,B$

(Segue-se a filosofia: num *bus* horizontal, fica em cima a variável de *menor* peso, '**A**', numa expressão algébrica é a *última*)

R2: $\overline{\overline{C}\,\overline{A}}\;\overline{\overline{C}\,\overline{B}\,A}\;\overline{C\,B}$ (basta aplicar as leis de DeMorgan a R1, *vidé* Preâmbulo P2)

R3: Pretendendo-se um logigrama *mais simples*, convirá *simplificar* f:

F = $C(\overline{A}+\overline{B}\,A+B)$ (Distributividade: C em evidência)

= $C(\overline{A}+\overline{B}+B)$ (Redundância: $x+\overline{x}y=x+y$, *vidé* Preâmbulo P2)

= $C(\overline{A}+1)$ (Associatividade e Complementaridade, *vidé* Preâmbulo P2)

= $C(1)$ (1 é Elemento absorvente da adição, *vidé* Preâmbulo P2)

= C (1 é Elemento neutro do produto, *vidé* Preâmbulo P2)

Um circuito simples com, no máximo, um circuito integrado poderá então ser: C ————F

Entretanto, convém prestar muita atenção ao reparo abaixo!!!

Reparos:

1) Num primeiro relance, o logigrama dado oferece F como um *OR de três entradas*, F = $(...)+(...)+(...)$.

Olhando de perto, cada uma dessas entradas é um AND, F = $((...)\bullet(...))+((...)\bullet(...)\bullet(...))+((...)\bullet(...))$,

ou, porquanto *o* AND *é prioritário relativamente ao* OR, F = $(\ )\bullet(\ )+(\ )\bullet(\ )\bullet(\ )+(\ )\bullet(\ )$.

As entradas desses ANDs são as próprias variáveis de entrada {C,B,A}, eventualmente complementadas:

$F(C,B,A) = (C)\bullet(\overline{A})+(C)(\overline{B})(A)+(C)(B)$ - ou, mais simplesmente, a solução reportada acima.

2) A *dupla negação* não altera o valor de uma função (*vidé* Preâmbulo P2/involução), F = $\overline{C\,\overline{A}+C\overline{B}A+CB}$; aplicando imediatamente a esta expressão as leis de Morgan (*vidé* Preâmbulo P2), deduz-se a solução reportada em R2; nela, usam-se:

- portas NOT, para obter $\overline{A}$ e $\overline{B}$

- portas NAND: de 2-entradas, para obter $\overline{C\,\overline{A}}$ e $\overline{C\overline{B}}$, e de 3-entradas, para obter $\overline{C\overline{B}A}$ e $\overline{\overline{C\,\overline{A}}\;\overline{C\overline{B}A}\;\overline{CB}}$

3) O método seguido acima, para lograr um logigrama com, *no máximo, um circuito integrado*, teve a favor a extrema simplicidade a que a função se reduziu! Não é expectável que as coisas corram sempre assim tão simples – pelo que convém um *método geral* – válido (ao menos em teoria) para *qualquer* função, por *muito complexa* que seja! Primeiro que tudo, deduz-se a tabela de verdade de **f**: Conforme R1, **f** exprime-se como **OR** das expressões $C\,\overline{A}$, $C\overline{B}A$ e CB; conclui-se que **f** se volve em '**1**' se, e só se, *ao menos* uma delas for '**1**', ou seja: se {C=1 ***e*** A=0} **ou** {C=1 ***e*** B=0 ***e*** A=1} **ou** { C=1 ***e*** B=1}:

	C	**B**	**A**				
0ª	0	0	0	0	0	0	**0**
1ª	0	0	1	0	0	0	**0**
2ª	0	1	0	0	0	0	**0**
3ª	**0**	**1**	**1**	0	0	0	**0**
4ª	1	0	0	**1**	0	0	**1**
5ª	1	0	1	0	**1**	0	**1**
6ª	1	1	0	**1**	0	**1**	**1**
7ª	1	1	1	0	0	**1**	**1**
				$C\,\overline{A}$	$C\overline{B}A$	CB	**f**

ou

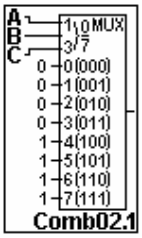
	0ª	1ª	2ª	3ª	4ª	5ª	6ª	7ª	
	0	0	0	**0**	1	1	1	1	**C**
	0	0	1	**1**	0	0	1	1	**B**
	0	1	0	**1**	0	1	0	1	**A**
$C\,\overline{A}$	0	0	0	0	**1**	0	**1**	0	
$C\overline{B}A$	0	0	0	0	0	**1**	0	0	
CB	0	0	0	0	0	0	**1**	**1**	
f	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	

(A tabela de verdade é apresentada sob duas disposições *equivalentes*:

- uma, à esquerda, onde as variáveis se distribuem por *colunas* – quiçá aquela a que o leitor estará mais habituado…
- e outra, à direita, em que as variáveis se distribuem por *linhas* – quiçá a mais usada quando o número delas crescer…

Na tabela da esquerda, a variável de *menor-peso*, seja '**A**', fica à direita, e a de *maior-peso*, seja '**C**', fica à esquerda; com isso, as linhas abaixo das variáveis são, de cima para baixo, as representações binárias de '0', '1', '2', …, até '7'; o *número-designativo* de **f** – que está realçado: 00001111 - lê-se *de cima para baixo*.

Na tabela da direita, a variável de *menor-peso*, '**A**' fica em baixo, e '**C**' fica em cima; com isso, o *número-designativo* de **f** lê-se *da esquerda para a direita*… Esta orientação é compatível com quadros de Karnaugh – mas *inversa daquela seguida em logigramas* (*vidé* por ex. o símbolo do Multiplexer na figura Comb02.1) em que as entradas de *selecção* de *menor-peso* estão em cima e as de *maior-peso* estão em baixo)



A partir desta tabela, e visando dar-lhe corpo com, *no máximo, um circuito integrado*, apresenta-se em Comb02.1 uma solução usando um Multiplexer "3:8", i.e.: com 3 entradas-de-selecção/8 entradas-de-dados:

- aplica-se, às entradas de *selecção*, as variáveis {A,B,C} - **ordenadas A→'1'**, **B→'2'** e **C→'3'** (i.e.: 'bit de *menor* peso' → 'entrada de selecção de *menor* peso');
- aplica-se, às entradas de *dados*, devidamente *ordenada*, o próprio *número-designativo* de F.

Repare-se: quando {C=**0,** B=**1**, A=**1**} (por exemplo) – ou seja, quando as entradas de *selecção* {3,2,1} assumem os valores {**011**}, a saída **f** irá assumir o valor que então se apresentar na entrada de *dados* '**3**' (repare-se: **011** é a representação binária de **3**). Bastará conhecer o valor que a tabela de verdade assume para a combinação {C=**0,** B=**1**, A=**1**} – que no caso é '**0**' –, e aplicar esse valor nessa entrada '**3**'… Raciocínio similar vale para as demais combinações das variáveis de entrada…

(Atenção: o símbolo *correcto* do MUX não envolve os *parêntesis* – como seja (000) a seguir ao rótulo da entrada de dados '0': a sua inclusão é uma anotação do autor, quiçá facilite a vida a quem se está iniciando ao mapeamento decimal↔binário)

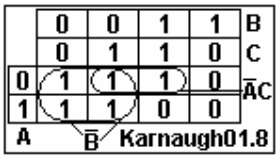
3. [10T1.2] Simplifique algebricamente a expressão da função $f(A,B,C) = A\overline{B} + \overline{A}\overline{B} + \overline{A}BC$.

   R: f = $(A + \overline{A})\,\overline{B} + \overline{A}\,B\,C$ (Distributividade: $\overline{B}$ em evidência)

   = $\overline{B} + \overline{A}\,B\,C$ (Complementaridade, *vidé* Preâmbulo P2, e 1 é neutro do AND)

   = $\overline{B} + \overline{A}\,C$ (Redundância: $x + \overline{x}y = x + y$, *vidé* Preâmbulo P2)

   Confirmação (pelo método de Karnaugh): em Karnaugh01.8



Método de Karnaugh

{Karnaugh_r.doc}

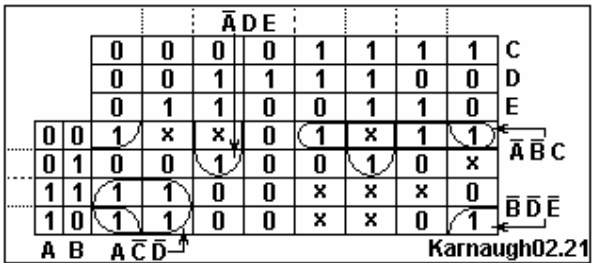
1. [10E2.3, 10T3.3] Considere a seguinte função booleana, em que A é a variável de maior peso:

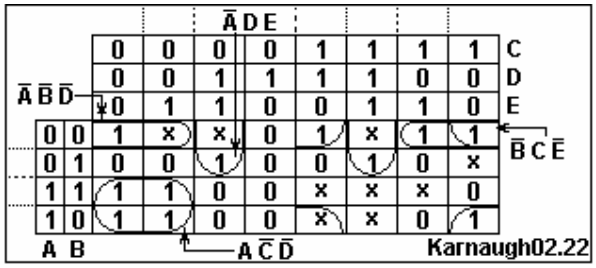
$$f(A,B,C,D,E) = \sum m(0,4,5,6,11,15,16,17,20,24,25) + \sum m_d(1,3,7,12,22,23,29,30,31)$$

1. 1. Obtenha a expressão mínima na forma disjuntiva (soma de produtos), utilizando o método de Karnaugh.
1. 2. Identifique os implicantes primos essenciais da função:

R.1: $f(A,B,C,D,E) = \overline{A}DE + A\overline{C}\,\overline{D} + \overline{B}\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}C$ (*vidé* Karnaugh02.21);

$f(A,B,C,D,E) = \overline{A}DE + A\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + \overline{B}C\overline{E}$ (*vidé* Karnaugh02.22).





R.2: $\overline{A}DE$ e $A\overline{C}\,\overline{D}$, em virtude de, respectivamente, **m11** e **m24**.

Repare-se: a determinação da expressão mínima na *forma disjuntiva* para uma função booleana usando o método de Karnaugh envolve as *seguintes etapas*:

a) Configurar o quadro de Karnaugh para as variáveis concretas da função;
b) Preencher as células do quadro com '**1**'s, '**0**'s, e '**x**'s, tendo em conta o que se sabe acerca da função;
c) Procurar os implicantes primos *essenciais*; hão-de necessariamente fazer parte do resultado final;
d) Procurar o menor número de implicantes primos (entre os *não essenciais*) que cubra os '**1**'s que falta contemplar.
e) Escrever a expressão mínima da função.

3-1-a) As variáveis da função são 5: {A, B, C, D, E}; isso significa que elas podem assumir 2^5=32 combinações distintas. Convém rotulá-las {0, 1, …, 31}. Para tal, e sabido que **A** é a de *maior peso*, começa-se por elaborar *mentalmente* a correspondência adiante: a cada variável é atribuído um peso - que é exactamente uma *potência-de-2* -, **A** ficando com o maior:

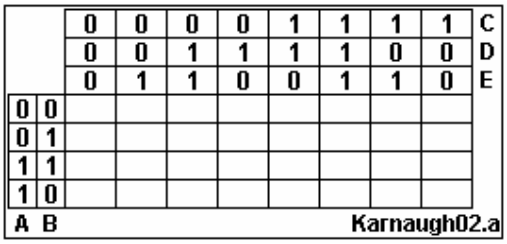
Variável	A	B	C	D	E
Pesos:	**16**	**8**	**4**	**2**	**1**

Isto sendo pacífico, qual será o rótulo a atribuir à combinação {A=**0**, B=**1**, C=**0**, D=**1**, E=**1**} (por exemplo)? Resposta: somem-se os *pesos* (potências-de-2) associados às variáveis que nela valem '**1**' – no caso, as variáveis {B, D e E}; virá 8+2+1=11: subentende-se que aquela combinação fica rotulada de combinação "11ª"… Idem para as demais 31 combinações…

Curiosidade: *qual o valor que a função assume para essa combinação '11ª'?* Resposta: '**1**': a expressão da função envolve um primeiro termo, $\sum m$, - que lista precisamente as combinações para as quais a função assume o valor '**1**'! De facto, e conforme ao elenco desse termo, ela vale '**1**' também para as combinações '**0**', '**4**', '**5**', '**6**', '**15**', '**16**', '**17**', '**20**', '**24**' e '**25**'!

E quanto ao segundo termo na expressão da função, $\sum m_d$? Resposta: ele lista as combinações para as quais a função *não está definida*: são combinações que por alguma razão "*não ocorrem na vida real*", "*não há que gastar tempo com elas*".

Após estes preliminares, é hora de configurar o quadro de Karnaugh. No caso entre mãos, ele suporta-se em uma *tabela* (*vidé* Karnaugh02.a), com 4 linhas e 8 colunas, resultando um total de 32 células - de início "a branco". Elas destinam-se a conter os *valores admissíveis da função*, a cada uma corresponde uma *combinação única, particular*, de todas as possíveis que {A, B, C, D e E} podem assumir.



À *esquerda* da tabela, dispõem-se duas colunas: linha após linha, exibem todas as 2^2=4 combinações possíveis do par {**A, B**}; em *cima*, dispõem-se três linhas: coluna após coluna, exibem todas as 2^3=8 combinações possíveis do trio {**C, D, E**}.

1. dado o modo como é especificada a função, via $\sum m$, a rotulagem de linhas e colunas *não é arbitrária*: as variáveis da função são repartidas em dois conjuntos disjuntos - {**A, B**} e {**C, D, E**} – para rotular respectivamente as colunas e as linhas;

2. as sequências de combinações das variáveis também *não são arbitrárias*: estão dispostas segundo o assim designado *código binário reflectido*; concretamente, e varrendo *mentalmente* as linhas e colunas *a partir da área central da tabela*,

- considerando a 1ª coluna, {**B**}, de cima para baixo, e a 1ª linha, {**E**}, da esquerda para a direita, a sequência de '**0**'s e '**1**'s é a seguinte: escreve-se '**01**', depois *reflecte-se*, isto é escreve-se '**10**'; volta-se a reflectir, etc. – resultando 01 10 01 10…

- considerando a 2ª coluna, {**A**}, e a 2ª linha, {**D**}, a sequência de '**0**'s e '**1**'s é a seguinte: escreve-se '**0011**', depois *reflecte-se*, isto é escreve-se '**1100**'; volta-se a reflectir, etc. – resultando 0011 1100…

- considerando a 3ª linha, {**C**}, a sequência de '**0**'s e '**1**'s é: escreve-se '**00001111**' (se houvesse mais que 5 variáveis, depois reflectir-se-ia, isto é escrever-se-ia '**11110000**'; voltar-se-ia a reflectir, etc. – resultando 0000 1111 1111 0000…

É hora de apor às células da tabela os rótulos {0, …, 31}. É uma praxis trivial, *vidé* fig Karnaugh02.b:

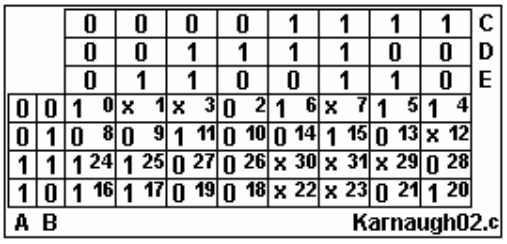
- *mentalmente*, sob as colunas {**A, B**} e à direita das linhas {**C, D, E**}, apõem-se os pesos das respectivas variáveis (*vidé* correspondência acima: **A→16**, **B→8**, **C→4**, **D→2**, **E→1**);

- *mentalmente*, determinam-se os pesos de cada *linha e coluna da tabela*; nomeadamente, às *linhas* da tabela, associam-se os pesos {**0**, **8**, **24** e **16**} (Para determinar o peso de, por ex., a 2ª linha, repare no par '**01**', à esquerda – e some os *pesos* das colunas (**B**) em que ele é '**1**': 8; pode inscrevê-lo na 2ª linha, 1ª coluna); e às *colunas* da tabela associam-se os pesos {**0**, **1**, **3**, **2**, **6**, **7**, **5** e **4**} (Para determinar o peso de, por ex., a 3ª coluna a contar da esquerda, repare no trio '**011**', em cima – e some os *pesos* das linhas (**D**, **E**) em que ele é '**1**': 2+1=3; pode inscrevê-lo na 1ª linha, 3ª coluna).

- *mentalmente*, determinam-se os pesos de todas as *células da tabela* (Para determinar o peso de, por ex., a célula criada pela intersecção da 2ª linha com a 3ª coluna, repare nos respectivos pesos, 8 e 3, e some-os: 8+3=11 – e inscreva-o na célula).

> Notas:
> - Existem outras maneiras de configurar o quadro de Karnaugh para 5 variáveis. Por exemplo, elas poder-se-iam repartir em dois outros sub-conjuntos, seja {A, B e C} e {D e E} – o resultado sendo uma tabela com 8 *linhas* e 4 *colunas*…
> - De outra parte, nem todas as funções remetem para 5 variáveis: poder-se-ia, por exemplo, estar frente a uma função com, apenas, 4 variáveis; claramente, agora haveria que as repartir em outros sub-conjuntos, como seja {A e B} e {C e D}.
> Em qualquer caso, tudo o resto - rotular linhas, colunas e células - seria análogo…

3-1-b) É hora de preencher as células do quadro com '**1**'s, '**0**'s, e '**x**'s



- inscreve-se '**1**' nas células rotuladas com os números listados no termo $\sum m$ na expressão da função: 0, 4, 5, 6, 11, etc.;

- inscreve-se '**x**' nas células rotuladas com os números listados no termo $\sum m_d$ na expressão da função: 1, 3, 7, 12, etc.;

- inscreve-se '**0**' nas restantes células (*vidé* Karnaugh 02.c).

Repare-se, em particular, que, conforme ao antes entrevisto, o valor que a função assume para a célula associada à combinação {A=**0**, B=**1**, C=**0**, D=**1**, E=**1**} – rotulada de "11ª" - é '**1**'…

3-1-c) É hora de procurar os *implicantes primos essenciais*…

Antes, porém, abra-se um parêntesis, para compreender os cuidados especiais tidos na construção do quadro… Considerem-se, por exemplo, as células {**4**} e {**5**}, *ambas* contendo '**1**'. O que elas *dizem* é que a função assume '**1**' quando as variáveis assumem os valores {A=**0**, B=**0**, C=**1**, D=**0**, E=**0**} (célula 4) **ou** {A=**0**, B=**0**, C=**1**, D=**0**, E=**1**} (célula 5); depreende-se que a função se pode escrever como OR de vários termos de que (antes de ser simplificada) fazem parte $\overline{A}\,\overline{B}C\overline{D}\,\overline{E}$ e $\overline{A}\,\overline{B}C\overline{D}E$:

$$f(A,B,C,D,E)=\ldots+\overline{A}\,\overline{B}C\overline{D}\,\overline{E}+\overline{A}\,\overline{B}C\overline{D}E+\ldots$$

Em virtude do axioma da Distributividade, pode pôr-se $\overline{A}\,\overline{B}C\overline{D}$ *em evidência*, resultando

$$f(A,B,C,D,E)=\ldots+\overline{A}\,\overline{B}C\overline{D}\left(\overline{E}+E\right)+\ldots$$

e por conseguinte, por Complementaridade ($x+\overline{x}=1$), e pois que 1 é o neutro do AND ($x\bullet 1=x$):

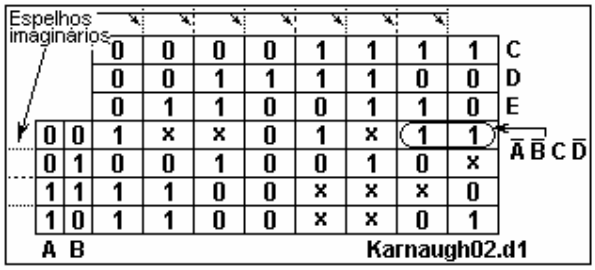
$$f(A,B,C,D,E)=\ldots+\overline{A}\,\overline{B}C\overline{D}+\ldots$$

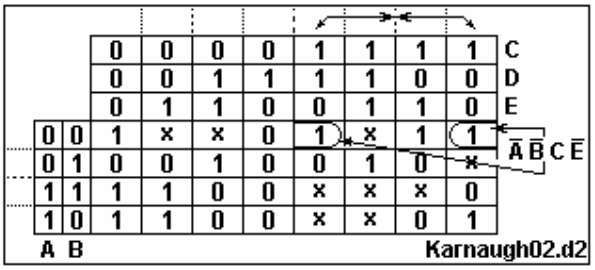
A função ficou mais simples! Atente-se: ficou mais simples porque os dois termos que se aglutinaram, $\overline{A}\,\overline{B}C\overline{D}\,\overline{E}$ e $\overline{A}\,\overline{B}C\overline{D}E$, *só diferem em* ***uma*** *variável*, **E**. Isso permitiu pôr $\overline{A}\,\overline{B}C\overline{D}$ em evidência – e eliminar essa variável **E**.

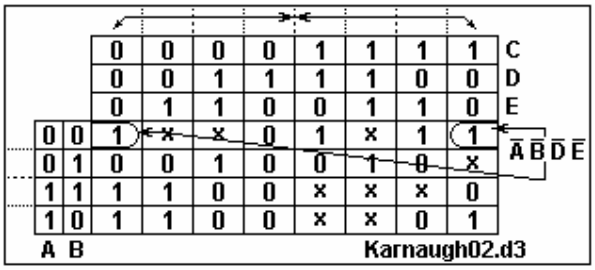
Que tem isto a ver com o quadro de Karnaugh? O seguinte: as células em causa são *adjacentes* uma à outra, *vidé* Karnaugh02.d1. De facto, para *todas as células fisicamente adjacentes*, as respectivas combinações de variáveis diferem em, só, *uma* delas. E tal como a adjacência das células {4} e {5} habilitou a uma simplificação, assim os demais pares de células adjacentes permitem outras tantas simplificações: pense o leitor nos pares de células {24, 25}, {16, 17}, {16, 24} e {17, 25}.

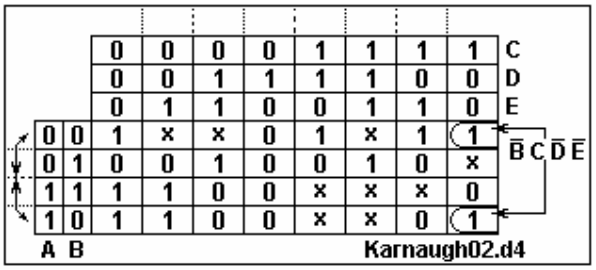
Não são apenas as células *fisicamente* adjacentes que *diferem em apenas uma variável*: o quadro de Karnaugh está pejado de muitos mais pares de células com tal propriedade: à *esquerda* e por *cima* do quadro em Karnaugh02.d1 traçaram-se alguns *espelhos imaginários – e células equidistantes deles correspondem a combinações de variáveis que só diferem numa variável*.

Restringindo a casos que envolvam a célula {4} ($\overline{A}\,\overline{B}C\overline{D}\,\overline{E}$), Karnaugh02.d1-d4 apresentam agrupamentos possíveis:









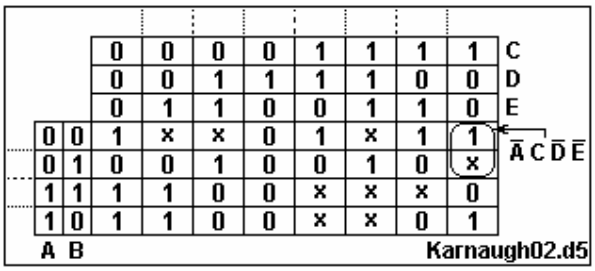
- em Karnaugh02.d2, agrupa-se {4} com a célula {6} ($\overline{A}\,\overline{B}C\overline{D}\,\overline{E}$), sendo D a única variável diferente

- em Karnaugh02.d3, agrupa-se {4} com a célula {0} ($\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}$), sendo C a única variável diferente;

- em Karnaugh02.d4, agrupa-se {4} com a célula {20} ($A\overline{B}C\overline{D}\,\overline{E}$), sendo A a única variável diferente.

> Nota importante: doravante, e ainda que não sejam *fisicamente* adjacentes, designar-se-ão de *'adjacentes'* células que – como ocorre com {4, 5}, {4, 6}, {4, 0}, {4, 12} e {4, 20} - são *equidistantes* dos espelhos imaginários supracitados. O leitor pode confirmar que, *não por acaso*, e a propósito de uma célula *qualquer*, {**i**},
> - o *número de células que lhe são adjacentes* é, *sempre*, o *número de variáveis da função* – no caso presente, **5**;
> - o *módulo da diferença* entre {i} e os rótulos dessas células adjacentes é *sempre* uma potência-de-2 (1, 2, 4, 8 e 16).

Relativamente à célula {4}, já só resta a célula {12} ($\overline{A}\,BC\overline{D}\,\overline{E}$) – cujo conteúdo é '**x**'. Que fazer? *Adiante-se desde já que, se for conveniente*, é viável combinar ambas as células {4, 12} - pois que só diferem numa variável, **B**, *vidé* Karnaugh02.d5



Uma última interrogação: dadas *duas* células *adjacentes*, *qual a expressão algébrica a que corresponde o seu agrupar?* Resposta: será um AND das variáveis (ou, quiçá, das suas negações) que tenham o *mesmo valor* nessas células. Eis exemplos:

- seja o par de células {4, 6}, *vidé* Karnaugh02.d2: ambas se situam na *mesma linha*, para a qual {A=**0**, B=**0**} - então, na expressão algébrica deverá aparecer { $\overline{A}\,\overline{B}$ }; por outro lado, localizam-se em *duas colunas*, para as quais {C=**1**, E=**0**} - então, na expressão algébrica deverá aparecer { $C\overline{E}$ }: aquele par será representado por $\overline{A}\,\overline{B}C\overline{E}$. O leitor pode conferir que este AND assume o valor '**1**' para a combinação {A=**0**, B=**0,** C=**1**, E=**0**} – e *apenas para ela* (e independentemente do valor de D);

- seja o par {4, 20}, *vidé* Karnaugh02.d4: as células situam-se, ambas, na *mesma coluna*, para a qual {C=**1**, D=**0** e E=**0**} - então, na expressão algébrica deverá aparecer { $C\overline{D}\,\overline{E}$ }; por outro lado, localizam-se em *duas linhas*, para as quais {B=**0**} - então, na expressão deverá aparecer { $\overline{B}$ }: aquele par será representado por $\overline{B}C\overline{D}\,\overline{E}$; o leitor pode conferir que este AND assume o valor '**1**' para a combinação {B=**0**, C=**1**, D=**0**, E=**0**} – e *apenas para ela* (e independentemente do valor de A).

Até ao momento, apenas se considerou a junção de *duas* células – artifício esse que permite descartar *uma* variável. Repare-se, todavia, nos resultados que se obteriam se se aglutinassem os *pares de células* {4, 5} e {6, 7}:

- o primeiro par conduziria, isto já se viu acima, a $\overline{A}\,\overline{B}C\overline{D}$;

- o segundo par conduziria, não custa discerni-lo, a $\overline{A}\,\overline{B}CD$;

Ora acontece que estas duas expressões diferem em apenas *uma* variável, D – pelo que se podem simplificar em, apenas:

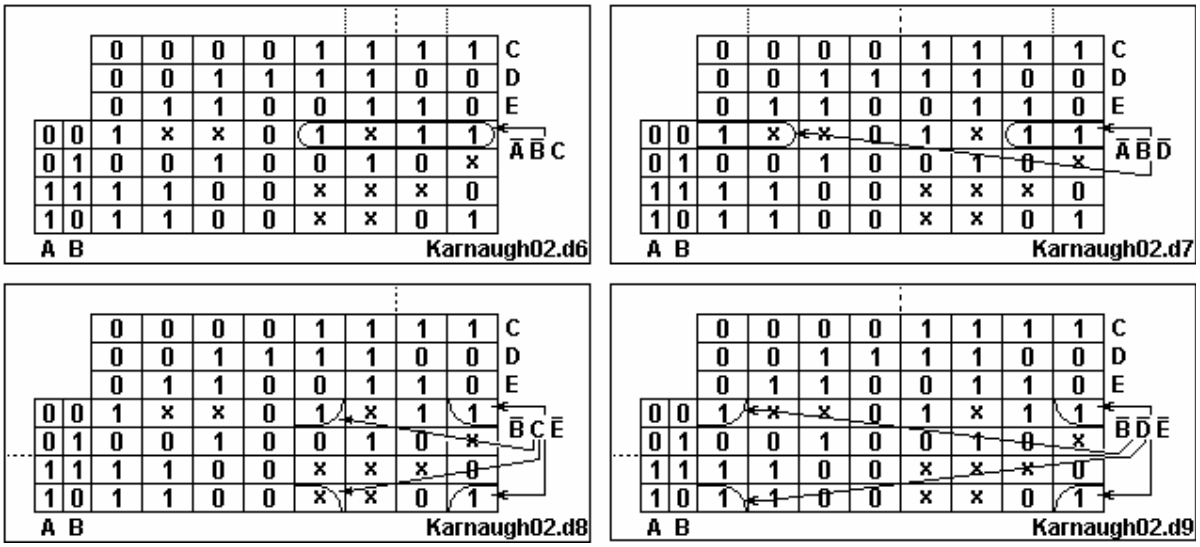
$$\overline{A}\,\overline{B}C\left(\overline{D}+D\right) = \overline{A}\,\overline{B}C$$

Em termos da tabela de Karnaugh, o que se está a fazer é associar **4** células, {4, 5, 6, 7}, que diferem em, apenas, *duas* variáveis – **D** e **E**. Essas células gozam de uma propriedade notável: são dois pares de células que, no seio do par, são adjacentes – e os próprios pares são adjacentes entre si (isto é: *equidistantes* de um dos espelhos imaginários supracitados).

Para escrever a *expressão algébrica a que corresponde o seu agrupar*, vale uma regra similar àquela enunciada acima: será um AND das variáveis (ou, quiçá, das suas negações) que tenham o *mesmo valor* nas quatro células. No caso, as células situam-se na *mesma linha*, para a qual {A=**0**, B=**0**} - então, na expressão algébrica deverá aparecer { $\overline{A}\,\overline{B}$ }; por outro lado, localizam-se em *quatro colunas*, para as quais {C=**1**}; então a expressão algébrica que representa aquelas 4 células será $\overline{A}\,\overline{B}C$; o leitor pode conferir que este AND assume o valor '**1**' para a combinação {A=**0**, B=**0,** C=**1**} – e *apenas para ela* (e independentemente dos valores de **D** e **E**).

Continuando a restringir por ora a casos que envolvam a célula {4}, os agrupamentos possíveis de 4 células encontram-se em Karnaugh02.d6-d9:

- em Karnaugh02.d6, agrupam-se {4, 5, 6 , 7};
- em Karnaugh02.d7, agrupam-se {0, 1, 4, 5};
- em Karnaugh02.d8, agrupam-se {4, 6, 20, 22};
- em Karnaugh02.d9, agrupam-se {0, 4, 16, 20}.



A seguinte sinopse faz um resumo *mental* dos agrupamentos feitos:

Implicantes			Células com '1'											
			0	4	5	6	11	15	16	17	20	24	25	
Mintermo	$\overline{A}\,\overline{B}CD\overline{E}$.			**1**										Contido nos seguintes
Não primos	$\overline{A}\,\overline{B}C\overline{D}$	Karnaugh02.d1		**1**	1									Contido em $\overline{A}\,\overline{B}C$
	$\overline{A}\,\overline{B}C\overline{E}$	Karnaugh02.d2		**1**		1								Contido em $\overline{A}\,\overline{B}C$
	$\overline{A}B\overline{D}\,\overline{E}$	Karnaugh02.d3	1	**1**										Contido em $\overline{B}\,\overline{D}\,\overline{E}$
	$\overline{B}C\overline{D}\,\overline{E}$	Karnaugh02.d4		**1**							1			Contido em $\overline{B}C\overline{E}$
Primos	$\overline{A}C\overline{D}\,\overline{E}$	Karnaugh02.d5		**1**										
	$\overline{A}\,\overline{B}C$	Karnaugh02.d6		**1**	1	1								
	$\overline{A}\,\overline{B}\,\overline{D}$	Karnaugh02.d7	1	**1**	1									
	$\overline{B}C\overline{E}$	Karnaugh02.d8		**1**		1					1			
	$\overline{B}\,\overline{D}\,\overline{E}$	Karnaugh02.d9	1	**1**					1		1			

Resumindo: a célula {**4**} corresponde à combinação de variáveis {A=**0**, B=**0**, C=**1**, D=**0**, E=**0**}. O facto de ela conter '**1**' significa que, para *essa* combinação, a função assume o valor '**1**'. Pensando na função como OR de vários ANDs, ela deverá então conter (antes de simplificada) o *mintermo* $\overline{A}\,\overline{B}C\overline{D}\,\overline{E}$: precisamente o AND que se volve em '**1**' *apenas* para tal combinação. O que a sinopse acima diz é que, ademais desse mintermo, existem mais 9 ANDs capazes de *cobrir* esse '**1**' – correspondentes a outros tantos modos distintos de *agrupar* essa célula (com outras). Consoante o que se vier a fazer, o efeito naquele OR será *diferente*:

- *sem simplificar*, a função envolveria onze *mintermos*, cada um de **5** variáveis:

$$f(A,B,C,D,E)=...+\overline{A}\,\overline{B}C\overline{D}\,\overline{E}+\overline{A}\,\overline{B}C\overline{D}E+\overline{A}\,\overline{B}CD\overline{E}+...$$

- agrupando as *duas* células {**4**, 5} poder-se-á substituir o mintermo referido por um produto de **4** variáveis, $\overline{A}\,\overline{B}C\overline{D}$;

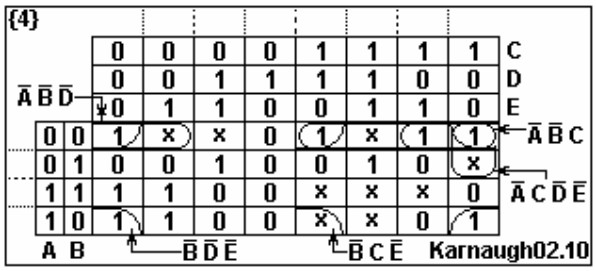
$$f(A,B,C,D,E)=...+\overline{A}\,\overline{B}C\overline{D}+\overline{A}\,\overline{B}CD\overline{E}+...$$

- mas, por junção das *quatro* células {**4**, 5, 6 e 7} já se poderá substituí-lo por um produto de **3** variáveis, $\overline{A}\,\overline{B}C$:

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

Sendo o objectivo chegar á expressão *mínima* para a função dada – em particular com menos ANDs -, é claro que, entre estes dois agrupamentos, será preferível o último: descarta-se um maior número de ANDs…

Nota *terminológica*: os nove termos a que se chegou - por *associação de células adjacentes* - designam-se de *implicantes*. Os primeiros quatro podem ser simplificados – originando-se os restantes cinco; estes designam-se de *implicantes primos*. Karnaugh02.10 apresenta – em um único quadro - os *cinco implicantes primos que cobrem a célula* {4}.



Adivinha-se a etapa seguinte: varrer todo o quadro de Karnaugh e, para cada uma das células em que ocorre '**1**', determinar o conjunto de *implicantes primos* que a cobre. Isso é apresentado a seguir – ressalvando-se desde já que, por motivos óbvios, convém o leitor treinar-se a fazê-lo *mentalmente*:

- a propósito da célula {0}, as células *adjacentes* a considerar são: {1, 4 e 16}; tentando aglutinar {0} com duas destas – e alguma outra (*em geral*, como que perfazendo os *vértices dum rectângulo*) -, surgem estes *implicantes primos*:

{0, 1, 16, 17} → $\overline{B}C\overline{D}$; {0, 4, 16, 20} → $\overline{B}\,\overline{D}\,\overline{E}$; {0, 1, 4, 5} → $\overline{A}\,\overline{B}\,\overline{D}$ (*vidé* Karnaugh02.11);

- a propósito da célula {6}, as células *adjacentes* a considerar são: {4, 7 e 16}; tentando agora aglutinar {6} com duas deste conjunto – e alguma outra -, surgem os seguintes *implicantes primos*:

$\{4, 5, 6, 7\} \rightarrow \overline{A}\,\overline{B}C$; $\{6, 7, 22, 23\} \rightarrow \overline{B}CD$; $\{4, 6, 20, 22\} \rightarrow \overline{B}C\overline{E}$ (*vidé* Karnaugh02.12);

Na dedução dos implicantes primos que *cobrem* uma célula {i}, parte-se do *conjunto de células* que lhe são adjacentes – e que há *a considerar*, isto é, *não estão preenchidas com '**0**'s!*

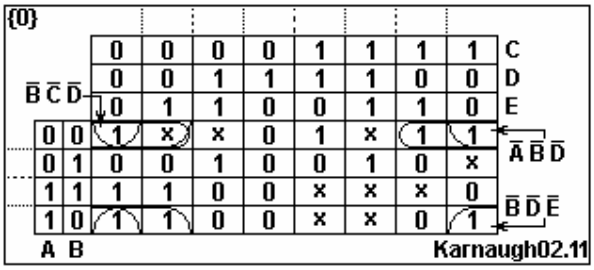
Se ele estiver *vazio* (não há *adjacentes a considerar*), então o respectivo mintermo é implicante primo…

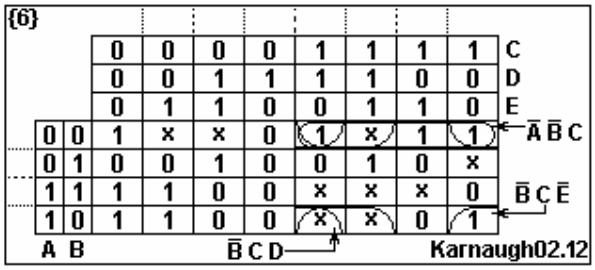
Se ele contiver *uma* célula só, então o respectivo aglutinar volve-se no implicante primo que a cobre…

Mas, se ele tiver mais células, como são os casos das células {0} e {6}, então:

- tenta-se aglutinar {**i**} com *duas* das suas adjacentes – e alguma *outra não adjacente*. (*vidé* acima)
- tenta-se aglutinar {**i**} com *três* das suas adjacentes – e mais *quatro não adjacentes*.
- e assim sucessivamente…

Sempre que, neste aglutinar, *um grupo se encontra contido noutro mais amplo*, aquele é descartado em favor do último! Por exemplo, ambos os grupos {**4**, 5} e {**4**, 6} se encontram dentro do grupo {**4**, 5, 6, 7} – pelo que são descartados…



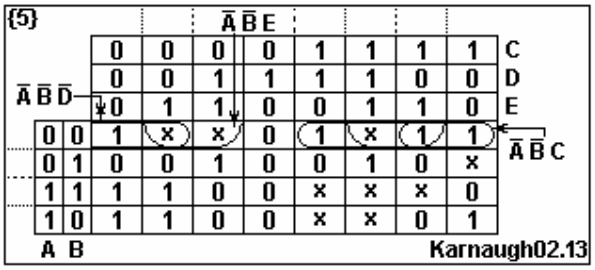


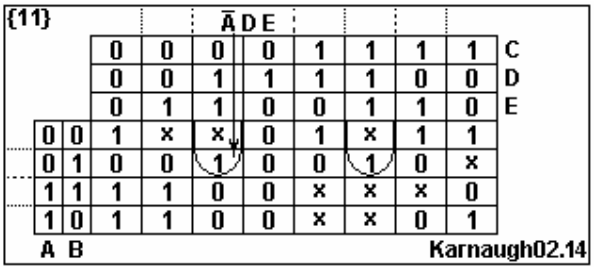
- a propósito da célula {5}, as *adjacentes* a considerar são: {1, 4 e 7}; aglutinando, chega-se aos *implicantes primos*:

$\{4, 5, 6, 7\} \rightarrow \overline{A}\,\overline{B}C$; $\{0, 1, 4, 5\} \rightarrow \overline{A}\,\overline{B}\,\overline{D}$; $\{1, 3, 5, 7\} \rightarrow \overline{A}\,\overline{B}E$ (*vidé* Karnaugh02.13);

- a propósito da célula {11}, as *adjacentes* a considerar são: {3 e 15}; aglutinando, chega-se ao *implicante primo*:

$\{3, 7, 11, e\ 15\} \rightarrow \overline{A}DE$ (*vidé* Karnaugh02.14);



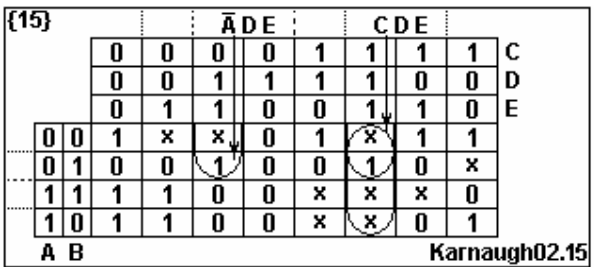


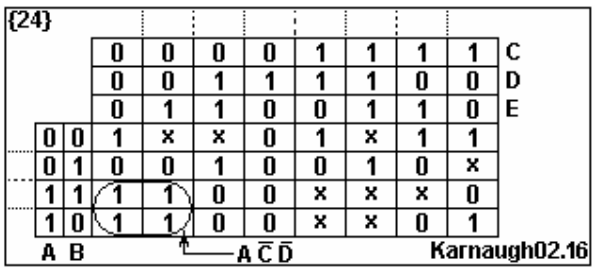
- a propósito da célula {15}, as *adjacentes* a considerar são: {7, 11 e 31}; aglutinando, chega-se aos *implicantes primos*:

$\{3, 7, 11, e\ 15\} \rightarrow \overline{A}DE$; $\{7, 15, 23, 31\} \rightarrow CDE$ (*vidé* Karnaugh02.15);

- a propósito da célula {24}, as *adjacentes* a considerar são: {16 e 25}; aglutinando, chega-se ao *implicante primo*:

$\{16, 17, 24\ e\ 25\} \rightarrow A\overline{C}\,\overline{D}$ (*vidé* Karnaugh02.16);

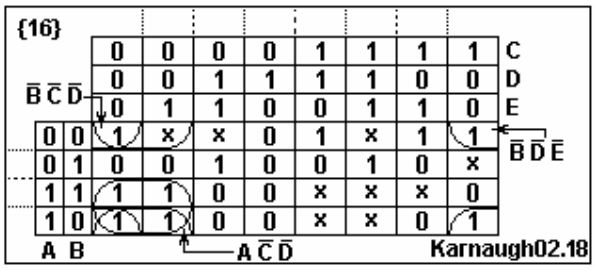




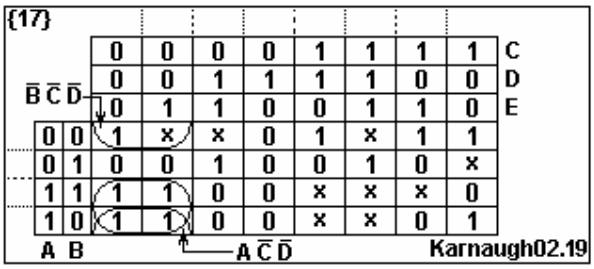
- a propósito da célula {25}, as *adjacentes* a considerar são: {17, 24 e 29}; aglutinando, chega-se aos *implicantes primos*:

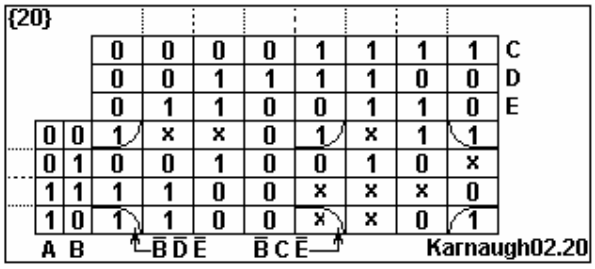
$\{16, 17, 24\ e\ 25\} \rightarrow A\overline{C}\,\overline{D}$; $\{25, 29\} \rightarrow AB\overline{D}E$ (*vidé* Karnaugh02.17);

- a propósito da célula {16}, as *adjacentes* a considerar são: {0, 17, 20 e 24}; aglutinando, chega-se aos *implicantes primos*:
{16, 17, 24 e 25} → $A\overline{C}\overline{D}$; {0, 4, 16, 20} → $\overline{B}\overline{D}\overline{E}$; {0, 1, 16, 17} → $\overline{B}\overline{C}D$ (*vidé* Karnaugh02.18);



- a propósito da célula {17}, as *adjacentes* a considerar são: {1, 16 e 25}; aglutinando, chega-se aos *implicantes primos*:
{16, 17, 24 e 25} → $A\overline{C}\overline{D}$; {0, 1, 16 e 17} → $\overline{B}\overline{C}\overline{D}$ (*vidé* Karnaugh02.19);
- a propósito da célula {20}, as *adjacentes* a considerar são: {4, 16 e 22}; aglutinando, chega-se aos *implicantes primos*:
{0, 4, 16, 20} → $\overline{B}\overline{D}\overline{E}$; {4, 6, 20, 22} → $\overline{B}C\overline{E}$ (*vidé* Karnaugh02.20);





Neste varrimento das células, há dois casos singulares: para a generalidade delas, chegou-se a dois ou mais implicantes primos – mas, para {**11**} e {**24**}, chegou-se a *um* implicante primo só: $\overline{A}DE$ e $A\overline{C}\overline{D}$. Que significa isto? Que, para cada uma dessas células, há uma maneira – *e uma só!* - de a cobrir: na simplificação da função terão que constar *obrigatoriamente* esses dois implicantes primos! A um implicante primo com esta propriedade – de que há ao menos uma célula que só ele a cobre – dá-se o nome de *essencial*. Na expressão simplificada deverá então vir $f(A,B,C,D,E)=\overline{A}DE+A\overline{C}\overline{D}+...$

Que deverá constar na *cauda* da função? Claramente, o conjunto de implicantes primos precisos para cobrir os '**1**'s *ainda não cobertos* por $\overline{A}DE$ e $A\overline{C}\overline{D}$! Poderá ajudar o seguinte resumo *mental* das células que eles cobrem:

Implicantes essenciais	**Células com '1':**	**0**	**4**	**5**	**6**	**11**	**15**	**16**	**17**	**20**	**24**	**25**
$\overline{A}DE$	Karnaugh02.14					**1**	1					
$A\overline{C}\overline{D}$	Karnaugh02.16							1	1		**1**	1

Falta discernir o *menor 'chapéu' de implicantes primos* cobrindo as células {0, 4, 5, 6 e 20}. Novo exercício *mental*:

Implicantes Primos		**0**	**4**	**5**	**6**	**20**
{0, 1, 16, 17}	$\overline{B}\overline{C}\overline{D}$	1				
{0, 4, 16, 20}	$\overline{B}\overline{D}\overline{E}$	1	1			1
{0, 1, 4, 5}	$\overline{A}\overline{B}\overline{D}$	1	1	1		
{4, 5, 6 , 7}	$\overline{A}\overline{B}C$		1	1	1	
{4, 6, 20, 22}	$\overline{B}C\overline{E}$		1		1	1
{1, 3, 5, 7}	$\overline{A}\overline{B}E$			1		
{6, 7, 22, 23}	$\overline{B}CD$				1	
{4, 12}	$\overline{A}CD\overline{E}$		1			

Eis algumas escolhas possíveis (com, apenas, *dois* implicantes primos):

- {**0**, **4**, 16, **20**}/ $\overline{B}\,\overline{D}\,\overline{E}$ e {4, **5**, **6** , 7}/ $\overline{A}\,\overline{B}\,C$

- {**0**, 1, **4**, **5**}/ $\overline{A}\,\overline{B}\,\overline{D}$ e {4, **6**, **20**, 22}/ $\overline{B}\,C\,\overline{E}$

Daqui se deduzem as expressões mais simples para a função, dadas acima (em R.1).

Último cuidado – **muiiiiiiito importante**: nas expressões algébricas a que se chega por estes métodos, o *traço* sobre uma variável – representando o respectivo complemento – deve *cobrir só essa variável. Não é correcto prolongá-lo para a variável seguinte!* $A\,\overline{C}\,\overline{D}$ e $A\,\overline{C\,D}$ (por exemplo) *não são o mesmo*: atribua o leitor dois valores *diferentes* a C e D, e topa-o logo. Ao escrever $A\,\overline{C}\,\overline{D}$, *o leitor tem mesmo que levantar o lápis do papel entre os dois traços* sobre C e D…

Cumpre dizer: após deduzir tais expressões R.1, *não há mais nada a fazer!* Em particular, *seria um erro* aplicar-lhes uma transformação algébrica, como seria pôr em evidência $\overline{A}$ e/ou $\overline{D}$:

$$f(A,B,C.D) = \overline{A}\left(D\,E + \overline{B}\,C\right) + \overline{D}\left(A\,\overline{C} + \overline{B}\,\overline{E}\right)$$

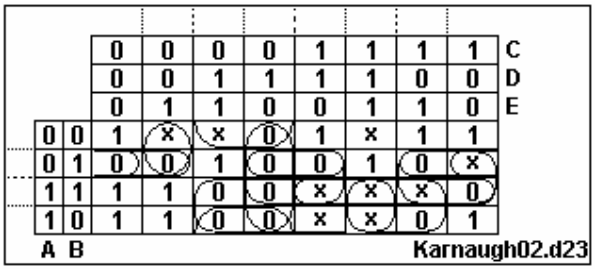
É que isto *já não é a forma disjuntiva*, isto é: já não é uma soma de produtos…

Reparo: a resolução acima atendeu à *forma* pedida para a função – a *disjuntiva (soma de produtos)*; houvera sido solicitada a *forma conjuntiva (produtos de somas)*, o método de resolução seria em tudo semelhante, com as seguintes diferenças:

- as células que teriam necessariamente de ser *cobertas* seriam agora aquelas preenchidas com '**0**'s;
- as células *adjacentes a considerar* seriam agora aquelas não preenchidas com '**1**'s;
- agora, haveria que procurar os *implicados* primos, começando também por discernir os essenciais.

Observando tais cuidados, as *células a cobrir*, as suas *adjacentes* e os *implicados primos* seriam (*vidé* Karnaugh02.23):

	2	**8**	**9**	**10**	**13**	**14**	**18**	**19**	**21**	**26**	**27**	**28**
	3,10,18	9,10,12	1,8,13	2,8,14,26	9,12,29	10,12,30	2,19,22,26	3,27,18,23	23,29	10,18,27,30	19,26,31	12,29,30
2,3,18,19	0						0	0				
2,10,18,26	0			0			0			0		
8,912,13		0	0		0							
8,10,12,14		0		0		0						
1,9			0									
10,14,26,30				0		0				0		
12,13,28,29					0							0
18,19,22,23, 26,27,30,31							0	0		0	**0**	
21,23,29,31									**0**			
28,29,30,31												0



As células cobertas (cada uma) por, apenas, *um* implicado, são {21} e {27}. Os implicados primos *essenciais* seriam então aqueles obtidos pelo agrupamento das células **enegrecidas** (na coluna à esquerda da sinopse acima): respectivamente, $\overline{A} + \overline{C} + \overline{E}$ e $\overline{A} + \overline{D}$. Entrando eles obrigatoriamente na solução, resta cobrir as células {2, 8, 9, 10, 13, 14 e 28}…

- {2} é coberto por dois implicados – sendo preferível aquele que cobre mais células ainda por cobrir: $C + \overline{D} + E$;

- {28} é coberto por dois implicados – sendo preferível aquele que cobre mais células ainda por cobrir: $\overline{B} + \overline{C} + D$ …

Com tais escolhas, resta cobrir as células {8, 9 e 14}… Optando por uma das três possibilidades que se oferecem, viria:

$$f(A,B,C,D,E) = \left(\overline{A} + \overline{C} + \overline{E}\right)\left(\overline{A} + \overline{D}\right)\left(C + \overline{D} + E\right)\left(\overline{B} + \overline{C} + D\right)\left(A + \overline{B} + D\right)\left(A + \overline{B} + E\right)$$

Uma nota, quiçá desnecessária: dado *um grupo de células adjacentes*, *qual é agora a expressão algébrica a que corresponde o seu agrupar?* Resposta: será um OR das variáveis (ou, quiçá, das suas negações) que tenham o *mesmo valor* nessas células. Como exemplo, repare-se no grupo de células {18, 19, 22, 23, 26, 27, 30 e 31}: todas elas se situam em *duas linhas*, para as quais {A=**1**} - então, na expressão algébrica deverá aparecer { $\overline{A}$ }; por outro lado, todas elas se situam em *quatro colunas*, para as quais {D=**1**} - então, na expressão algébrica deverá aparecer { $\overline{D}$ }: aquele grupo será representado por $\overline{A}+\overline{D}$. O leitor pode conferir que este OR assume o valor '**0**' para a combinação {A=**1**, D=**1**} – e *apenas para ela* (e independentemente dos valores de B, C e E).

2. [10E1.2] Considere a seguinte função booleana, em que A é a variável de maior peso:

$$f(A,B,C,D,E)=\sum m(0,4,6,8,10,11,21,24,27,29,30,31)+\sum m_d(1,3,7,12,13,14,15,16,17,20,22,25,26,28)$$

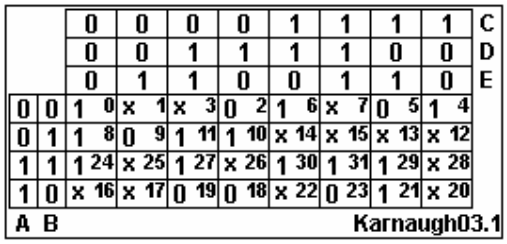
Obtenha a expressão mínima na forma disjuntiva (soma de produtos) para esta função utilizando o método de Karnaugh. Identifique os implicantes primos essenciais da função.

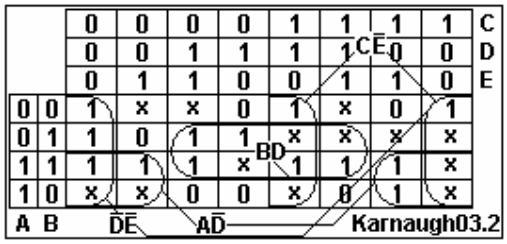
R.1: Karnaugh03.1 foi logrado seguindo procedimentos referidos em [10E2.3]. A tabela adiante exibe as *células a cobrir*, as suas *adjacentes* e os *implicantes primos*. Sendo a célula '21' coberta por um só implicante primo, ele é *essencial*. Cobrindo ele as células {21, 24 e 29}, sobram {0, 4, 6, 8, 10, 11, 27, 30 e 31}; a menor combinação que as cobre é (*vidé* Karnaugh03.2):

$$f(A,B,C,D) = A\overline{D} + B\,D + \overline{D}\,\overline{E} + C\,\overline{E}$$

	0	**4**	**6**	**8**	**10**	**11**	**21**	**24**	**27**	**29**	**30**	**31**
	1,4,8,16	0,6,12,20	4,7,14,22	0,10,12,24	8,11,14,26	3,10,15,27	17,20,29	8,16,25,26,28	11,25,26,31	13,21,25,28,31	6,14,22,26,28	15,27,29,30
0,1,16,17	1											
0,4,8,12, 16,20,24,28	1	1		1				1				
4,6,12,14, 20,22,28,30		1	1								1	
6,7,14,15			1									
8,10,12,14, 24,26,28,30				1	1			1			1	
10,11,14,15, 26,27,30,31					1	1			1		1	1
3,11						1						
16,17,20,21 24,25,28,29							**1**	1		1		
24,25,26,27, 28,29,30,31								1	1	1	1	1
12,13,14,15, 28,29,30,31										1	1	1

(Bem entendido: não é estritamente necessário *escrever* a tabela acima: ela está aí só por atenção a quem está abordando o método "Karnaugh" pela primeira vez… Mas não deverá o leitor estranhar se, algo mais à frente neste documento, a resolução de problemas similares vier a prescindir da *escrita* da tabela: com algum treino, será pacífico ao leitor discernir *mentalmente* ao menos uma combinação de implicantes primos que possa ser solução – e isso tendo como apoio (e justificação) apenas as curvas "fechadas" que os representam, *vidé* Karnaugh03.2)





R.2: $A\overline{D}$, em virtude de **m21**.

Note-se: é um *erro* afirmar que o implicante primo essencial é m21! É, sim, $A\overline{D}$ - resultado da aglutinação de 8 mintermos: m16, …, m21, …. A menção a m21 significa isto: *não há outro* implicante primo cobrindo m21 (*vidé* tabela acima), o *único* que o cobre é $A\overline{D}$, é precisamente *por essa singularidade* que $A\overline{D}$ é *essencial*.

3. [10T1.3] Considere a seguinte função booleana, em que A é a variável de maior peso:

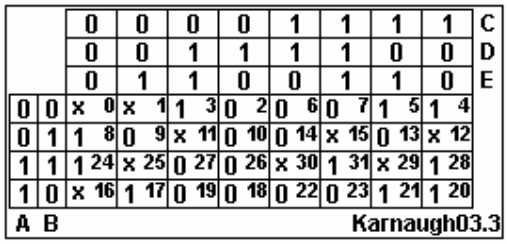
$$f(A,B,C,D,E)=\sum m(3,4,5,8,17,20,21,24,28,31)+\sum m_d(0,1,11,12,15,16,25,29,30)$$

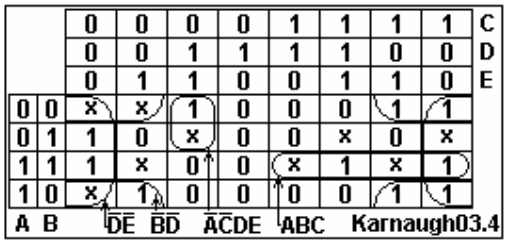
Obtenha a expressão mínima na forma disjuntiva (soma de produtos) para esta função utilizando o método de Karnaugh. Identifique os implicantes primos essenciais da função.

R.1: Karnaugh03.3 foi logrado seguindo procedimentos referidos em [10E2.3]. A tabela adiante exibe as *células a cobrir*, as suas *adjacentes* e os *implicantes primos*. Sendo as células '5' e '8' cobertas por um só implicante primo (cada uma), são *essenciais*. Cobrindo {4, 5, 8, 17, 20, 21, 24, 28}, sobram {3 e 31}; a menor combinação que as cobre é (*vidé* Karnaugh03.4):

$$f(A,B,C,D) = \overline{B}\,\overline{D} + \overline{D}\,\overline{E} + ABC + \overline{A}\,\overline{B}\,\overline{C}E \text{ (ou } \overline{B}\,\overline{D} + \overline{D}\,\overline{E} + ABC + \overline{A}\,\overline{C}DE)$$

	3	**4**	**5**	**8**	**17**	**20**	**21**	**24**	**28**	**31**
	1,11	0,5,12,20	1,4,21	0.12.24	1,16,21,25	4,16,21,28	5,17,20,29	8,16,25,28	12,20,24,29,30	15,29,30
1,3	1									
3,11	1									
0,4,8,12,16,20,24,28		1		**1**		1		1	1	
0,1,4,5,16,17,20,21		1	**1**		1	1	1			
16,17,20,21, 24,25,28,29					1	1	1	1	1	
28,29,30,31									1	1
15,31										1





R.2: $\overline{B}\,\overline{D}$ e $\overline{D}\,\overline{E}$ em virtude de, respectivamente, **m5** e **m8**.

4. [10E4.2] Considere a seguinte função booleana, em que A é a variável de maior peso:

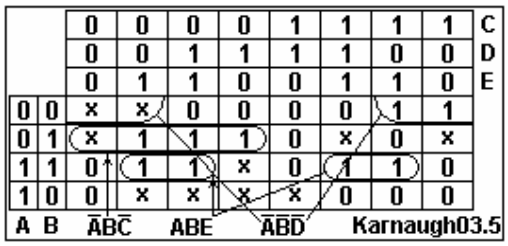
$$f(A,B,C,D,E) = \sum m(4,5,9,10,11,25,29,31) + \sum m_d(0,1,8,12,15,17,18,19,22,26,27)$$

4. 1. Obtenha a expressão mínima na forma disjuntiva (soma de produtos) para esta função utilizando o método de Karnaugh. Identifique os implicantes primos essenciais da função.

4. 2. Manipule algebricamente a função obtida de forma a poder implementá-la utilizando unicamente portas NAND e portas NOT.

R.1: O preenchimento do quadro em Karnaugh03.5 foi logrado seguindo procedimentos referidos em [10E2.3]. A tabela adiante exibe as *células a cobrir*, as suas *adjacentes* e os *implicantes primos*. Sendo as células '**5**' e '**29**' cobertas por um só implicante primo (cada uma), esse par de implicantes primos é *essencial*. Cobrindo {4,5,25,29,31}, sobram {9,10 e 11}; a menor combinação que as cobre é (*vidé* Karnaugh03.4):

$$f(A,B,C,D) = \overline{A}\,\overline{B}\,\overline{D} + ABE + \overline{A}\,B\,\overline{C}$$



	4	**5**	**9**	**10**	**11**	**25**	**29**	**31**
	0,5,12	1,4	1,8,11,25	8,11,26	9,10,15,27	9,17,27,29	25,31	15,27,29
0,1,4,5	1	**1**						
0,8,4,12	1							
0,1,8,9			1					
8,9,10,11			1	1	1			
1,9,17,25			1			1		
11,15,27,31					1			
9,11,25,27			1		1	1		
10,11,26,27				1	1			
25,27,29,31						1	**1**	1

Implicantes primos essenciais: $\overline{A}\,\overline{B}\,\overline{D}$ e ABE em virtude de, respectivamente, **m5** e **m29**.

Alerta importante: é um *erro* aglutinar os *nove* mintermos {m9,m10,m11,m17,m18,m19,m25,m26 e m27} (ou apenas *seis* deles) – por muito *adjacentes* que *pareçam* ser; repare-se na coluna à esquerda da tabela acima: linha a linha, o número de mintermos aglutinados é sempre 4 – conquanto, valha a verdade, no caso geral de uma função de 5 variáveis pudesse também assumir os valores 1, 2, 8, 16 ou 32… É possível aglutinar um mintermo com *outro* mintermo, isto é: um total de **2** mintermos;

quiçá seja possível aglutinar esses 2 mintermos com *outros* 2 mintermos, isto é: um total de **4** mintermos; quiçá seja possível aglutinar esses 4 mintermos com *outros* 4 mintermos, isto é: um total de **8** mintermos… Resumindo e concluindo: o número de mintermos passível de ser aglutinado é *sempre* uma potência de 2 – e 6 e 9 não o são…

R2: Procedendo à *dupla complementação* de f (conforme ao teorema da *involução*, isso não muda o valor de f), e aplicando depois as leis de De Morgan, vem: $f = \overline{\overline{A\,B\,D} + A\,B\,E + \overline{A}\,B\,C} = \overline{\overline{A\,B\,D}}\;\overline{A\,B\,E}\;\overline{\overline{A}\,B\,C}$ - com, apenas, NANDs e NOTs

5. [10E3.2] Considere a seguinte função booleana, em que A é a variável de maior peso:

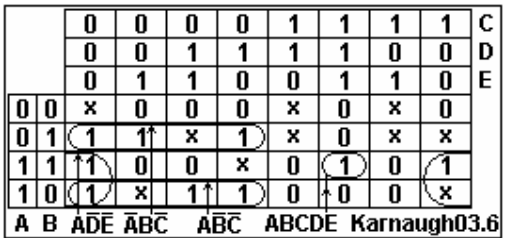
$$f(A,B,C,D,E) = \sum m(8,9,10,16,18,19,24,28,31) + \sum m_d(0,5,6,11,12,13,14,17,20,26)$$

Obtenha a expressão mínima na forma disjuntiva (soma de produtos) para esta função utilizando o método de Karnaugh. Identifique os implicantes primos essenciais da função.

R.1: Karnaugh03.6 foi preenchido seguindo procedimentos referidos em [10E2.3]. A tabela adiante exibe as *células a cobrir*, as suas *adjacentes* e os *implicantes primos*. Sendo as células '19' e '31' cobertas por um só implicante primo (cada uma), são *essenciais*. Cobrindo {16, 18, 19 e 31}, sobram {8, 9, 10, 24 e 28}; a menor combinação que as cobre é (*vidé* Karnaugh03.6):

$$f(A,B,C,D) = \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABCDE + A\overline{D}\overline{E} \text{ (ou } \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABCDE + B\overline{D}\overline{E}\text{)}$$

	8	**9**	**10**	**16**	**18**	**19**	**24**	**28**	**31**
	9,10,12,24	8,11,13	11,14,26	17,18,20,24	19,26	17,18	8,26,28	12,20,24	
0,8,16,24	1			1			1		
8,9,10,11	1	1	1						
8,9,12,13	1	1							
8,12,24,28	1						1	1	
8,10,12,14	1		1						
8,10,24,26	1		1				1		
16,17,18,19				1	1	1			
16,18,24,26				1	1		1		
16,20,2428				1			1	1	
31									1



R.2: $A\overline{B}\overline{C}$ e $ABCDE$ em virtude de, respectivamente, **m19** e **m31**.

Circuitos Combinatórios

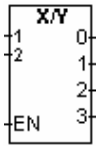
{Comb_r.doc}

Preâmbulo:

P1: ***Descodificadores***:

Um descodificador *(decoder)* é um dispositivo de cujas saídas está activa (por ex., a '**1**' enquanto as demais estão a '**0**') apenas aquela *rotulada* pelo valor das entradas de *selecção*. (*En* simboliza o conjunto de entradas *Enable*).

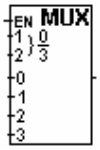
Símbolo qualificador IEC: X/Y, DEC/BCD, BIN/OUT, BIN/7-SEG, etc



P2: ***Multiplexers***:

Um *multiplexer* é um dispositivo que entrega na *saída* o valor que se apresentar na entrada de dados *rotulada* pelo valor das entradas de *selecção* (*En* simboliza o conjunto de entradas *Enable*)

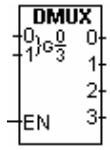
Símbolo qualificador IEC: MUX



P3: ***Demultiplexers***:

Um *Demultiplexer* é um dispositivo que entrega na *saída rotulada* pelo valor das entradas de *selecção* o valor que se apresentar na sua (única) entrada de dados

Símbolo qualificador IEC: DMUX ou DX



Uma imagem para um descodificador/*decoder* será um dispositivo que recebe *permanentemente* corrente eléctrica para um semáforo – e que a *entrega a uma de* três lâmpadas (de cores verde, amarelo e vermelha). *Não há informação a provir do exterior*: quando muito, quiçá a algumas horas o semáforo se encontre apagado – e daí a conveniência em dotar aquele dispositivo de uma entrada EN: só quando ela estiver '**1**' é que a corrente eléctrica chegará às lâmpadas…

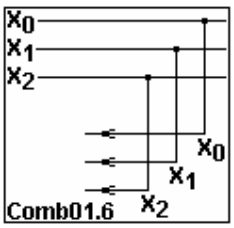
Já uma imagem para um *demultiplexer* poderá ser a de um carteiro que da sua mala extrai cartas – para as enfiar uma a uma nas caixas do correio dum prédio: uma no r/c-D, outra no 2º E, outra no 6º F… Agora, *há informação a provir do exterior*: aquela que vem nas próprias cartas… Tirando situações particulares (por ex., publicidade), o que ele deposita numa caixa será *distinto* daquilo que foi para alguma outra…

Uma imagem para um *multiplexer* poderá ser um semáforo com *uma* só lâmpada a que está associado um *dispositivo* que, consoante as variáveis de selecção, assim cobre a lâmpada com um *filtro* da pertinente côr: verde **ou** amarela **ou** vermelha; uma outra imagem será um computador de alguém que detém várias *mail-boxes* na Internet: consoante a selecção que ele fizer, assim nesse portátil serão descarregados os *e-mails* de uma *mail-box* **ou** de outra **ou** de outra…

P4: ***Recomendações para a Ordenação de fios em logigramas***:

Um dos perigos que *sempre* espreita quem materializa um *logigrama* é trocar fios, mormente no seio de barramentos (*buses*). Para minimizar o número de horas de aperto, seguem-se *recomendações* (*vidé* Comb01.6):

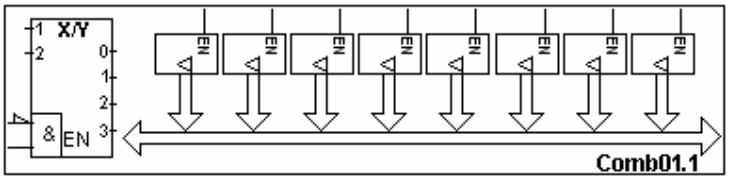
A informação corre, como ***regra geral***, *da esquerda para a direita*, e *de cima para baixo*; violações a esta regra (*que hão de ser justificadas*) devem ser explicitadas mediante *setas nos próprios fios*…



Em *buses* verticais, o autor recomenda dispôr ***sempre*** *à direita* o fio de *menor-peso*;

Em *buses* horizontais, o autor recomenda dispôr ***sempre*** *no topo* o fio de *menor-peso*

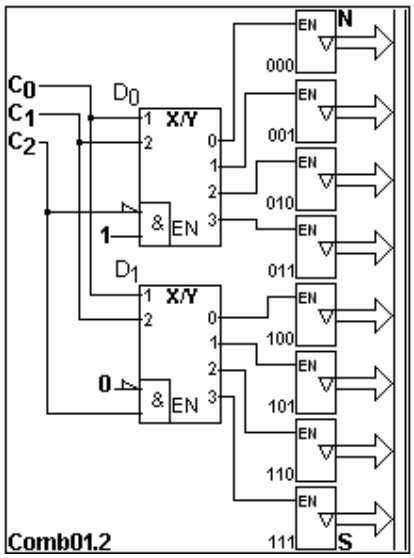
1. [10E2.4] Uma parte de um circuito de transferência de dados é composto por 8 registos de 4 bits com *enable* e saída *tri-state*, ligados a um *bus* de 4 bits comum (*vidé* Comb01.1). Pretendem-se eliminar



conflitos no *bus*. Indique de quantas variáveis irá precisar para controlar o acesso ao *bus*, e desenhe as ligações necessárias utilizando o número mínimo possível de circuitos descodificadores idênticos ao apresentado. Não pode utilizar portas lógicas adicionais.

R: *Eliminar conflitos* significa: quando um registo acede ao bus, nenhum mais acede…Posto que lhe acedem 8 registos - e $8=2^3$ -, serão precisas **3** variáveis de controlo. As ligações necessárias encontram-se esquematizadas em Comb01.2.

Repare-se: Designem-se de $C_2C_1C_0$ as variáveis de controlo – e rotulem-se (de **N** para **S**) os registos de {0, 1, 2, …, 6, 7} ou, em binário, {000, 001, 010, …, 111}. Então, para seleccionar o *penúltimo* registo (por exemplo), deve forçar-se C_2=**1**,C_1=**1**,C_0=**0** (repare-se: **110** é a representação binária de **6**); quando tal suceder, na entrada **En** *desse* registo surge **1**, e em todos os outros registos deve surgir **En**=**0**.



O descodificador comporta 4 saídas – com que se pode seleccionar um entre, seja, os 2^2=4 registos a **N** {000, …, 011**}**; restando mais 4, é preciso pelo menos mais um descodificador… Nomeando-os de D_1 e D_0, eis a estratégia das ligações:

- quando C_2=**1**, activa-se D_1 (um dos 4 registos a **S** vai aceder ao *bus*), quando C_2=**0**, activa-se D_0 (um dos 4 registos a **N** vai aceder ao *bus*). A activação de D_1 é controlada pelo respectivo **En** – ele mesmo um '**&**' de duas entradas, activas a *Low* (a de cima) e *High* (a de baixo). Pressuponha-se *lógica positiva*, isto é, a correspondência: {'**0**'→**Low**, '**1**'→**High**}. Então, em ordem a activar D_1 quando C_2=**1**, C_2 é ligado à entrada "de baixo" do '**&**'que gera **En**, a "de cima" ficando, é claro, ligada a '**0**'; consideração análoga vale para o **En** de D_0;

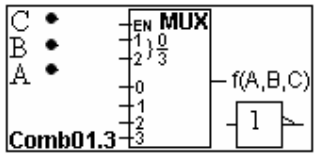
- C_1C_0 aplicam-se a ambas as entradas de selecção {1,2} dos descodificadores – **ordenadas C_0→'1', C_1→'2'** (i.e.: 'variável de controlo de *menor* peso' → 'entrada de selecção de *menor* peso'); então, quando, seja, C_2=**1**,C_1=**1**,C_0=**0**, D_1 fica activado - e a saída que nele fica activa é a **2** (repare-se: **10** representa '**2**' em binário).

Nota 1: as ligações poderiam ser outras… Considere-se, por ex., a troca de C_0 com C_2: às entradas de selecção (de D_1D_0) liga-se C_1C_2, e às entradas **En** liga-se C_0… Então, para activar o *penúltimo* registo, o que se logra activando a saída '**2**' de D_1, é preciso forçar C_2=**0**,C_1=**1**,C_0=**1** (e 011 é bem distinto do rótulo do registo, 110: é *menos elegante* que a solução preconizada).

Nota 2: Se o *decoder* disponibilizado oferecesse *uma única* entrada *Enable*, ter-se-ia que se recorrer a mais um *decoder* (ou a um NOT): na entrada *Enable* de D_0, aplicar-se-ia $\overline{C_2}$, e na entrada *Enable* de D_1 aplicar-se-ia C_2…. Acontece que o *decoder* disponibilizado oferece *duas* entradas *Enable* – e isso dispensa a adição de tal lógica: uma arquitectura *Enable* mais complexa habilita a um logigrama mais simples…

Nota 3: Comb01.2 segue o Preâmbulo P4: em particular, os fios que suportam $C_2C_1C_0$ foram dispostos de modo que C_0 fica no topo (e C_2 fica em baixo), idem quanto ao *decoder*, tanto nas entradas de selecção (a de *menor peso*, '**1**', fica *em cima*) quanto nas saídas (a de *menor peso*, '**0**', fica *em cima*); o mesmo quanto aos registos: o de *menor peso*, '**000**', fica em cima…

2. [10E2.5, 10T3.4] A partir dos circuitos indicados em Comb01.3, desenhe o logigrama de um circuito que indica se um número binário ABC de 3 bits tem paridade ímpar (i.e., contém um número impar de '1').

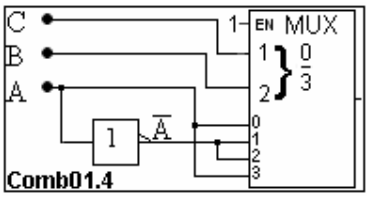


R: Para a dedução da tabela de verdade de **f**, sabe-se que, quando ABC tem um número ímpar de '1's – na prática, quando tem 1 ou 3 '**1**'s – **f** assume o valor '**1**'; nos restantes, assume o valor '**0**'. Por ex., para o número 011, que tem um número *par* de '1's (dois, para se ser preciso), deve ser f=0… Varridos todos os casos, chega-se a:

A	**B**	**C**		
0	0	0	**0**	
0	0	1	**1**	← Nº ímpar de 1's
0	1	0	**1**	← Nº ímpar de 1's
0	**1**	**1**	**0**	

1	0	0	**1**	← Nº ímpar de 1's
1	0	1	**0**	
1	1	0	**0**	
1	1	1	**1**	← Nº ímpar de 1's
			f	

O correspondente logigrama encontra-se em Comb01.4.



Comentário: pretende-se materializar a *tabela de verdade* – função de **3** variáveis – por um *multiplexer* com apenas **2** entradas de *selecção* (rotuladas '1' e '2')… Uma possibilidade (*vidé* Comb01.4) é aplicar-lhes, às entradas de selecção, as variáveis {B,C} - **ordenadas C→'1', B→'2'** (i.e.: 'bit de *menor* peso' → 'entrada de selecção de *menor* peso'); fica por saber: que é que se liga às entradas de *dados* do *multiplexer*?

Para o discernir, repare-se: quando {B=**1**, C=**0**} (por exemplo) – ou seja, as entradas de *selecção* {2,1} assumem os valores {1,0}, a saída **f** irá assumir o valor na entrada de *dados* '2' (repare-se: **10** representa **2** em binário). Bastará saber o valor da tabela de verdade quando {B=**1**, C=**0**} – e aplicar esse valor nessa entrada '2'… Para o discernir, *reordenem-se mentalmente* as linhas da tabela de verdade, originando-se as *quatro* tabelas adiante:

A	**B**	**C**	
0	0	0	**0**
1	0	0	**1**

A	**B**	**C**	
0	0	1	**1**
1	0	1	**0**

A	**B**	**C**	
0	**1**	**0**	**1**
1	**1**	**0**	**0**

A	**B**	**C**	
0	1	1	**0**
1	1	1	**1**

Varrendo os *pares de linhas* – da cima para baixo –, {B,C} assumem sucessivamente os valores {00}, {01}, {10} e {11}. Centrando a atenção por ora no par em que {B=**1**, C=**0**}, repare-se: quando A=0, vem f=1, mas, quando A=1, vem f=0! Isto é, **f** vem a ser simplesmente o *complemento de A*! Ou seja: quando {B=**1**,C=**0**}, o valor na entrada de dados '**2**' deve ser, precisamente, $\overline{A}$! Varrendo os restantes casos, deduz-se, por raciocínio similar, o que aplicar às entradas {0, 1, e 3}:

- quando {B=**0**, C=**0**}, f=A → aplicar A na entrada de dados '**0**';
- quando {B=**0**, C=**1**}, f= $\overline{A}$ → aplicar $\overline{A}$ na entrada de dados '**1**';
- quando {B=**1**, C=**1**}, f=A → aplicar A na entrada de dados '**3**';

Nota: o logigrama poderia ser outro… Experimente o leitor aplicar {A,B} às entradas de selecção (que não {B,C}) – e verificará por si mesmo que já não será necessária a reordenação das colunas da tabela: uma resolução *mais rápida*, portanto…

3. [10E1.3, 10T3.5] Desenhe o logigrama de um circuito combinatório que recebe na entrada um número de 2 bits e gera como saída um número de 4 bits correspondente ao quadrado do número de entrada. Utilize um único descodificador com saídas activas a "*Low*" (com as dimensões mínimas necessárias) e o número mínimo de portas lógicas elementares adicionais.

R: Para a dedução da(s) tabela(s) de verdade do circuito, designem-se de $\{X_1X_0\}$ os 2 bits à entrada, e de $\{Y_3Y_2Y_1Y_0\}$ os 4 bits de saída; esta deve representar o *quadrado da entrada*. Por ex., para a entrada 11 (que representa '3' em binário), a saída será 1001 (que representa '9' em binário). Varridos todos os casos, chega-se a:

Entrada:	X_1	X_0
0 →	0	0
1 →	0	1
2 →	1	0
3 →	1	1

0	0	0	0	← 0
0	0	0	**1**	← 1
0	**1**	0	0	← 4
1	0	0	**1**	← 9
Y_3	Y_2	Y_1	Y_0	Saída

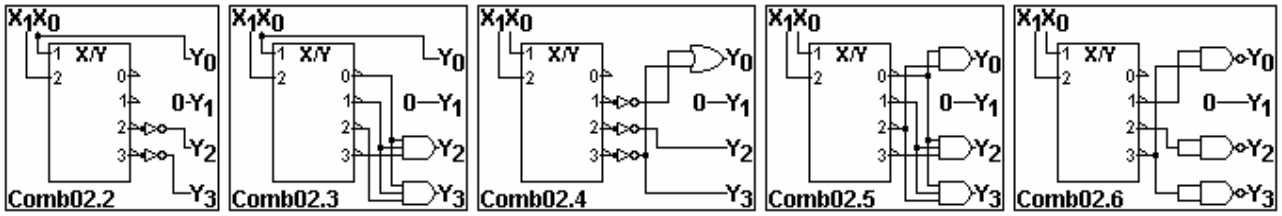
Construídas as tabelas, segue-se o logigrama Comb02.2… Eis a estratégia seguida:

- Porquanto a entrada comporta *dois* bits, opta-se por um descodificador 2:4;
- X_1X_0 aplicam-se às entradas de *selecção* {1,2} – **ordenadas X_0→'1', X_1→'2'** (i.e.: 'bit de *menor* peso' → 'entrada de selecção de *menor* peso'). Então, quando, seja, {X_1=**1**,X_0=**1**} (e reparando que **11** é a representação binária de **3**), a saída '**3**', e apenas ela, fica *activa* – o que, conforme ao enunciado, significa: ela fica *Low*, as outras ficam *High*. Pressupondo *lógica positiva*, i.e., a correspondência: {'**0**'→**Low**, '**1**'→**High**}, o funcionamento do

descodificador será: para a combinação {X_1=**1**,X_0=**1**}, a saída '**3**' assume o valor '**0**' e as outras assumem o valor '**1**';

- Conforme à respectiva tabela de verdade, Y_3 deve ser '**1**' se, e só se, {X_1=**1**,X_0=**1**}; ou seja, se, e só se, precisamente a saída '**3**' está a '**0**'! Bastará então *negar* essa saída '3', e ligar o resultado a Y_3;
- Quanto a Y_2, esta deve ser '**1**' se, e só se, {X_1=**1**,X_0=**0**}; ou seja (e reparando que **10** é a representação binária de **2**), se, e só se, precisamente a saída '**2**' está a '**0**'! Bastará então *negar* essa saída '**2**', e ligar o resultado a Y_2;
- Quanto a Y_1, esta deve ser '**0**' *sempre*! Bastará então ligar Y_1 a '0';
- Quanto a Y_0, esta deve ser *sempre* igual a X_0! Bastará então ligar Y_0 a X_0.

(Sublinhe-se: a solução supõe saídas activas a *Low*; se fossem activas a *High*, as saídas '**2**' e '**3**' não deveriam ser negadas!)



Comentário: Existe pelo menos outra solução, *vidé* Comb02.3: com ANDs cujas entradas são as saídas do descodificador.

- De facto, e conforme à tabela de verdade, Y_3 deve ser '**0**' se, e só se, {X_1=**0**,X_0=**0**} **ou** {X_1=**0**,X_0=**1**} **ou** {X_1=**1**,X_0=**0**} - ou seja, se, e só se, *estiver a* **0***' ao menos uma das saídas* {'**0**', '**1**' ou '**2**'} do descodificador! Bastará então ligar a Y_3 um AND – cujas entradas sejam precisamente essas saídas {'0', '1' e '2'}…
- Quanto a Y_2, esta deve ser '**0**' se, e só se, {X_1=**0**,X_0=**0**} **ou** {X_1=**0**,X_0=**1**} **ou** {X_1=**1**,X_0=**1**} - ou seja, se, e só se, *estiver a* **0***' ao menos uma das saídas* {'**0**', '**1**' ou '**3**'} do descodificador! Bastará então ligar a Y_2 um AND – cujas entradas sejam precisamente essas saídas {'**0**', '**1**' e '**3**'}…

Advertência aos mecanicistas: ambos os raciocínios acima se podem, é óbvio, estender também a Y_0 (por exemplo)…

- concretamente, Y_0 deve ser '**1**' se, e só se, {X_1=**0**,X_0=**1**} **ou** {X_1=**1**,X_0=**1**} - ou seja, se, e só se, *estiver a '0' uma das saídas* {'1' ou '3'} do descodificador! Bastará então ligar a Y_0 um OR – cujas entradas sejam precisamente os complementos dessas saídas {'1' e '3'}, *vidé* Comb02.4;
- uma outra linha de reflexão será: Y_0 deve ser '**0**' se, e só se, {X_1=**0**,X_0=**0**} **ou** {X_1=**1**,X_0=**0**} - ou seja, se, e só se, *estiver a* **0***' uma das saídas* {'0' ou '2'} do descodificador! Bastará então ligar a Y_0 um AND – cujas entradas sejam precisamente essas saídas {'0' e '2'}, *vidé* Comb02.5.

Do ponto de vista do enunciado, *estes logigramas estão incorrectos*: estão acrescentando, a Comb02.2 (ou Comb02.3), um OR (ou um AND) - e nesse sentido não cumprem o requisito "*número* mínimo *de portas lógicas elementares adicionais*". Dum engenheiro espera-se, não só que o circuito que ele concebeu realize a função pretendida, como o logre com o mínimo custo!

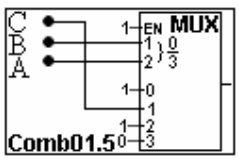
O uso de NANDs: As tabelas de verdade de Y_3 e Y_2 remetem para números designativos com um só '**1**' – e daí a simplicidade de Comb02.2… Mas já o de Y_0 envolve dois '**1**'s; então, e embora Comb02.4 não possa ser considerado solução para *este exercício particular*, todavia patenteia uma orientação *mais genérica*. Nele, Y_0 é a saída dum OR de dois NOTs – o que em termos práticos clama por *dois* integrados. Mais compacto será substituir ORs de NOTs (ou simplesmente NOTs) por NANDs (*vidé* leis de DeMorgan), como se apresenta em Comb02.6: este é um logigrama que, ademais do *decoder*, exige um só integrado mais (e, de facto, se se ligar Y_0 directamente a X_0, conduz a uma outra solução válida para o enunciado).

O uso de NORs: Também é possível, para *este exercício particular (o que não quer dizer que não seja possível para outros)*, desenhar uma solução com somente NORs... Bastará, em Comb02.2, usar NORs em vez de NOTs…

4. [10T1.5] Dado o logigrama Comb01.3, complete-o com as ligações necessárias para implementar a função $f(A,B,C) = \overline{A}\,\overline{B} + \overline{A}BC + A\overline{B}$. Não pode utilizar portas lógicas adicionais.

.

R: Pretende-se materializar uma função – de **3** variáveis – por um *multiplexer* que disponibiliza apenas **2** entradas de *selecção* (rotuladas '**1**' e '**2**'). O correspondente logigrama encontra-se em Comb01.5. Eis dois métodos de a ele chegar:



O Método mais moroso: **f** é um OR: volve-se em '1' se, e só se, *ao menos um* dos operandos vale '1'; no caso, os operandos são ANDs: assumem o valor '1' se, e só se,

todos os seus operandos são '1'. Concretamente, f será '1' quando {A=0 **e** B=0} **ou** {A=0 **e** B=1 **e** C=1} **ou** {A=1 **e** B=0}, *vidé* tabela:

A	**B**	**C**				
0	0	0	1	0	0	**1**
0	0	1	1	0	0	**1**
0	1	0	0	0	0	**0**
0	1	1	0	1	0	**1**
1	0	0	0	0	1	**1**
1	0	1	0	0	1	**1**
1	1	0	0	0	0	**0**
1	1	1	0	0	0	**0**
			$\overline{A}\,\overline{B}$	$\overline{A}BC$	$A\overline{B}$	f

Considerandos:

$\overline{A}\,\overline{B}=1$: se e só se A=**0 e** B=**0**

$\overline{A}BC=1$: se e só se A=**0 e** B=**1 e** C=**1**

$A\overline{B}=1$: se e só se A=**1 e** B=**0**

f=1: se e só se $\overline{A}\,\overline{B}=1$ **ou** $\overline{A}BC=1$ **ou** $A\overline{B}=1$

De modo similar a [10E2.5], será pacífico aplicar às entradas de selecção *um par* das variáveis {A,B,C}. Poderá ser {AB}, {AC} ou {BC}. De forma a confrontar as várias opções, *reordenem-se mentalmente* as quadrículas da tabela de verdade:

A	**B**	**C**	
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**0**
			f

A	**C**	**B**	
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**
			f

B	**C**	**A**	
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	**1**
1	1	1	**0**
			f

Daqui, deduz-se o que se deve aplicar nas várias entradas de dados do multiplexer… Por ex., a tabela da esquerda refere que, quando {A=**0**, B=**0**}, deve **f** tomar o valor **1**; então, na entrada rotulada '0', deve aplicar-se '1'. Em forma de tabela:

Conforme à tabela *à esquerda*,	Conforme à tabela *do meio*,	Conforme à tabela *à direita*,
'0'→ quando {A=0, B=0}, f=**1**	quando {A=0, C=0}, f=$\overline{B}$	quando {B=0, C=0}, f=1
'1'→ quando {A=0, B=1}, f=**C**	quando {A=0, C=1}, f=1	quando {B=0, C=1}, f=1
'2'→ quando {A=1, B=0}, f=**1**	quando {A=1, C=0}, f=$\overline{B}$	quando {B=1, C=0}, f=0
'3'→ quando {A=1, B=1}, f=**0**	quando {A=1, C=1}, f=$\overline{B}$	quando {B=1, C=1}, f=$\overline{A}$

Entre as várias escolhas em jogo, a mais simples – pois que não envolve o inversor que as restantes exigem – é a da *esquerda*: Aplica-se {A, B} às entradas de selecção - **ordenadas B→'1', A→'2'** (i.e.: 'variável de *maior* peso' → 'entrada de selecção de *maior* peso') – e nas entradas de *dados* forçam-se os valores enegrecidos nas várias linhas dessa escolha: {**1**,**C**,**1**,**0**}

Um Método mais rápido: O método que acabou de se aplicar é *universal* – no sentido de que não lhe escapa nenhuma função; mas tem um custo: a demora em chegar à solução. Com o fito de lhe chegar mais rapidamente, convém reflectir… Do que se discutiu em [10E2.5], a solução pretende responder a duas perguntas:

- quais as variáveis a aplicar às *entradas de selecção* do multiplexer?
- quais os valores a forçar nas *entradas de dados* do multiplexer?

A resposta à primeira pergunta é pacífica: nas entradas de *selecção* devem aplicar-se *duas* das variáveis da função {A, B e C} - *preferivelmente deixando de fora aquela que nunca aparece complementada na sua expressão algébrica!* Ora, essa expressão contém $\overline{A}$ e $\overline{B}$ - mas não contém $\overline{C}$! Pelo que será preferível aplicar A e B nas entradas de selecção – ordenadas por, *por exemplo*, A→'2', B→'1' …

Após essa decisão, fica por decidir: que valor se deve forçar na entrada de *dados* rotulada '**0**' (por exemplo)? Claramente, o valor que a função adquire para a combinação de dados {A=**0**, B=**0**} (repare-se: **00** é a representação binária de **0**). Ora, para tal combinação, a função assume o valor

$f(0,0,C)=\bar{0}\bar{0}+\bar{0}0C+0\bar{0}=1+0+0=1$ - pelo que será pacífico: na entrada de dados rotulada '**0**' deve aplicar-se o valor '**1**'… Para as demais entradas de dados, o método será análogo:

A=**0**, B=**0** $f(0,0,C)=\bar{0}\bar{0}+\bar{0}0C+0\bar{0}=1+0+0=1$

A=**0**, B=**1** $f(0,1,C)=\bar{0}\bar{1}+\bar{0}1C+0\bar{1}=0+C+0=C$

A=**1**, B=**0** $f(1,0,C)=\bar{1}\bar{0}+\bar{1}0C+1\bar{0}=0+0+1=1$

A=**1**, B=**1** $f(A,B,C)=\bar{1}\bar{1}+\bar{1}1C+1\bar{1}=0+0+0=0$

A conclusão é: às entradas de dados do multiplexer deve aplicar-se {1, C, 1 e 0} – como se patenteia em Comb01.5.

5. [10E4.1] A saída Y de um circuito lógico de três entradas {A, B, C} é dada por:

$Y=A+B\,C$, se $A\,B=0$;

$Y=A+\bar{B}\,\bar{C}$, se $A\,B=1$;

Desenhe o logigrama do circuito:

5. 1. usando apenas portas NOR;

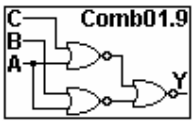
5. 2. usando um Multiplexer com duas entradas de selecção.

5. 3. usando um decoder com duas entradas de selecção, a que se aplicaram {A, C}.

R1: Comb01.9, *vidé* justificação adiante

R2: Comb01.10 ou Comb01.12 (sendo possíveis outras soluções mais)

R3: Comb01.11 e Comb01.13



Eis duas estratégias na abordagem – conduzindo aos logigramas *mais simples*:

A: Por *manipulações algébricas*: confirme o leitor a expressão algébrica $Y=\overline{A\,B}(A+BC)+(AB)\left(A+\bar{B}\,\bar{C}\right)$.

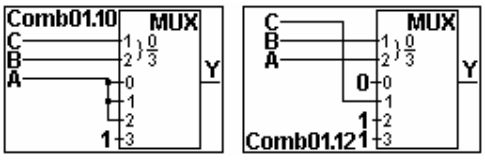
- quando $A\,B=0$, vem $Y=\bar{0}(A+BC)+(0)\left(A+\bar{B}\,\bar{C}\right)=A+BC$;

- quando $A\,B=1$, vem $Y=\bar{1}(A+BC)+(1)\left(A+\bar{B}\,\bar{C}\right)=A+\bar{B}\bar{C}$ - precisamente o que o enunciado estabelece…

Convém *simplificar*: $Y=\left(\bar{A}+\bar{B}\right)(A+BC)+(AB)\left(A+\bar{B}\,\bar{C}\right)=\bar{A}BC+\bar{B}A+AB=\bar{A}\,BC+A=A+BC$

Ra1) Para chegar a um logigrama só com portas NOR, convém re-escrever Y sob a forma dum *produto de somas*, no caso pela aplicação da *distributividade*: $Y=(A+B)(A+C)$. Após o que se procede à *dupla complementação de* Y: conforme ao teorema da *involução*, isso não muda o valor de Y: $Y=\overline{\overline{Y}}=\overline{(A+B)(A+C)}$. Aplicando as leis de De Morgan, a expressão volve-se em: $Y=\overline{\overline{A+B}+\overline{A+C}}$ - com, apenas, NORs! O correspondente logigrama encontra-se em Comb01.9.

Ra2) Em ordem a materializar o circuito mediante um *Mux*, a primeira decisão concerne: quais, das três entradas do circuito (A, B, C), escolher para aplicar nas respectivas entradas de *selecção* (sejam S_2,S_1)? Claramente, existem 3*2=6 possibilidades distintas… A *lei do menor esforço* manda aplicar **B**→S_2 e **C**→S_1, *vidé* Comb01.10. Fica por dizer: que aplicar nas entradas de *dados* do Mux? Recorde-se o funcionamento deste dispositivo (*vidé* Preâmbulo P2): quando nomeadamente '**BC**' forem '**00**', Y irá assumir o valor que estiver aplicado à entrada '**0**'… Pelo que: devem aplicar-se, nas quatro entradas de dados, os valores que $Y=A+BC$ deve gerar quando '**BC**' assumirem os valores de, sucessivamente, '00', '01', '10' e '11'! Ei-los: para as primeiras três combinações - em que $BC=0$ -, vem $Y=A$, já para a última (em que $BC=1$), vem $Y=1$. Bastará então ligar '**A**' às entradas {0,..,2} e '**1**' à entrada '3'…

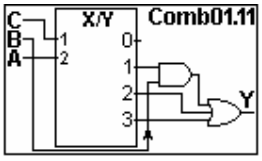


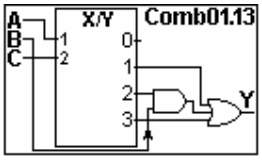
(Nota: a aplicação **B**→S_2 e **C**→S_1 foi justificada pela *lei do menor esforço*… Verifique o próprio leitor: se optar por **C**→S_2 e **A**→S_1 (por ex.), gasta um pouco mais de sangue, suor e lágrimas até chegar à solução)

Ra3) Em ordem a materializar o circuito mediante um *decoder*, admita-se que, nas respectivas entradas de selecção, se aplicaram **A**→S_2 e **C**→S_1, *vidé* Comb01.11. A presença de '+' em $Y = A + BC$ traduz que Y é gerado por um OR; resta saber: como lhe ligar as saídas do *decoder*? Recorde-se o seu funcionamento (*vidé* Preâmbulo P1): quando nomeadamente '**AC**' forem '**00**', a saída '**0**' irá ficar '**1**'… Então:

- Y advém '**1**' quando '**A**' é '**1**' – independentemente de '**C**' (e '**B**'); pelo que será pacífico ligar *directamente* ao OR as saídas do *decoder* que advêm '**1**' quando '**A**' é '**1**'- e elas são precisamente as saídas '**2**' e '**3**'…

- Y advém '**1**' quando '**C**' é '**1**' **e** '**B**' é '**1**'; pelo que será pacífico ligar ao OR as saídas do *decoder* que advêm '**1**' quando '**C**' é '**1**'- que são as saídas '**1**' e '**3**' – porém através de ANDs "*enabled*" por '**B**'; e de facto, porquanto a saída '**3**' já está *permanentemente* ligada ao OR, bastará o AND entre '**B**' e a saída '**1**' do *decoder*, *vidé* Comb01.11.





B: Uma outra estratégia na abordagem será através da *tabela de verdade*... Confirme o leitor a tabela adiante:

Y	0	1	C		$A+BC$	0	1	C		$A+\overline{B}\,\overline{C}$	0	1	C
0	0	**0**	**0**	← AB=0		0	0	**0**	**0** ← AB=0		0	0	
0	1	**0**	**1**	← AB=0	⇐	0	1	**0**	**1** ← AB=0	+	0	1	
1	0	**1**	**1**	← AB=0		1	0	**1**	**1** ← AB=0		1	0	
1	1	**1**	**1**	← AB=1		1	1				1	1	**1** **1** ← AB=1
A	B					A	B				A	B	

- se $AB = 0$ - o que sucede nas *linhas* '00', '01', '10'- vem $Y = A + B\,C$: Y assume '**1**' quando A=1 **ou** B=1 **e** C=1

- se $AB = 1$, vem $Y = A + \overline{B}\,\overline{C}$: Y assume '**1**' quando A=1 **ou** B=0 **e** C=0…

Rb1) Para chegar a um logigrama com, apenas, portas NOR, convém escrever Y sob a forma de um *produto de somas*, no caso pela aplicação do método de Karnaugh – em que se aglutinam os '**0**'s… Vem, de imediato, $Y = (A + B)(A + C)$. E, a partir daqui, continua-se como em Ra1), até chegar a Comb01.9.

Rb2) Em ordem a materializar o circuito mediante um *Mux*, a primeira decisão concerne: quais, das entradas do circuito (A, B, C), escolher para aplicar nas entradas de *selecção* (sejam S_2,S_1)? A *lei do menor esforço* manda aplicar **A**→S_2 e **B**→S_1, *vidé* Comb01.12. Fica por dizer: que aplicar nas entradas de *dados* do Mux? Resposta: recordando o Preâmbulo P2, devem aplicar-se, nas entradas {0, 1, 2, 3}, os valores que Y deve gerar quando '**AB**' assumirem os valores de, sucessivamente, '00', '01', '10' e '11'! Com um relance à tabela de verdade, constata-se que: para '00', vem $Y = 0$; para '01', vem $Y = C$. e nos restantes casos vem $Y = 1$. Bastará então ligar '**0**', '**C**', '**1**' e '**1**' às entradas de *dados* do Mux…

(Nota: a aplicação **A**→S_2 e **B**→S_1 foi justificada pela *lei do menor esforço*… Verifique o próprio leitor: se se optar por **C**→S_2 e **A**→S_1 (por ex.), o esforço em chegar à solução *a partir da tabela acima* será de facto algo maior)

Rb3) Em ordem a materializar o circuito mediante um *decoder*, convém, por comodidade, reescrever a tabela de verdade dispondo 'AC' na vertical.:

Y	0	1	B
0	0	**0**	**0** →Y=0
0	1	**0**	**1** →Y=1 se e só se B=1
1	0	**1**	**1** →Y=1
1	1	**1**	**1** →Y=1
A	C		

É patente que $Y = 1$ nas linhas da tabela em que '**AC**' assumem os valores '10' e '11' – para os quais ficam activas as saídas '**2**' e '**3**' do *decoder*; pelo que será pacífico ligar essas saídas *directamente* ao OR que gera Y. De outra parte, quando 'AC' assumem o valor '00', nunca Y se volve em '**1**' – pelo que não deverá estabelecer-se nenhuma ligação entra a saída '**0**' e aquele OR. Já quanto ao valor '01'em 'AC', que activa a saída '1', a tabela

mostra que Y se volve em '**1**' apenas na coluna 'B': antes do OR, há que intercalar um AND entre '**B**' e a saída '1' do *decoder*, *vidé* Comb01.11.

Soluções alternativas…

1. *Outra atribuição das entradas de selecção no decoder*: Acima, optou-se por aplicar **A**→S_2 e **C**→S_1; mas nada proíbe fazer-se de outra maneira, a saber: **A**→S_1 e **C**→S_2, *vidé* logigrama Comb01.13. Pergunta: que consequências acarreta isso? O decisivo é entender que, com estoutra opção, as saídas {0, 1, 2, 3} do *decoder* ficam activas quando, respectivamente, {A=**0**,C=**0**}, {A=**1**,C=**0**}, {A=**0**,C=**1**} e {A=**1**,C=**1**}. Então,

-ligar (*directamente* ao OR que gera Y) as saídas do *decoder* que advêm activas quando A=1 volve-se agora em ligar as saídas '1' e '3' (e, não, '2' e '3'!);

- ligar (àquele OR, através de um AND *enabled* por B) a saída do *decoder* que fica activa quando B=1 (e A=0) volve-se em ligar-lhe a saída '**2**' (e, não, a saída '**1**'!)

Ao progredir nesta abordagem, o leitor terá certamente deparado agora com uma maior incomodidade:

- quando se aplica **A**→S_2 e **C**→S_1, a saída '2' do *decoder* fica activa quando {A=**1**,C=**0**} – e, as "coisas batem certo": '**2**' representa-se em binário por '**10**';

- mas, quando se aplica **A**→S_1 e **C**→S_2, a saída '2' do *decoder* fica activa quando {A=**0**,C=**1**}; as "coisas já não parecem bater certo": '**2**' não se representa em binário por '**01**' - e isso obriga a uma maior ginástica mental…

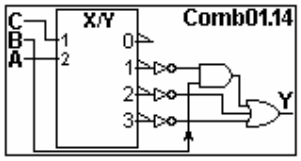
Por mor de comodidade (e, por conseguinte, de *eficiência*) na resolução de problemas, convém então observar a seguinte praxis na escolha das variáveis a aplicar às entradas de selecção dum *multiplexer* (ou *decoder*/*demultiplexer*): aplicar a variável de *menor* peso à entrada de selecção de *menor* peso, aplicar a variável de *maior* peso à entrada de selecção de *maior* peso, etc.

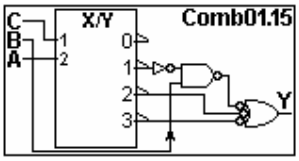
No caso, entre as variáveis '**A**' e '**C**', a de *maior* peso é '**A**' (na tabela, encontra-se à *esquerda* de '**C**'), e a de *menor* peso é '**C**'; quanto às variáveis de selecção do *decoder*, a de *maior* peso é a '**2**', e a de *menor* peso é a '**1**'; a atribuição preferível é, então: **A**→S_2 e **C**→S_1.

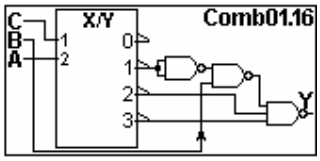
2. *Decoder com saídas activas a Low*: Seja o caso de o leitor ter desenhado o logigrama Comb01.11 – mas, ao montá-lo no laboratório, vem a saber que o *decoder* aí disponível tem as saídas activas a *Low*… Por outras palavras: uma saída *seleccionada* (por {AC}) advém '**0**', que não '**1**'… Um modo de ultrapassar este óbice é colocar um inversor (*NOT*) em cada uma das saídas usadas, *vidé* Comb01.14. Este pode ser transformado em Comb01.15 – mediante dois artifícios, a saber.

- trasladar os *NOT*s das saídas '**2**' e '**3**'do *decoder* para as entradas do OR que gera Y;

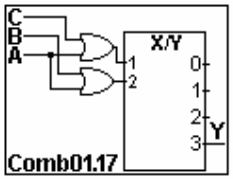
- interpor, à entrada do OR e à saída do AND, um par de '○' (simbolizando inversores que se cancelam mutuamente). Pelas leis de De Morgan, o OR, *cujas entradas estão todas negadas*, pode ser substituído por um NAND; de outra parte, o *NOT* na saída '1' também pode ser substituído por um NAND… A solução final é Comb01.16: além do *decoder*, já somente se vislumbram NANDs…



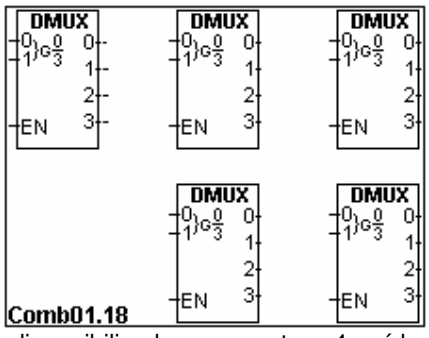




3. *Solução exótica*: A mente humana é por vezes muito fértil… e pode chegar a *pseudo-soluções* - que funcionam mesmo, mas que *devem ser descartadas*… Comb01.17 apresenta uma delas: Para a entender, repare-se que Y se pode escrever sob a forma de um *produto de somas,* $Y = (A + B)(A + C)$. Sabendo que a saída '**3**' do decoder fica '**1**' quando *ambas as entradas de selecção* advém '**1**', é tentador colocar nessas entradas exactamente A+B e A+C… Mesmo advogando que o logigrama não fica mais complicado que os demais, ele deve, é claro, ser descartado: em bom português, está usando "*um canhão para matar um mosquito*"… *Pode* significar algumas nuvens debaixo do chapéu de quem o subscreve…



6. [10E4.4] Faça as ligações necessárias para concretizar um *demultiplexer* 4:16 (4 bits de selecção; s_3, s_2, s_1, s_0; e 16 bits de saída: y_{15}, y_{14}, …, y_1, y_0) usando apenas os componentes da figura Comb01.18. Identifique todas as entradas e saídas.



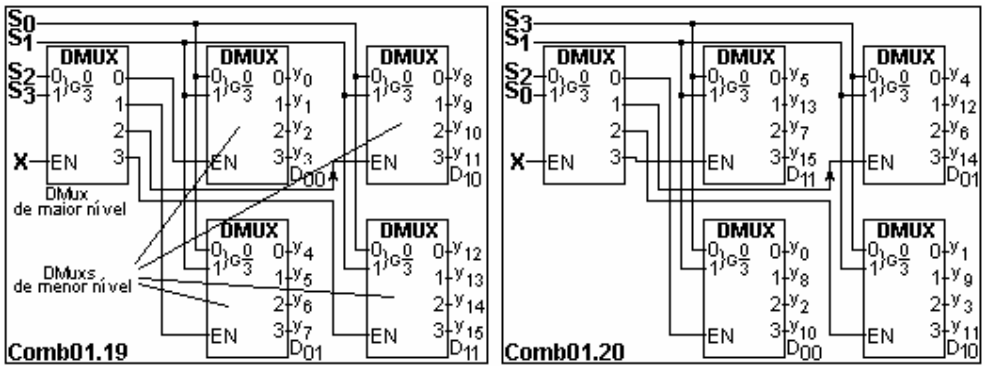
R: Comb01.19 apresenta uma das soluções possíveis:

Processo mental: Conforme ao Preâmbulo P3, um demultiplexer 4:16 é um circuito com 4 entradas de selecção, 16 saídas *e uma entrada de dados*; o valor '**0**'/'**1**' presente nessa entrada irá ser *encaminhado* para uma daquelas saídas – precisamente a que for seleccionada pela combinação de bits nas entradas de selecção… Acontece que os DMuxs disponibilizados comportam 4 saídas cada um; havendo que cobrir 16 saídas, são precisos cinco DMuxs: um de *maior* nível e 4 de *menor* nível,

numerados D_{00}, D_{01}, D_{10} e D_{11}. Com o primeiro DMUX, selecciona-se um dos restantes, e com estes selecciona-se uma das 16 saídas… Eis a estratégia das ligações:

- no DMux de *maior* nível, vidé Comb01.19, aplicam-se os 2 bits de selecção de *maior* nível {s_3, s_2} - **ordenados** $s_3 \rightarrow$**'1',** $s_2 \rightarrow$**'0'** (i.e.: 'bit de selecção de *maior* peso' → 'entrada de selecção de *maior* peso'); então, se, por ex., {s_3=**0**, s_2=**0**}, fica seleccionada a saída '**0**' desse DMux; para que fique seleccionado o DMux D_{00}, bastará ligar essa saída à entrada EN de D_{00}; considerações análogas valem para as ligações aos demais DMuxs: D_{01}, D_{10} ou D_{11}.

- nos quatro DMuxs de *menor* nível aplicam-se {s_1, s_0} - **ordenados** $s_1 \rightarrow$**'1',** $s_0 \rightarrow$**'0'** (i.e.: 'bit de selecção de *maior* peso' → 'entrada de selecção de *maior* peso'); então, se, por ex., {s_1=**1**, s_0=**1**}, fica seleccionada a saída '**3**' do DMux que tiver sido seleccionado (por {s_3, s_2}). No conjunto, se acontecer {s_3=**0**, s_2=**0**, s_1=**1**, s_0=**1**}, ficará seleccionada a saída '**3**' do DMUX D_{00} – pelo que será pacífico denominá-la de y_3 (**0011** é a representação binária de **3**). Identificar as outras saídas será trivial: não por acaso, bastará, para cada saída, somar o *peso* de DMUX em causa (entre '0', '4', '8' e '12') com a *identificação* (entre '0', '1', '2' e '3') dessa saída…



Parêntesis de alerta aos distraídos:

Poder-se-ia imaginar outras ligações… Considere-se, por ex., a aplicação de {s_0, s_2} ao DMUX da esquerda e de {s_1, s_3} aos restantes DMUX – e as interligações entre os DMUX representadas em Comb01.20… Agora, se, por ex., {s_0=**0**, s_2=**0**}, fica seleccionada a saída '**0**' do DMux da esquerda; para que fique seleccionado o DMux D_{00}, bastará ligar essa saída à entrada EN de D_{00}; considerações análogas valem para as ligações aos demais DMuxs. Até aqui, nada de novo… O desafio agora é: qual a identificação com que ficará agora a saída '**3**' do DMUX D_{00}? Repare-se: ela fica seleccionada para {s_0=**0**, s_2=**0**} e {s_1=**1**, s_3=**1**} – ou, reordenando: {s_3=**1**, s_2=**0**, s_1=**1**, s_0=**0**} – do que se segue que deverá ser identificada por y_{10}.

Compare agora o leitor a disposição dos identificadores {y_{15}, …, y_0} nos dois logigramas: a maneira elegante, *sequencial*, em Comb01.19, e a maneira *abstrusa* em Comb01.20… e quiçá fique mais disponível para aceitar a praxis já algures enunciada a propósito da escolha das variáveis a aplicar às entradas de selecção dum *multiplexer* (ou *decoder*/*demultiplexer*): em vez de o fazer *ao calhas*, aplicar a variável de *menor* peso à entrada de selecção de *menor* peso, aplicar a de *maior* peso à entrada de selecção de *maior* peso, etc.

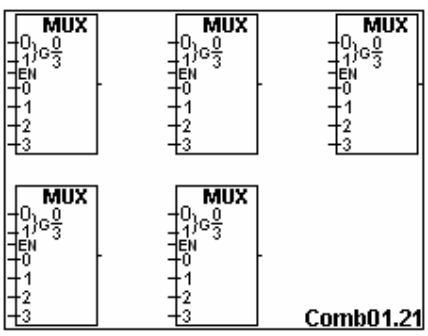
A disposição em Comb01.19 é fruto também da atenção tida na *distribuição geográfica* dos *demultiplexers*: compare-se, a esse respeito, as localizações de, por exemplo, D_{00} e D_{11}, em Comb01.19 e Comb01.20…

- Em Comb01.19, D_{10} e D_{11} ficam *alinhados à direita*: os DMUXs da *coluna* da direita são seleccionados quando s_3=1 (e os da que lhe fica à esquerda são seleccionados quando s_3=0).

- Em Comb01.19, D_{00} e D_{10} ficam *alinhados no topo*: os DMUXs da *linha* do topo são seleccionados quando s_2=0 (e os da que lhe fica por baixo são seleccionados quando s_2=1).

Última advertência: é um *erro* fixar '**1**' na entrada EN do DMUX de *maior* nível em Comb01.19! Tal '**1**' seria válido se o horizonte do problema versasse *decoders* – mas não, versa *demultiplexers*, que são algo diferente, *vidé* Preâmbulos P1 e P3! O que aí deve ser aplicado é o valor, designe-se de '**X**', que irá ser encaminhado (*demultiplexado*) para uma das 16 saídas {y_{15}, …, y_0}: se X=1, será '**1**' que irá aparecer na saída {y_{15}, …, y_0} seleccionada… mas se for X=0, será '**0**' que lá irá surgir…

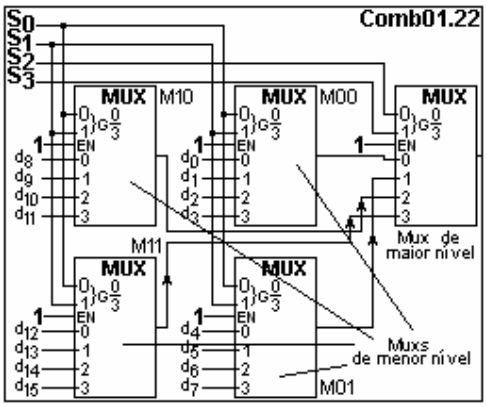
7. [10E3.4] Faça as ligações necessárias para concretizar um multiplexer 16:1 (4 bits de selecção: s3, s2, s1, s0; e 16 bits de dados: d15, d14, …, d0) usando apenas os componentes da figura Comb01.21. Identifique todas as entradas e saídas. Justifique.



R: Comb01.22 apresenta uma das soluções possíveis:

Processo mental: Conforme ao Preâmbulo P2, um multiplexer 16:1 é um circuito com 4 entradas de selecção, 16 entradas de dados *e uma saída*; será *encaminhado* para essa saída o valor '**0**'/'**1**' que se apresentar na entrada seleccionada pela combinação de bits nas entradas de

selecção… Acontece que os Muxs disponibilizados comportam 4 entradas de dados cada um; havendo que cobrir 16 entradas, são precisos cinco Muxs: um de *maior* nível e 4 de *menor* nível, numerados M_{00}, M_{01}, M_{10} e M_{11}. Com o primeiro MUX, selecciona-se um dos restantes, e com estes selecciona-se uma das 16 entradas… Eis a estratégia das ligações:



- no Mux de *maior* nível, *vidé* Comb01.22, aplicam-se os 2 bits de selecção de *maior* nível {s_3, s_2} - **ordenados** $s_3 \rightarrow$ **'1',** $s_2 \rightarrow$ **'0'** (i.e.: 'bit de selecção de *maior* peso' $\rightarrow$ 'entrada de selecção de *maior* peso'); então, se, por ex., {s_3=**0**, s_2=**0**}, fica seleccionada a entrada '**0**' desse Mux; para que fique seleccionado o Mux M_{00}, bastará ligar essa entrada à saída de M_{00}; considerações análogas valem para as ligações aos demais Muxs: M_{01}, M_{10} ou M_{11}.

- nos quatro Muxs de *menor* nível aplicam-se {s_1, s_0} - **ordenados** $s_1 \rightarrow$ **'1',** $s_0 \rightarrow$ **'0'** (i.e.: 'bit de selecção de *maior* peso' $\rightarrow$ 'entrada de selecção de *maior* peso'); então, se, por ex., {s_1=**1**, s_0=**1**}, fica seleccionada a entrada '**3**' do Mux que tiver sido seleccionado (por {s_3, s_2}). No conjunto, se acontecer {s_3=**0**, s_2=**0**, s_1=**1**, s_0=**1**}, ficará seleccionada a entrada '**3**' do MUX M_{00} – pelo que será pacífico denominá-la de d_3 (**0011** é a representação binária de **3**). Identificar as outras entradas será trivial: não por acaso, bastará, para cada entrada, somar o *peso* de MUX em causa (entre '0', '4', '8' e '12') com a *identificação* (entre '0', '1', '2' e '3') dessa entrada…

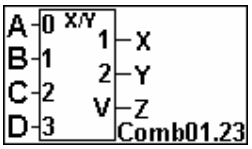
Enfim, para que tudo funcione como desejável, as entradas EN (*enable*) terão, é claro, que ficar alimentadas com '**1**'

Parêntesis de alerta aos distraídos: convém um parêntesis análogo ao incluído em [10E4.4]. É que se poderiam conceber outras ligações - como seja, por ex., a aplicação de {s_0, s_2} ao MUX da direita e de {s_1, s_3} aos restantes MUX. Qual seria então a identificação da entrada '**3**' de M_{00}? Repare-se: ela ficaria seleccionada para {s_0=**0**, s_2=**0**} e {s_1=**1**, s_3=**1**} – ou, reordenando: {s_3=**1**, s_2=**0**, s_1=**1**, s_0=**0**} – do que se segue que deveria ser identificada por d_{10}. Estas 'contas' são bem mais *tortuosas* do que a acima referida "soma do *peso* do MUX com a *identificação* da entrada" – daí a conveniência em repetir o aviso à navegação já algures enunciado a propósito da escolha das variáveis a aplicar às entradas de selecção: *recusar soluções tortuosas*, pelo artifício de aplicar a variável de *menor* peso à entrada de selecção de *menor* peso, aplicar a de *maior* peso à entrada de selecção de *maior* peso, etc.

A disposição em Comb01.22 é fruto também da atenção tida na *distribuição geográfica* dos *multiplexers*:

- M_{10} e M_{11} ficam *alinhados à esquerda*: os MUXs da *coluna* da esquerda são seleccionados quando s_3=1 (e os da que lhe fica à direita são seleccionados quando s_3=0).
- M_{10} e M_{00} ficam *alinhados no topo*: os MUXs da *linha* do topo são seleccionados quando s_2=0 (e os da que lhe fica por baixo são seleccionados quando s_2=1).

8. [11P4.1] Considere o *codificador de prioridade* da figura Comb01.23, em que a entrada 3 é a entrada de maior prioridade. Preencha a tabela abaixo com os valores lógicos nas saídas para as combinações de entrada indicadas.



A	B	C	D	X	Y	Z
0	1	0	0			
0	1	0	1			
0	0	0	0			
0	0	0	1			

R: A saída Z será '1' sse estiver activa alguma das entradas, pelo que será '0' apenas para a combinação '0000'

Sse Z estiver activa, as restantes saídas disponibilizam o código da entrada que estiver activa ou, se for o caso de estarem activas duas ou mais, o código daquela que tiver maior prioridade. Será então:

A	B	C	D	X	Y	Z	
0	1	0	0	1	0	1	← Só B/1 é que está activa
0	1	0	1	1	1	1	← B/1 e D/3 estão ambas activas, mas D/3 tem maior prioridade
0	0	0	0	X	X	0	← Nenhuma das entradas está activa
0	0	0	1	1	1	1	← Só D/3 é que está activa

N.B. Quando B está activa, as saídas {X,Y} devem apresentar a codificação de '1', pois é à entrada '1' que B está ligada. As saídas {X,Y} tem os pesos {1,2}, respectivamente; com esses pesos, '1' escreve-se {1,0}...

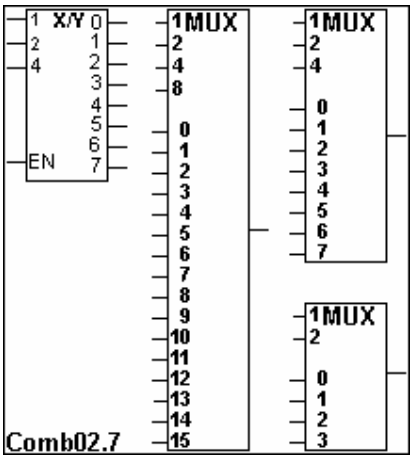
9. [11P4.1] Pretende-se um circuito para testar os conhecimentos dos estudantes de Sistemas Digitais acerca de *gates* (portas lógicas) de 2-entradas. O circuito tem 5 entradas:

   - duas entradas $\{S_1S_0\}$, com que se escolhe a *gate* sobre a qual incide o teste, e que pode ser uma das seguintes: AND, NAND, XOR e NOR;
   - duas outras entradas $\{I_1I_0\}$, para especificar as entradas dessa *gate*;
   - uma entrada onde o examinando insere o valor que acha ser a saída da *gate* para esse par de entradas $\{I_1I_0\}$.

   O circuito tem uma saída que ficará activada sse o estudante acertou...

   Realize o circuito com os *decoders* ou *multiplexers* adiante indicados e o número mínimo de portas lógicas adicionais:

   9. 1. descodificadores 3:8 com *enable*
   9. 2. um multiplexer 16:1
   9. 3. multiplexers 8:1 e 4:1



R: A Tabela de verdade, deitada para ocupar menos espaço, e codificando "Resposta certa" ↔ '1', vem a ser a seguinte:

AND				NAND				XOR				NOR				
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	**E**
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	**I0**
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	**I1**
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	**S0**
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	**S1**
1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	

0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	**E**
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	**I0**
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	**I1**
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	**S0**
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	**S1**
1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	

Processo mental: as primeiras 8 células à esquerda correspondem a combinações em que S_1S_0={00}, que quer dizer: está-se testando os conhecimentos dum estudante acerca da *gate* AND. Dessas, as primeiras 2 células correspondem a combinações em que I_1I_0={00}, que quer dizer: está-se inquirindo ao estudante a resposta à seguinte pergunta: "se a uma porta AND se aplicarem as entradas {00}, qual será a saída?" Ele pode responder '0' ou '1'; no primeiro caso, a resposta está certa, no segundo está errada; pelo que as duas primeiras posições da tabela de verdade advêm preenchidas com **10**... Para as demais células, o raciocínio é semelhante

Ou seja: $Z = \Sigma\, m\ (0, 2, 4, 7, 9, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 30)$

R1: Uma solução encontra-se em Comb02.12.

Processo mental: Da expressão de Z na notação Σ, depreende-se a necessidade de "algo" que produza os mintermos {m0, m2, ..., m30}. Ora um *decoder* é precisamente um circuito que produz todos os mintermos que se podem formar a partir das suas variáveis de entrada... Porquanto são 5 as variáveis de entrada, conviria ter à mão um *decoder* 5:32... Mas é coisa que o enunciado não pressupõe – pelo que há que o materializar a partir do que se tem, a saber: *decoder*s 3:8...

Um *decoder* 3:8 tem 8 saídas, um *decoder* 5:32 tem 32; a primeira coisa a fazer é construir um *plano* vertical de 4 (=32/8) *decoders*, *vidé* Comb02.12.

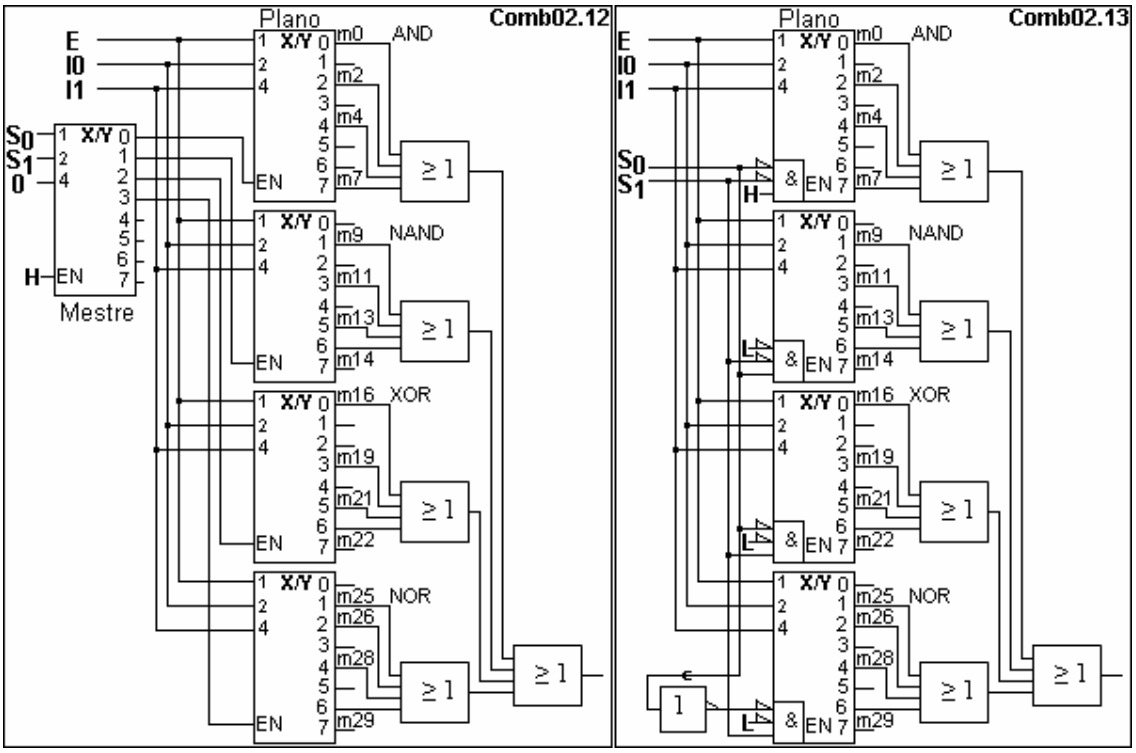
Por comodidade, conviria associar cada um deles a uma operação específica... Para o efeito, inclui-se um *decoder Mestre*, à esquerda, cujas saídas {0, 1, 2, 3} serão ligadas às entradas *ENable* do Plano: ligando $\{S_1S_0\}$ às entradas {1,2} do Mestre (e fixando 0 na entrada '4'), observar-se-á o seguinte: quando $\{S_1=0,S_0=0\}$ a saída '0' do Mestre (*e só ela*) fica activa, pelo que o *único decoder* do Plano que ficará *enabled* é o do topo (isto é, aquele que se pretende associar à operação AND); asserções análogas valem para as demais combinações de $\{S_1S_0\}$.

Também por comodidade, associam-se as entradas $\{I_1I_0E\}$ ás entradas {1, 2, 4} de cada *decoder* do Plano. Com isso: quando $\{I_1=0,I_0=0,E=0\}$ a saída '0' será a *única* que ficará activa...

Conjugando os dois efeitos, de $\{S_1S_0\}$ e $\{I_1I_0E\}$, ter-se-á:

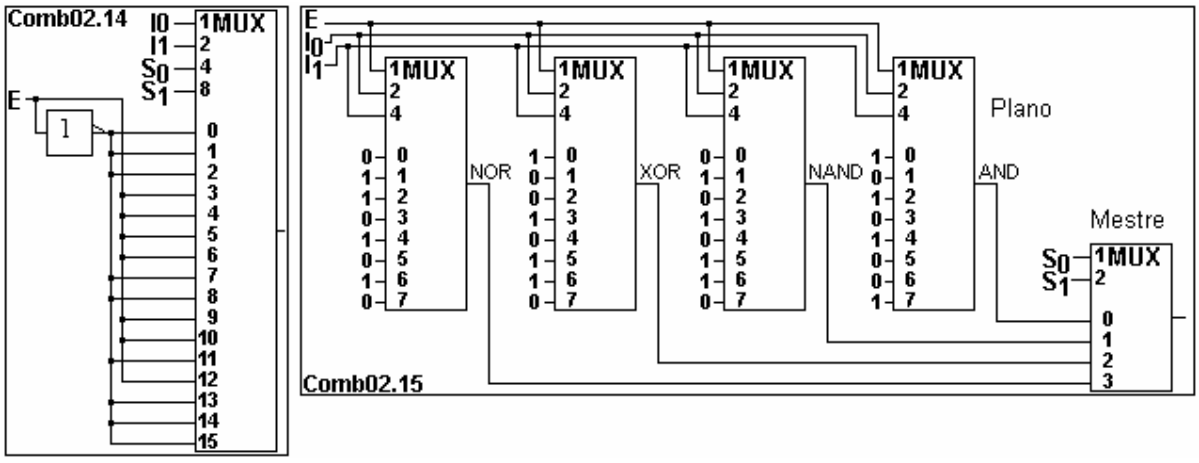
- quando $\{S_1=0,S_0=0,I_1=0,I_0=0,E=0\}$, a *única* saída que ficará activa será a saída '0' do *decoder* 'AND': ela representa o mintermo m0;
- quando $\{S_1=0,S_0=1,I_1=0,I_0=0,E=0\}$, a *única* saída que ficará activa será a saída '0' do decoder 'NAND': ela representa o mintermo m8...

Que resta fazer? Recordando a primeira forma canónica de materializar uma função, bastará um OR cujas entradas sejam as saídas dos *decoders* que geram os 16 mintermos que a compõem, ou, como a figura testemunha, uma árvore de 4+1 ORs...



Nota: Se se dera o caso de usar *decoders* 3:8 com 3 entradas *Enable*, de que um ex. é a chip SN74138, poder-se-ia, com uma gestão inteligente dessas entradas e um inversor, dispensar o decoder Mestre, *vidé* Comb02.13

R2: Uma solução encontra-se em Comb02.14



Justificação: A tabela de verdade envolve 32 combinações de 5 variáveis, todavia o multiplexer aceita somente 16 entradas de dados. Isso é um convite a *condensar* a tabela de verdade... A partir da primeira, constrói-se uma outra tabela de verdade com 4 variáveis, precisamente as de maior peso:

AND				NAND				XOR				NOR				
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	**I0**
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	**I1**

0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	**S0**
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	**S1**
$\overline{E}$	$\overline{E}$	$\overline{E}$	E	E	E	E	$\overline{E}$	$\overline{E}$	E	E	$\overline{E}$	E	$\overline{E}$	$\overline{E}$	$\overline{E}$	

Processo mental: *a* célula {0} – que corresponde a {S_1=0, S_0=0, I_1=0, I_0=0} – condensa o *par* de células {0,1} da tabela de verdade (não-condensada). Nesse par, {S_1, S_0, I_1, I_0} assumem os mesmos valores {0, 0, 0, 0}, mas E é **0** na primeira e **1** na segunda. Com isso, a saída volve-se **1** na primeira e **0** na segunda, isto é: *nesse* par de células, a saída é o *inverso* de E; isso justifica inscrever $\overline{E}$ na célula {0} da tabela condensada... Raciocínio análogo vale para as células {1, 2}... Já para a célula {3}, a saída advém **0** quando E é **0**, e advém **1** quando E é **1**: isso justifica inscrever E na célula {3} da tabela condensada...

R3: Uma solução encontra-se em Comb02.15

Processo mental: estando disponíveis multiplexers 8:1, a primeira coisa a fazer é construir um *plano* horizontal de 4 (=32/8) *multiplexers*.

Por comodidade, convirá associar cada um deles a uma operação específica... Para o efeito, inclui-se um *multiplexer Mestre*, à direita, cujas entradas {0, 1, 2, 3} serão ligadas às saídas do Plano: ligando {S_1S_0} às entradas de selecção {1,2} do Mestre, observar-se-á o seguinte: quando {S_1=0,S_0=0} é o valor à sua entrada '0' que será transmitido para a saída, pelo que será o *multiplexer da direita* aquele que ficará associado à operação AND; asserções análogas valem para as demais combinações de {S_1S_0}.

Também por comodidade, associam-se as entradas {I_1I_0E} ás entradas {1, 2, 4} de cada *multiplexer* do Plano. Com isso: quando {I_1=0,I_0=0,E=0} é o valor à entrada '0' que será transmitido para a saída...

Conjugando os dois efeitos, de {S_1S_0} e {I_1I_0E}, ter-se-á:

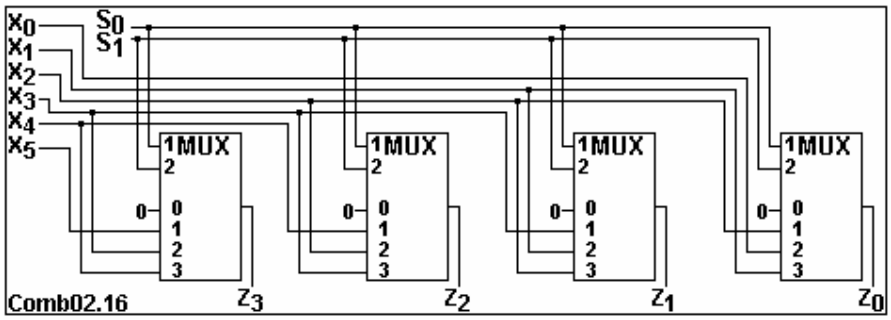
- quando {S_1=0,S_0=0,I_1=0,I_0=0,E=0}, o valor que surge na saída do circuito será aquele que estiver ligado à entrada '0' do multiplexer da direita; isso justifica que ai se force o valor '1'... O resto intui-se...

Nota: Em vez o Mestre ser um *multiplexer* 4:1, poder-se-ia usar, como Mestre, um multiplexer 8:1, e constituir o Plano com multiplexers 4:1...

10. Desenhe um sistema com 6 entradas de dados {X5, ..., X0}, duas entradas de selecção {S1S0} e quatro saídas {Z3,...,Z0}, com o seguinte comportamento:

{Z3,...,Z0} = {X5, ..., X2} se {S1S0}={01}
= {X3, ..., X0} se {S1S0}={10}
= {X4, ..., X1} se {S1S0}={11}
= {0,0,0,0} se {S1S0}={00}

R. Uma solução encontra-se em Comb02.15

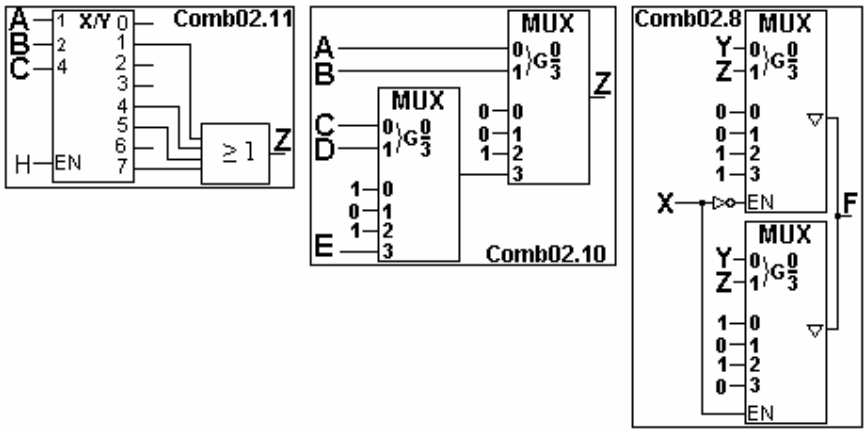


Processo mental: para clareza, convirá escrever as condições do problema sob a forma de uma tabela...

0	**0**	0	0	0	0
0	**1**	X5	X4	X3	X2
1	**0**	X3	X2	X1	X0
1	**1**	X4	X3	X2	X1
S1	**S0**	**Z3**	**Z2**	**Z1**	**Z0**

Para Z0, por ex., a saída será {0, X2, X0, X1} conforme respectivamente {S1,S0} forem {0, 1, 2 ou 3}. Isso remete para um *multiplexer* de 4 entradas com variáveis de selecção {S1,S0} e a cujas entradas de dados se aplicam precisamente {0, X2, X0, X1}

11. Escreva as expressões algébricas das saídas dos circuitos
    11. 1. Comb02.11
    11. 2. Comb02.10
    11. 3. Comb02.8



R1: $Z = \sum_{CBA}(m1, m4, m5, m7) = \overline{B}\,A + C\,\overline{B} + C\,A$

R2: $Z = \sum_{BA}(m2, m3 * \sum_{DC}(m0,m2,m3 * E)) = B\,\overline{A} + B\,A\left(\overline{D}\,\overline{C} + D\,\overline{C} + E\,D\,C\right) = B\left(\overline{A} + \overline{C} + E\,D\right)$

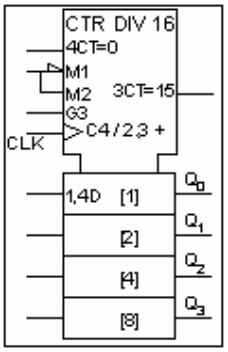
R3: $Z = \overline{X}\ \sum_{ZY}(m_2, m_3) + X\sum_{ZY}(m_0, m_2) = \overline{X}\,Z + X\,\overline{Y}$

Contadores

{CTR_r.doc}

Preâmbulo:

P1: ***CTR DIV 16***



Como o denota a mnemónica, CTR DIV 16 (*Counter*), o circuito figurado vem a ser um Contador de módulo 16: ao ritmo dos impulsos de relógio, conta *ciclicamente* entre '**0**' e '**15**':

0000, 0001, 0010, ..., 1110, 1111, 0000, 0001, ...

Tal contagem é apresentada nas saídas {Q_3, Q_2, Q_1, Q_0}; conforme aos valores entre parêntesis '[]', Q_3 é a de *maior* peso, '**8**', e Q_0 a de *menor* peso, '**1**'.

Para o sinal de relógio, CLK, é disponibilizada uma entrada – nomeada C**4**; o contador reage ao flanco ascendente do CLK: repare-se no símbolo (▷) que acompanha a entrada C4. O ***sufixo*** '**4**' significa que *todas as entradas com* ***prefixo '4'*** *estão sincronizadas pelo sinal de relógio*.

Além de *contar*, o contador viabiliza o *reinício da contagem a partir de um valor arbitrário* escolhido pelo projectista. Formalmente, diz-se que o contador tem 2 *modos de funcionamento* {M_1 e M_2}: quando a entrada M_1 está *Low*, activa-se M_1; caso contrário é activado M_2.

O que M1 e M2 significam (*contagem* ou *reinício*?) é determinado pelos ***prefixos*** de D e de '+':

M_1	M_2	Modo	Funcionamento	
Low	*High*	**1**	é feito o carregamento *em paralelo* do contador	*vidé* notação '**1**,4D' nas 4 entradas D
High	*Low*	**2**	o conteúdo do contador incrementa (Se G3=*High*)	*vidé* notação em C4: '/**2**,3+'

Em ordem a proporcionar o *reinício da contagem* a partir dum valor arbitrário - não necessariamente '0000', o contador disponibiliza 4 entradas de *dados* '**4**D', todas elas sincronizadas pelo CLK: basta aplicar tal valor a essas entradas e provocar um impulso de relógio (com '**0**' na entrada 4CT).

Repare-se que, dessas 4 entradas, apenas a superior se encontra referenciada 1,4D: subentende-se que as que lhe estão por baixo exibem um comportamento idêntico, não há que *repetir* a símbólica…

O contador figurado tem ainda:

- uma entrada *Enable* (denotada por G**3**); este ***sufixo*** '**3**' significa que *todas as entradas/saídas com* ***prefixo '3'*** *'funcionam' somente se G****3*** *estiver High*; em particular, a contagem acontece somente se G3 estiver *High*, *vidé* notação C4: '/2,**3**+';
- uma entrada de *Reset síncrono* (denotada por **4**CT=0): se estiver *High* no CLK seguinte, o contador retorna a {0000} (se a denominação da entrada fosse CT=0, então tratar-se-ia de um Reset *assíncrono*: é o ***prefix*****o** '**4**' em 4CT=0 que indica que o Reset está *sincronizado* pela entrada de relógio, C4);
- uma saída, denotada **3**CT=15, que advém *High* quando o contador atinge o valor limite 15 (1111), e G**3** se encontra *High*.

Parêntesis: M1 e M2 *não* significam *dois* pinos distintos! *O pino é só um* - porém com duas designações. Usa-se M1 ou M2 consoante se lhe for aplicado, respectivamente *Low* ou *High*. Claro é que, em vez de M1 ou M2, se poderia ter recorrido a, por ex., $\overline{\mathrm{M}}$ e M. Todavia, isso não teria o alcance de M**1** e M**2**: estes nomes, porquanto fazem entrar em jogo os *sufixos* '**1**' e '**2**', *são muito mais informativos*. Usando-os nas entradas '**1**,4D' e em '/**2**,3+', fica-se com uma visão clara de qual a finalidade de cada Modo. ***Sufixos*** (e ***prefixos***) *não estão lá para enfeitar!* Assim, antes de tudo há que determinar quais os sufixos/prefixos usados (no caso: '1', ...,'4') – e interpretá-los!

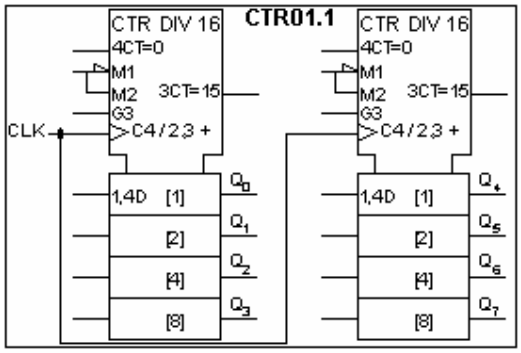
P2: ***CTR DIV 8***

CTR DIV 8 significa um Contador de módulo 8: ao ritmo do relógio, conta *ciclicamente* entre '**0**' e '**7**'

(Para o entendimento das demais etiquetas, *vidé* P1: o autor dispensa-se de o voltar a repetir...)

(Salvo indicação em contrário, neste documento pressupõe-se o uso de *lógica positiva*, isto é, a correspondência: {'**0**'→**Low**, '**1**'→**High**})

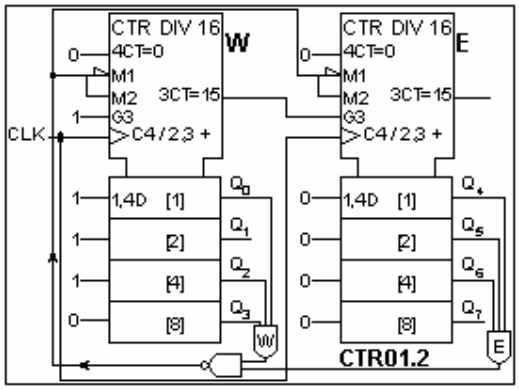
1. [10T2.3,10E1.7] A partir do circuito em CTR01.1, concretize um contador binário que conte ciclicamente entre 7 e 125.



R: O contador pedido encontra-se esquematizado em CTR01.2

Justificação: os dispositivos dados *contam ciclicamente* entre 0 e 15. Em ordem a contar até, ao menos, 125, há que *interligar dois deles,* sejam **W** e **E**: 16*16 = 256, o que é bastante para cobrir 125. Isso faz-se interligando a saída 3CT=15 de **W** à entrada G3 de **E**: só *depois* de **W** atingir o fim do ciclo (15) é que **E** incrementa mais '**1**' (similar ao que ocorre num vulgar relógio de pulso: somente *depois do* ponteiro dos segundos perfazer uma volta completa – isto é, marcar 59 – é que o ponteiro dos minutos avança…); naturalmente, é também preciso ligar a *High* a entrada G**3** de **W**, que só assim é que é **3**CT chega a ser *High*…

Depois de o conjunto atingir 125 (em binário: 0111 1101), deve carregar-se 7 (em binário: 0000 0111), para que a contagem recomece novamente – a partir de 7… Há que ter cuidado, porém – que, com as ligações feitas, o bit *mais* significativo é Q_7, e o *menos* significativo é Q_0: *o contador que roda mais lentamente é* **E***!* Em termos práticos, 125 corresponde a **W** marcar 1101 e **E** marcar 0111, e 7 corresponde a marcarem 0111 e 0000, respectivamente. Assim:



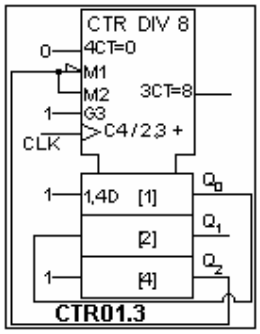
- aplicam-se, às entradas de *dados*, os valores 0111 e 0000 – para os carregar em paralelo, quando for caso disso;

- aplica-se, em M1 (de **W** e **E**), um valor que é '**0**' só quando os contadores marcarem 1101 e 0111 – o que na prática se pode lograr com o NAND e o par de ANDs incluídos.

(Note-se que no AND-W não é preciso aplicar-lhe $\overline{Q_1}$ (pois que, estando o contador **E** a 0111, **W** não chega a exceder 1101); e no AND-E não é preciso aplicar-lhe $\overline{Q_7}$ (pois **E** não chega a contar 1000).

O que irá suceder é o seguinte: quando os contadores marcarem 1101 e 0111 (isto é, 125), e só então, *ambos* os ANDs se volvem em '1'; e a saída do NAND advém '**0**'; com isso, o modo de ambos os contadores passa a ser M1; no impulso CLK *seguinte*, é feito o carregamento em paralelo, isto é, em vez de os contadores incrementarem, para 126, regressam a 7.

2. [10E2.8] Considere o circuito representado na figura CTR01.3, no qual é utilizado um contador. Considere como estado inicial Q2=Q1=Q0=0. Qual o ciclo de contagem efectuado por este circuito?



R: 0→5→6→7→0…

Justificação: Tratando-se de um contador de 3-bit, só pode assumir valores da gama {000, 001, 010, 011, 100, 101, 110, 111}. O modo M_1 é activado quando Q_2=0: enquanto Q_2=1 (isto é, enquanto o contador marcar um valor da gama {**1**00, **1**01, **1**10, **1**11]), está activo o modo M_2. A passagem para Q_2=0 ocorre quando o contador passa a marcar 0/'000' (após já ter marcado 7/'111'). M_1 é então activado – o que significa que, no impulso CLK *seguinte*, será feito o carregamento em paralelo do que se encontrar nas 3 entradas de dados; acontece que, quando o contador marca '000', o que se encontra nessas entradas de dados é '101', isto é: 5. Depois de marcar 0, o contador carregará 5, depois irá incrementando: 6, 7, 0 – caso em que voltará de novo a carregar 5…

3. [10E4.7] A partir do circuito indicado em CTR01.1, concretize um contador BCD que conte ciclicamente entre $00_{(BCD)}$ e $99_{(BCD)}$.

R: CTR01.5 esquematiza uma solução.

Justificação: os dispositivos dados *contam ciclicamente* entre 0 e 15 (*vidé* Preâmbulo P1]). Em ordem a contar em BCD, '**00**' até '**99**', há que:

1) forçar o *retorno a '0'* de cada contador quando for altura de ele "progredir para '**10**'"); isso faz-se com dois ANDs:

- um primeiro AND entre Q_3Q_0 do contador **W**: a sua saída será '**1**' quando ele marcar '1001' (isto é, tiver atingido o valor 9);

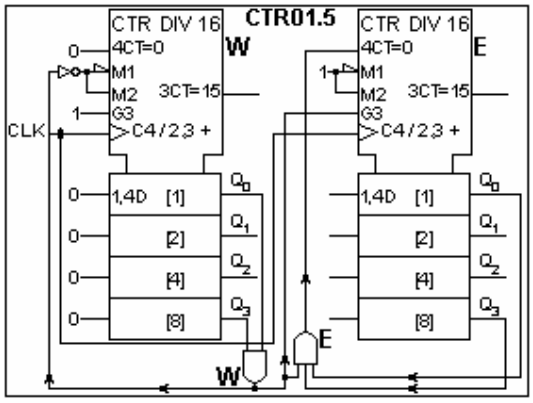
- um segundo AND cuja saída seja '**1**' quando *ambos* os contadores marcarem '1001' (isto é, tiverem atingido o valor 9);

Seguem-se duas tácticas de *uso* da saída desses ANDs:

1a. A saída do AND, após *complementada*, é ligada à entrada M1 (*vidé* contador **W**): ao atingir '**9**', M1 advém '**0**'; com isso, é activado o Modo M1 - que corresponde ao *carregamento em paralelo* do contador; visando o seu retorno a '**0**', bastará fixar em '**0**' as entradas **1**,4D…

1b. A saída do AND é ligada à entrada 4CT=0 (*vidé* contador **E**): ao se atingir '**99**', essa entrada advém '**1**'- com o que será feito o *reset* do contador (é o que significa CT=0);

2) interligar a saída do AND **W** à entrada G3 de **E**: só após o contador **W** atingir '**9**' é que **E** incrementará; naturalmente, é também preciso ligar a *High* a entrada G**3** de **W**, que só assim é que **W** incrementa (vidé especificação na entrada CLK: /2,**3+**)…



Notas:

1) tanto o carregamento em *paralelo* de '0000' (em **W**) como o *reset* (em **E**) são *síncronos*, só acontecem no impulso de relógio *seguinte* (quando seria altura de passar a '10');

2) *não deve "deixar-se no ar" nenhuma entrada usada* (dos contadores) – daí a fixação da entrada 4CT de **W** (em '**0**') e M1 de **E** (em '**1**').

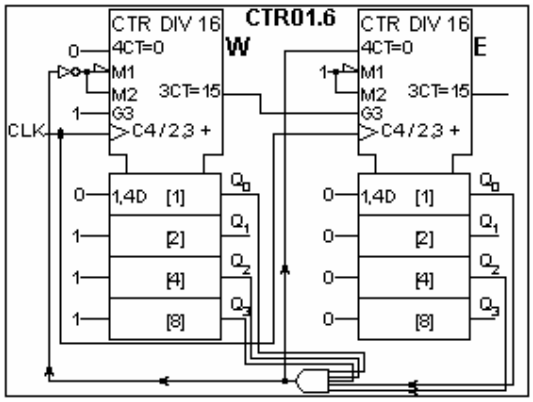
3) O AND **W** necessita das entradas Q_3Q_0, mas *dispensa* as entradas (complementadas) Q_2Q_1: adiante, apresenta-se o quadro de Karnaugh (das 4 entradas $\{Q_3Q_2Q_1Q_0\}$ para a função que se volve em '**1**' só quando o contador **W** marca '**9**' – e se sabe *que ele depois não chega nunca a marcar* '10'…'15'; aplicando o método de Karnaugh, a expressão simplificada dessa função vem a ser Q_3Q_0. Considerações análogas valem para o AND **E**.

	0	0	1	1	Q_1
	0	1	1	0	Q_0
00	0	0	0	0	
01	0	0	0	0	
11	x	x	x	x	
10	0	**1**	x	x	
Q_3Q_2					

4. [10E3.7] A partir do circuito em CTR01.1, concretize um contador binário síncrono que conte ciclicamente entre 14 e 93.

R: O contador pedido encontra-se esquematizado em CTR01.6.

Justificação (*vidé* Considerandos de [10E1.7]): os dispositivos dados *contam ciclicamente* entre 0 e 15. Em ordem a contar até, ao menos, 93, há que *interligar dois deles,* sejam **W** e **E**: 16*16 = 256, o que é bastante para cobrir 93. Isso faz-se interligando a saída 3CT=15 de **W** à entrada G3 de **E**: só *depois* de **W** atingir o fim do ciclo (15) é que **E** incrementa mais '**1**'; naturalmente, é também preciso ligar a *High* a entrada G3 de **W** …



Depois de o conjunto atingir 93 (em binário: 0101 1101), deve carregar-se 14 (em binário: 0000 1110), para que a contagem recomece novamente – a partir de 14… Com as ligações feitas, o bit *mais* significativo é Q_7, e o *menos* significativo é Q_0: *o contador que roda mais lentamente é* **E***!* Em termos práticos, 93 corresponde a **W** marcar 1101 e **E** marcar 0101, e 14 corresponde a marcarem 1110 e 0000, respectivamente. Assim:

- aplicam-se, nas entradas de dados de W e E, os valores 1110 e 0000 – para os carregar em paralelo, ao se atingir '93';

- no sentido de assegurar que **W** passe a marcar 1110 (e, não, 1110), ao atingir-se 93: aplicar, em M1 (de **W**), um valor que é '**0**' só quando os contadores marcarem 1101 e 0101 – o que na prática se pode lograr com o NOT e o AND incluídos;

- no sentido de assegurar que **E** passe a marcar 0000 (e, não continue em 0101), ao atingir-se 93: pode seguir-se uma filosofia análoga (ligar a saída do NOT à entrada M1 de E) ou, como se fez em CTR01.6, ligar a saída do AND à entrada 4CT (Reset síncrono), e manter a entrada M1/M2 permanentemente a '1' (neste caso, às entradas de dados de E para carregamento em paralelo não é necessário ser-lhes aplicado '0')

(Note-se que no AND não é preciso aplicar-lhe $\overline{Q_1}$ (pois que, estando o contador **E** a 0101, **W** não chega a exceder 1101); e não é preciso aplicar-lhe $\overline{Q_7}$ nem $\overline{Q_5}$ (pois **E** não chega a contar 0110).

O que irá suceder é o seguinte: quando os contadores marcarem 1101 e 0101 (isto é, 93), e só então, *o* AND volve-se em '1'; e a saída do NOT advém '**0**'; com isso, o modo do contador W passa a ser M1; no impulso CLK *seguinte*, é feito o carregamento em paralelo, isto é, em vez de ele incrementar, para 1110, regressa a 14.

Sistemas de Numeração

{Numeric_r.doc}

Preâmbulo:

P1: Representações em binário – de ***4-bit***! - dos números {0 a 15}:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

P2: Representações em binário – de ***3-bit***! - dos números {0 a 7}:

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

P3: Representação em base 16 (hexadecimal) dos primeiros 16 inteiros:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

P4: Primeiras 11 potências da base 2:

2^{10}=1024 2^{9}=512 2^{8}=256 2^{7}=128 2^{6}=64 2^{5}=32 2^{4}=16 2^{3}=8 2^{2}=4 2^{1}=2 2^{0}=1

1. [10E2.1] Sejam x = 23 e y = 35.
   1. 1. Determine a representação em binário de x e y.

R1: Posto que $23_{(10)} = 17_{(16)} \Rightarrow 23_{(10)} = 1\ 0111_{(2)}$
Posto que $35_{(10)} = 23_{(16)} \Rightarrow 35_{(10)} = 10\ 0011_{(2)}$

1ª etapa: Eis *três* métodos de se conseguir a representação em binário de **x** e **y**:

1-1-a) o método *mais rápido*: por divisões sucessivas por 16, seguida de conversão para 2:

```
23 |16        35 |16
 7  1         3  2
```

A divisão 23 : 16 dá quociente 1 e resto 7; então, $23_{(10)} = 17_{(16)}$. ***(Confirmação**: 1*16+7=23);*
A divisão 35 : 16 dá quociente 2, resto 3; então, $35_{(10)} = 23_{(16)}$. ***(Confirmação**: 2*16+3=35).*
Trata-se agora de converter, para binário, $17_{(16)}$ e $23_{(16)}$. Para isso, recordem-se as representações em binário – de ***4-bit***! - dos números {0 a 15}. A substituição de cada um dos símbolos presentes em $17_{(16)}$ e $23_{(16)}$ pelo respectivo *tetra-bit* (*vidé Preâmbulo-1*) conduz a (após se descartarem os '0's à esquerda):

$$17_{(16)} = 1\ 0111_{(2)} \text{ e } 23_{(16)} = 10\ 0011_{(2)},$$

> *Notas importantes:*
> *1) Apenas para* comodidade na leitura*, as sequências de bits aparecem, neste texto, repartidas em grupos de 4-bit, com um espaço entre eles, isto é: escreve-se 1 0111 e, não, 10111; e escreve-se 10 0011 e, não, 1000011;*
> *2) Se não for explicitada a base de escrita dum número, subentende-se que ela é 10: escreve-se +23 e, não, $+23_{(10)}$*

Confirmação: antes de prosseguir, convém duvidar: estará certo o resultado obtido – ou ter-se-á cometido algum erro? Para o discernir construa-se *mentalmente* a tabela, cujas colunas estão etiquetadas pelas sucessivas *potências de 2* (*vidé Preâmbulo-4*) - sendo que a de *menor peso* (a menos significativa) é aquela que se encontra *mais à direita*:

Pesos:	**32**	**16**	**8**	**4**	**2**	**1**
$17_{(16)}$		1	0	1	1	1
$23_{(16)}$	1	0	0	0	1	1

Os números **1 0111**$_{(2)}$ e **10 0011**$_{(2)}$ foram *alinhados à direita*. Somando os *pesos* das colunas em que surgem '1's:
$17_{(16)} \Rightarrow$ **16 + 4 + 2 + 1** = 23 (que é o *módulo* de x)

$23_{(16)} \Rightarrow$ **32 + 2 + 1** = 35 (que é o *módulo* de y)

Repare-se: a partir dos *resultados* obtidos ($1\ 0111_{(2)}$ e $10\ 0011_{(2)}$), regressou-se aos valores dados (23 e 35). Isso predispõe a confiar que eles estarão correctos.

Comentário a propósito: revendo esta confirmação, entrevê-se um outro método de lograr a representação em *binário* de um inteiro. Tanto 23 como 35 acabam por se exprimir em potências da base 2… Isso é generalizável a qualquer inteiro positivo – pelo que o problema se resume a isto: como decompor um inteiro positivo numa soma de potências-de-2?

Seja o caso do inteiro 35: começa-se por procurar a maior potência de 2 que *não excede* 35 – que é 32. Subtraindo, obtém-se 35-32=3 (ou seja: 35=32+3); procura-se agora a maior potência que não excede 3 – que é 2. Subtraindo, fica 3-2=1 (ou seja: 35=32+2+1); procura-se agora a maior potência que não excede 1 – que é justamente 1. Concluindo: 35 = 32 + 2 + 1, ou, exprimindo em potências de 2, $35 = 2^5 + 2^1 + 2^0$. Quer dizer: a representação binária de 35 deve ter '**1**'s nas posições **5**, **1** e **0** – e '**0**'s nas restantes (4, 3, 2), isto é, deve ser: $35_{(10)}=10\ 0011_{(2)}$. O algoritmo é trivial – mas não é lá muito recomendável para números grandes; já para números pequenos, como sejam os dos Preâmbulos P1 e P2, pode (e deve) ser aplicado *mentalmente*!

(Apenas para o leitor testar a sua *agilidade mental*: qual a representação binária de $12345_{(10)}$ - passando pela base 32, e decompondo as componentes do resultado na base 32 em somas de potências de 2?)

1-1-b) o método não tão rápido: por divisões sucessivas por 8, seguida de conversão para 2:

23 | 8	35 | 8
7 2	3 4

A divisão 23 : 8 dá quociente 2 e resto 7; então, $23_{(10)} = 27_{(8)}$. ***(Confirmação**: 2*8+7=23);*

A divisão 35 : 8 dá quociente 4, resto 3; então, $35_{(10)} = 43_{(8)}$. ***(Confirmação**: 4*8+3=35).*

Trata-se de converter, para binário, $27_{(8)}$ e $43_{(8)}$. Para isso, recordem-se as representações em binário – de ***3-bit***! - dos números {0 a 7}. A substituição de cada símbolo presente em $27_{(8)}$ e $43_{(8)}$ pelo respectivo *tri-bit* (*vidé Preâmbulo-2*) leva a:

$27_{(8)} = 10\ 111_{(2)}$ e $43_{(8)} = 100\ 011_{(2)}$.

1-1-c) o método *mais moroso*: por divisões sucessivas por 2:

23	2				35	2			
03	11	2			15	17	2		
1	1	5	2		1	1	8	2	
		1	2	2			0	4	2
			0	1				0	2	2
									0	1

A divisão 23 : 2 dá quociente 11 e resto 1; a divisão 11 : 2 dá quociente 5 e resto 1; a divisão 5 : 2 dá quociente 2 e resto 1; a divisão 2 : 2 dá quociente 1 e resto 0; então, $23_{(10)} = 1\ 0111_{(2)}$.

A divisão 35 : 2 dá quociente 17 e resto 1; a divisão 17 : 2 dá quociente 8 e resto 1; a divisão 8 : 2 dá quociente 4 e resto 0; a divisão 4 : 2 dá quociente 2 e resto 0; a divisão 2 : 2 dá quociente 1 e resto 0; então, $35_{(10)} = 10\ 0011_{(2)}$.

Como seria de esperar, as representações logradas pelos vários métodos acima são exactamente as *mesmas*, a saber:

$23_{(10)} =$ **1 0111**$_{(2)}$ e $35_{(10)} =$ **10 0011**$_{(2)}$.

Importante: quando se usa o *primeiro* método, "por divisões sucessivas por 16", é necessário um cuidado especial. Seja, como exemplo para ajudar a clarificar, a conversão para binário de $202_{(10)}$. A divisão 202 : 16 volve-se em:

202 | 16
42 12
10

Seria *errado* escrever $202_{(10)}=1210_{(16)}$: $1210_{(16)}$ vale $1*16^3 + 2*16^2 + 1*16^1 = 4624$, e, não, 202! O correcto é $202_{(10)}=CA_{(16)}$ – em que se usam os símbolos (C e A) que, na base 16, valem 12 e 10, *vidé Preâmbulo-3*.

Todavia, nada impede imaginar *mentalmente* $202_{(10)}=12.10_{(16)}$ – que, para os menos distraídos, lembra o modo, dito de "*dotted decimal*", usado para representar endereços-IPv4 da Internet. A partir daí, a conversão é trivial: $202_{(10)}=1100\ 1010_{(2)}$

Nota: a representação em binário foi obtida por *três* métodos; mas há mais… Desafio ao leitor: como procederia para lograr a representação em binário por divisões sucessivas por 4? E já agora: porque é que divisões sucessivas por 8 e 16 aceleram a obtenção do resultado *correcto* – mas divisões sucessivas por, por exemplo, 5 ou 6, dão mau resultado?

2. [10T1.1] Considere o número hexadecimal B0A.
   2. 1. Converta-o para base 2.
   2. 2. Quanto vale o número indicado em decimal?

R1: $B0A_{(16)} = 1011\ 0000\ 1010_{(2)}$.
(Pois que $16=2^4$, substitui-se cada símbolo da base 16 pelo seu equivalente – com 4-bit – da base 2, *vidé* Preâmbulo P3))
R2: $B0A_{(16)} = 11*16^2+10 = 11 * 256 + 10 = 2816 + 10 = 2826_{(10)}$.
Confirmação: $1011\ 0000\ 1010_{(2)} = 2^{11} + 2^9 + 2^8 + 2^3 + 2^1 = 2048 + 512 + 256 + 8 + 2 = 2826_{(10)}$.
(A conversão dum número em base 2 (ou 16) para a base 10 é trivial: é uma soma ponderada de potências dessa base)

Nota: Repare o leitor na praxis de *confirmar* cada etapa: ela não é supérflua, convém mesmo fazê-lo - nem que seja para evitar que as respostas às interrogações finais *"o resultado é válido? e qual o número mínimo...?"* fiquem de antemão ameaçadas de insucesso. Dada a facilidade e rapidez em cada confirmação, não há a menor desculpa para que as respostas a tais interrogações estejam erradas porque se basearam no resultado erróneo de uma *simplérrima* adição…

Ainda que aparente ser puro desperdício de tempo, *Confirmar um resultado* cai por conseguinte sob o signo da ***lei do menor esforço***: fazer (***bem***) a disciplina em, apenas, ***um*** semestre...

Números com Sinal

{SignedN_r.doc}

1. [11P5.1] Admita que pretende representar valores em notação de complemento para 2 com 8 bits
   1. 1. Qual a gama de valores que é possível representar assim?
   1. 2. Represente, em notação de complemento para 2 com 8 bits, os seguintes números decimais:
   + 57; - 57; +70; - 70; + 127; – 127; – 128.

R1: {-128 a +127)
Um código c2 com 8 bits tem os seguintes pesos:

-128	64	32	16	8	4	2	1

O *maior* valor obter-se-á quando o primeiro bit for '0' (para não haver contribuição de -128) e todos os outros forem '1' (para haver contribuição de todos os pesos positivos): 0111 1111, que é a codificação em c2 de $64+32+16+8+4+2+1=+127_{(10)}$

O *menor* valor obter-se-á quando o primeiro bit for '1' (para haver contribuição de -128) e todos os outros forem '0' (para não haver contribuição dos pesos positivos): 1000 0000, que é a codificação em c2 de $-128_{(10)}$.

R2: $57_{(10)}=71_{(8)}=111\ 001_{(2)}\rightarrow +57(10)\equiv 0011\ 1001_{(c2)}$
$-57_{(10)}= \equiv 1100\ 0111_{(c2)}$
$70_{(10)}= 46_{(16)}=100\ 0110_{(2)}\rightarrow +70(10)\equiv 0100\ 0110_{(c2)}$
$-70_{(10)}= \equiv 10111010_{(c2)}$

2. [10E2.1] Sejam x = 23 e y = –35.
   2. 1. Converta x e y para binário em notação de complemento para 2.
   2. 2. Utilizando os valores obtidos em 1) execute a operação x+y em binário. Justifique se o resultado obtido é válido.
   2. 3. Qual é o número mínimo de bits necessário para representar em notação de complemento para 2 o resultado da operação x+y?

R1: Posto que $23_{(10)} = 17_{(16)} \Rightarrow 23_{(10)} = 1\ 0111_{(2)}$, vem x = **01 0111**$_{(c2)}$
Posto que $35_{(10)} = 23_{(16)} \Rightarrow 35_{(10)} = 10\ 0011_{(2)}$, vem y = **101 1101**$_{(c2)}$

Repare-se: a representação, em complemento para 2 (abreviadamente: **c2**), dum *inteiro* envolve as *seguintes etapas*:

a) Representar o *módulo* desse inteiro em *binário*;

b) Se o inteiro for *negativo* e a representação binária for '1' seguido de '0's, *nada mais há a fazer* (*mas vidé abaixo!*);
caso contrário, *prefixar com um '0'*: obtém-se a representação do inteiro positivo cujo módulo é o valor dado;

c) Apenas para o caso do inteiro dado ser *negativo*, *obter o simétrico de b)*.

1ª etapa: **x** e **y** representam-se, em binário, por $1\ 0111_{(2)}$ e $10\ 0011_{(2)}$. (*vidé* resolução em Numeric_r.pdf)

2ª etapa: P*refixar com um '0'*, é simples, mas é *quase sempre obrigatório*! É com ela que se obtém +23 e +35:

$+23 = \mathbf{0}1\ 0111_{(c2)}$ e $+35 = \mathbf{0}10\ 0011_{(c2)}$.

Uma nota importante:

- Lembre-se o leitor da instrução primária: quando se trata de representar inteiros *positivos*, é irrelevante o número de '0's *à esquerda*; algo *parecido* vale na representação em c2: +23 e +35 também se podem representar (em c2) – *embora em geral não seja recomendável* - por, por ex.: $+23_{(10)} = 000\ 0001\ 0111_{(c2)}$ e $+35_{(10)} = 0\ 0010\ 0011_{(c2)}$.

- Mas existe uma diferença substancial: quando se trata de representar um inteiro *positivo* em c2, *é mesmo necessário o '0' à esquerda!* É um *erro* afirmar que $10\ 111_{(c2)}$ é a representação em c2 de 23: é, sim, a representação em c2 de –9!

Ressalva: o "*quase sempre obrigatório*" da *2ª etapa* remete para o referido sobre inteiros *negativos* cuja representação em binário é '1' seguido de '0's. Seja, como exemplo a clarificar, a representação de –32. A divisão 32 : 16 volve-se em:

$$\begin{array}{r|l} 32 & 16 \\ \hline 0 & 2 \end{array}$$

Conclui-se: $32_{(10)} = 20_{(16)} = 10\ 0000_{(2)}$ – isto é, '1' seguido de '0's. Então, manda o dito acima, que nestes casos (e só nestes casos) *nada mais há a fazer*: a representação (*mais curta*) em c2 de –32 é **10 0000**. Já agora, a representação em c2 de +32 é **010 0000**: quando toca a representar em c2, –32 basta-se em 6 bits, mas +32 exige (pelo menos) 7 bits!

3ª etapa: A representação em c2 de x já é sabida: x = 01 $0111_{(c2)}$: já só falta a de y – que é um valor *negativo*! Para a lograr, oferecem-se *três* métodos, *todos eles partindo do resultado obtido na 2ª etapa*: +35 = 010 0011

1-3-a) o método *mais rápido*: por escrita imediata. A partir do bit mais à direita, até encontrar o primeiro '1' *inclusive*, varre-se a representação de +35: os bits de –35 são exactamente os mesmos de +35. Invertem-se os restantes bits. *Mentalmente*,

+35	0	1	0	0	0	1	1
Cópia de bits à direita do primeiro '1', *inclusivé*							1
Inversão dos bits à esquerda desse primeiro '1'	1	0	1	1	1	0	
–35	**1**	**0**	**1**	**1**	**1**	**0**	**1**

1-3-b) o método não tão rápido: por subtracção de +35 para '0'. Subentende-se: 0–35 = –35, e a representação de '0' em c2 é, precisamente, 0… Para o efeito, escreve-se a representação de +35, e, por cima, 0, e subtrai-se:

	7º	6º	5º	4º	3º	2º	1º
0	0	0	0	0	0	0	0
(+35)	0	1	0	0	0	1	1
–35	**1**	**0**	**1**	**1**	**1**	**0**	**1**

Soma binária:

+	**0**	**1**
0	0	1
1	1	10

Começando pelo *bit mais à direita*, e tendo em conta a tabela da adição binária, ao lado recordada:
- 1ºbit: 1 para 0 (ou, com mais rigor, 1 para 10) dá 1 e *vai 1*;
- 2ºbit: 1 mais 1 dá 10, para 10 dá 0 e *vai 1*;
- 3ºbit: 1 mais 0 dá 1, para 10 dá 1 e *vai 1*;
- 4ºbit: 1 mais 0 dá 1, para 10 dá 1 e *vai 1*;
- 5ºbit: 1 mais 0 dá 1, para 10 dá 1 e *vai 1*;
- 6ºbit: 1 mais 1 dá 10, para 10 dá 0 e vai 1;
- 7ºbit: 1 mais 0 dá 1, para 10 dá 1 e *vai 1*

A subtracção finda quando se tiver contemplado o bit mais à esquerda da representação de +35.

1-3-c) o método *mais moroso*: por subtracção de +35 para '1111', seguida de incremento de '1'. Para o efeito, escreve-se a representação de +35, e, por cima, '1111…1', e subtrai-se – ou, o que é o mesmo, *invertem-se todos os bits* de +35 –, após o que se soma '1':

	1	1	1	1	1	1	1
(+ 35)	0	1	0	0	0	1	1
Inversão dos bits	1	0	1	1	1	0	0
Incremento de 1							1
–35	**1**	**0**	**1**	**1**	**1**	**0**	**1**

Como seria de esperar, as representações logradas pelos vários métodos acima são sempre as mesmas, a saber:

$$-35_{(10)} = 101\ 1101_{(c2)}.$$

Confirmação: antes de prosseguir, convém duvidar: estarão certos os resultados obtidos – ou ter-se-á cometido algum erro? Para o discernir, construa-se *mentalmente* a tabela, cujas colunas estão etiquetadas pelas sucessivas *potências de 2*; ela é similar a uma já apresentada acima, com a diferença que o *peso da coluna mais significativa é agora negativo*!

Pesos:	**–64**	**32**	**16**	**8**	**4**	**2**	**1**
$-35_{(c2)}$	1	0	1	1	1	0	1

101 1101$_{(c2)}$ foi alinhado à direita. Somando os *pesos* das colunas em que surgem '1's:
101 $1101_{(c2)} \Rightarrow$ **–64 + 16 + 8 + 4 + 1** = –35 (que é o *valor* de **y**)

Repare-se: a partir do *resultado* obtido (101 $1101_{(c2)}$), regressou-se ao valor *dado* (**–**35); isso predispõe a confiar que ele estará correcto (Pode o leitor aplicar exactamente o mesmo "teste" à representação de x=01 $0111_{(c2)}$ – para confiar que também este estará correcto; em particular, se tivera chegado *erroneamente* a x=1 $0111_{(c2)}$, o teste daria –16 + 4 + 2 + 1 = –9, e, não, **x** – apontando ao leitor que esse valor estaria errado…)

Uma nota importante: Já se ressalvou que, na representação em c2 de inteiros *positivos*, é irrelevante o número de '**0**'s *à esquerda* – desde que haja ao menos um! Similarmente, na representação em c2 de inteiros *negativos*, podem *prefixar-se* '**1**'s *à esquerda*! Concretamente, –35 também se pode representar (em c2) por, por ex.: –35 = 1 1101 $1101_{(c2)}$. O *"nada mais há a fazer"* dito quando, a propósito da conversão para c2 de um inteiro negativo, o cálculo da representação binária do seu módulo se volve em '1' seguido de '0's, tem então uma ressalva: se se exigir um *tamanho mínimo* para a representação em c2, há que prefixar com '1's. Eis exemplos dessa ressalva, quando se exige um tamanho mínimo de, seja, 6 bits:

Decimal	Binário		Decimal	c2
2	10		–2	**1111**10
4	100		–4	**111**100
16	10000		–16	**1**10000

R2a: Somando x = **01 0111**$_{(c2)}$ com y = **101 1101**$_{(c2)}$, vem: x+y=**111 0100**$_{(c2)}$

		7º	6º	5º	4º	3º	2º	1º
Transporte	0	0	1	1	1	1	1	
x			0	1	0	1	1	1
+y		1	0	1	1	1	0	1
x+y		1	1	1	0	1		0

Repare-se que, na tabela, os números **01 0111** e **101 1101** foram alinhados à direita. Começando pelo *bit mais à direita*, e tendo em conta a tabela da soma binária, acima recordada:

- 1ºbit: 1 + 1 dá **0** e *vai 1*;
- 2ºbit: 1 + 1 + 0 dá **0** e *vai 1*;
- 3ºbit: 1 + 1 + 1 dá **1** e *vai 1*;
- 4ºbit: 1 + 0 + 1 dá **0** e *vai 1*;
- 5ºbit: 1 + 1 + 1 dá **1** e *vai 1*;
- 6ºbit: 1 + 0 + 0 dá **1** e não vai nada (com mais rigor: e *vai 0*);
- 7ºbit: 0 + 1 dá **1** e *vai* 0 (que de qualquer maneira é ignorado)

A subtracção finda quando se tiver contemplado o último bit do número "com representação mais comprida".

Nota importante: Suponha-se que, em vez de x=23 e y=–35, se tinha x=–23 e y=+35. São pacíficas as representações em c2: –23 = **10 1001**$_{(c2)}$ e 35 = **010 0011**$_{(c2)}$. Somando, tal como acima se fez:

	7º	6º	5º	4º	3º	2º	1º
–23		1	0	1	0	0	1
+35	0	1	0	0	0	1	1
–23+35	1	0	0	1	1	0	0

Acontece que o resultado, **100 1100**, começa por '1': representa, pois, um número *negativo* – quando seria expectável lograr-se um número *positivo*, já que –23 + 35 = +12! *Onde estará o erro?* Não custa dar com ele: aquando do tratamento do 7º bit, ter-se-á imaginado algo do género "1 + **0** + 0 dá 1 (e não vai nada)", isto é, ter-se-á *subentendido que o 7º bit de –23 - que está "em branco" - seria **0***… Mas se fora assim, a primeira linha, **0**10 1001, representaria – não –23, mas 32+8+1=+41: estar-se-ia somando +41+35, que resulta +76 – cuja representação *binária* é 100 1100 – precisamente, o resultado logrado! O artifício para evitar este erro é: *não deixar posições* em branco *"à esquerda" em números negativos*, preenchê-las '**1**'. E viria:

–23	**1**	1	0	1	0	0	1
+35	0	1	0	0	0	1	1
–23+35	0	0	0	1	1	0	0

(em que se ignora o *último* "e vai 1"). O resultado, **000 1100**, é *correcto*: representa, em c2, +12, o resultado expectável.

R2b: O resultado da soma x+y, que é **111 0100**, está *correcto*. Justificação:

Eis *três* métodos de verificar se o resultado está correcto:

2b-a) o método *mais moroso* – mas que serve de **Confirmação** – ele mesmo desdobrado em duas *alternativas*:
Sabido que, somando na base 10, se obtém: x+y = 23 + (–35) = **$-12_{(10)}$**,
- ou se parte do resultado obtido, **111 0100**, e se averigua se se chega a $-12_{(10)}$…
- ou se parte de $-12_{(10)}$, e se confere se se chega ao resultado obtido, **111 0100**…

2b-a-1) A conversão de $111\ 0100_{(c2)}$ para a base 10 é trivial: é uma soma de potências, em que *a de maior peso é negativa*. No caso presente, vem $-2^6 + 2^5 + 2^4 + 2^2 = -64 + 32 + 16 + 4 =$ **–12**;

2b-a-2) Posto que $12_{(10)} = 1100_{(2)}$, vem +12=0 $1100_{(c2)}$ e –12 = **1 0100**$_{(c2)}$. Aparentemente, este número, **1 0100**$_{(c2)}$, *não é a mesma coisa* que o resultado obtido, x+y = **111 0100**$_{(c2)}$. Mas só aparentemente: já se ressalvou que, na representação em c2 de inteiros *negativos*, se podem *prefixar* '1's *à esquerda* dessa representação – e o apôr de dois '1's à esquerda de **1 0100** volve-se precisamente em **11 10100**!

2b-b) o método *mais rápido*: reflectir… Eis que **x** e **y** são dois números de *sinal contrário*: o primeiro é *positivo* e o segundo é *negativo*. Pelo que a soma das suas representações em c2 é *sempre correcta*.

2b-c) No enunciado em causa, x e y têm *sinal contrário* – de que se pode concluir *imediatamente* que a sua soma em c2 é *sempre correcta*. Mas é altura de averiguar: e se tiverem o *mesmo sinal*? É claro que se pode sempre recorrer ao método **2-b)**, mas ele é moroso: a *lei-do-menor-esforço* manda que se abra um parêntesis para chegar a um algoritmo que permite discernir *instantaneamente* acerca da *correcção* do resultado…

2b-c-1) Suponha-se que se pretende somar –23 = **10 1001**$_{(c2)}$ e –35 = **101 1101**$_{(c2)}$. Eis a soma, procedendo como acima, e onde, para suportar conclusões mais adiante, se explicitou o transporte (o "*e vai …*"):

Transporte	**1**	**1**	1	1	0	0	1	
–23		1	1	0	1	0	0	1
–35		1	0	1	1	1	0	1
–23+(–35)		**1**	0	0	0	1	1	0

Estará *correcto* o resultado, **100 0110**$_{(c2)}$, a que se chegou? A soma de –23 com –35 resulta –58, que é um número *negativo* – o que significa que, em c2, a sua representação deve começar por '**1**'; de facto, **100 0110** começa por '**1**' – e, o leitor pode conferi-lo (*Confirmação*: $-2^6 + 2^2 + 2^1 = -58$), é a representação em c2 de –58: o resultado está *correcto*!

2b-c-2) Suponha-se agora que se pretende somar –35 e –35. Eis a soma, procedendo como acima:

Transporte	**1**	**0**	1	1	1	0	1	
–35		1	0	1	1	1	0	1
–35		1	0	1	1	1	0	1
–35+(–35)		**0**	1	1	1	0	1	0

Estará *correcto* o resultado, **011 1010**$_{(c2)}$, a que se chegou? É claro que *não*: a soma de –35 com –35 resulta um valor *negativo* – pelo que a sua representação em c2 deve começar por '**1**'; mas **011 1010** começa por '**0**'!

2b-c-3) Suponha-se porém que se pretende somar +23 e +23. Eis a soma, procedendo como acima (*e representando +23 pelo menor número de bits necessário para o representar em c2*):

Transporte	**0**	**1**	0	1	1	1	
+23		0	1	0	1	1	1
+23		0	1	0	1	1	1
+23+(+23)		1	0	1	1	1	0

Estará *correcto* o resultado, **10 1110**$_{(c2)}$, a que se chegou? É claro que *não*: a soma de +23 com +23 resulta um número *positivo* – pelo que a sua representação em c2 deve começar por '**0**'; mas **10 1110** começa por '**1**'!

2b-c-4) Suponha-se enfim que se pretende somar –23 e –9 = **11 0111**$_{(c2)}$. Eis a soma, procedendo como acima:

Transporte	**1**	**1**	1	1	1	1	
–23		1	0	1	0	0	1
–9		1	1	0	1	1	1
–23+(–9)		1	0	0	0	0	0

Estará *correcto* o resultado, **10 0000**$_{(c2)}$, a que se chegou? É claro que *sim*: a soma de –23 com –9 resulta –32 – cuja representação em c2 (*vidé* 2ª etapa acima) é exactamente **10 0000**$_{(c2)}$!

Revejam-se estes quatro "suponha-se" acima, trata-se de somar números com o *mesmo sinal*: –23 e –35, –35 e –35, +23 e +23 e –23 e –9. Na primeira e última somas, o resultado logrado é *correcto*, nos demais é *incorrecto*. Que se estará a passar?

- Repare-se em **2b-c-3)**: para representar +23 em c2, bastam 6 bits: **01 0111**$_{(c2)}$. Sucede que o *maior* inteiro *positivo* que se pode representar em c2 com 6 bits é 01 1111$_{(c2)}$ - que é a representação de +31. Ora, +23+23=46 – *que excede 31*: no jargão aplicável, diz-se que ocorreu *overflow*: a soma excede o máximo que pode ser representado com 6 bits, ou, por outras palavras: *a representação em c2 da soma precisa de mais do que 6 bits*! (Abra-se um parêntesis: se +23 houvera sido representado com 7 bits (0010111), a soma +23+23 (que daria 0101110) estaria *correcta* – pois, com 7 bits, o máximo que se pode representar vem a ser +63, que não é inferior a 46);

- Repare-se agora em **2b-c-2)**: para representar –35 em c2, bastam 7 bits: **101 1101**$_{(c2)}$. Sucede que o *menor* inteiro *negativo* que se pode representar com 7 bits é 100 0000 - que é a representação de –64. Ora acontece que –35+(–35)=–70 – *cujo módulo excede 64*: no jargão aplicável, diz-se que ocorreu *overflow/underflow*: a representação em c2 da soma precisa de mais do que 7 bits!

- Nos restantes casos, não ocorreu *overflow*: em **2b-c-1)**, –23+(–35)= –58, que não excede o *maior inteiro* que pode ser representado em c2 com 7 bits, a saber: 011 1111$_{(c2)}$ = +63; e, em **2b-c-4)**, –23+(–9)= –32, que não é inferior ao *menor inteiro negativo* que pode ser representado em c2 com 6 bits: 10 0000$_{(c2)}$ = –32.

Resumindo: os resultados obtidos quando se soma em c2 não estão correctos quando ocorre *overflow*, mas estão-no quando *não* ocorre *overflow* – e o que se pretende é um algoritmo simples que sinalize quando ocorre *overflow*…

Para o entender, reveja-se, nas somas acima, o transporte (o "*e vai…*") *gerado pelos dois últimos bits de cada soma*:

Transporte:	Posição à *esquerda* do número	Bit mais à *esquerda*	*Overflow*?
2b-c-1: –23+(–35)	1	1	Não
2b-c-2: –35+(–35)	1	0	⇐ Sim
2b-c-3: +23+(+23)	0	1	⇐ Sim
2b-c-4: –23+(–9)	1	1	Não
2a: +23+(–35)	0	0	Não

O leitor pode conferir que *não* há *overflow* exactamente nas linhas em que os *dois últimos* Transportes são *iguais*: ou ambos '**0**' ou ambos '**1**'; nos restantes casos, há *overflow*. É este o algoritmo que se buscava.

Para o caso presente, na soma x+y, apresentada acima, os dois últimos Transportes são, ambos, '**0**' – do que se conclui que não houve *overflow*, o resultado está *correcto*.

R3: 5

Eis *três* métodos para determinar o número mínimo de bits necessário para representar em c2 o resultado de x+y:

3a) o método *mais rápido*: Eliminar os *bits repetidos à esquerda* do resultado (excepto um), contar quantos bits ficam

.

Partindo de x+y = **111 0100**$_{(c2)}$, que começa com três '1', deixar ficar *um único* deles, e contar quantos ficam: 5

3b) o método *mais moroso*: obter o resultado na base 10, e achar a sua mais curta representação em c2:

x+y = 23 + (–35) = –12; 12 = 1100$_{(2)}$ ⇒ +12 = 0 1100$_{(c2)}$ ⇒ –12 = 1 0100$_{(c2)}$ – que se estende por 5 bits

3b) o método *não tão moroso*: obter o resultado na base 10, e reflectir (sem chegar a deduzir a sua representação binária)

x+y = 23 + (–35) = –12

O módulo do resultado é 12.

Não sendo 12 uma *potência de 2*, as potências de 2 entre as quais se encontra são 8 e 16: $2^3 < 12 < 2^4$. Isso significa que, para representar 12 em base 2, três bits são insuficientes: são necessários 4 bits. Para a representação em c2, é também necessário contabilizar também o bit de *sinal*: chega-se a 4 + 1 = 5 bits

Nota: repare-se que, se o número em causa fosse uma *potência de 2*, haveria dois casos a considerar:

- se ele fora *positivo*, como seria o caso de, por ex., +64, que é $2^6 = 100\ 0000_{(2)}$, seria preciso um total de 7 + 1 bits;
- mas, se ele fora *negativo*, como seria o caso de, por ex., –64, e posto que $64 = 2^6 = 100\ 0000_{(2)}$, então bastariam 7 bits.

3. [10E1.4] Dados A = 011011 e B = 001110 representados em complemento para 2 de 6 bits, execute em binário as operações A+B e A–B, indique qual o resultado das operações em binário e em decimal, e indique se os resultados das operações efectuadas são ou não válidos.

R1:

A:	0	1	1	0	1	1
B:	0	0	1	1	1	0
A+B=	**1**	**0**	**1**	**0**	**0**	**1**
A–B=	**0**	**0**	**1**	**1**	**0**	**1**

Soma dos pesos	Representação decimal
–32+8+1=	$-23_{(10)}$
+8+4+1=	$13_{(10)}$

$A+B=101001_{(c2)}$, não é válido pois representa (em c2) um valor *negativo*, e os valores de partida são *positivos*;
$A–B=001101_{(c2)}$, é válido pois é resultado da *subtracção de dois valores com mesmo sinal* (ambos positivos).

Confirmação: os transportes gerados nos dois últimos bits são: {0,1} e {0,0} (respectivamente na soma e subtracção) – e somente quando são iguais (o que só sucede em A–B) é que o resultado (em c2) vem a ser válido.

Reparo: para converter, em *decimal*, os resultados das operações em complemento-para-2, o mais imediato será somar os *pesos* dos '1's: $A+B \rightarrow -32+8+1=-23_{(10)}$ e $A–B \rightarrow 8+4+1=13_{(10)}$ (Recorde-se que os pesos em c2 são {–32,16,8,4,2,1})

4. [10E4.3] Considere A=–41 e B=55. Realize a operação –A–B utilizando a representação em notação de complemento para 2 de 7 bits. Indique se o resultado é válido e qual o número mínimo de bits necessário para realizar a operação correctamente.

R: Tomando um atalho evidente, –A–B volve-se em –(–41)–55=41+(–55): basta somar as representações em c2 de 41 e de –55 – pelo que a primeira etapa será determiná-las:

Eis um método rápido para as determinar (*vidé* [10E2.1]):

a) converter para a base 16, por divisões sucessivas do módulo por 16;

41	16		55	16
9	2		7	3

de que se conclui: $41_{(10)}=29_{(16)}$ e $55_{(10)}=37_{(16)}$ (*Confirmação*: 2*16+9=41 e 3*16+7=48+7=55)

b) converter em binário, pela substituição dos hexadecimais pelos tetra-bits correspondentes: $29_{(16)}= 101001_{(2)}$ e $37_{(16)}= 110111_{(2)}$ (*Confirmação*: 32+8+1=41 e 32+16+4+2+1=55)

c) adicionar um '0' à esquerda e, no caso de –55, fazer algo mais:

$101001_{(2)} \rightarrow 0101001_{(c2)}$ e $110111_{(2)} \rightarrow 0110111 \rightarrow 1001001_{(c2)}$ (*Confirmação*: 32+8+1=41 e –64+8+1=–55)

A soma requerida volve-se então em:

+41	0	1	0	1	0	0	1
–55	1	0	0	1	0	0	1
–14	1	1	1	0	0	1	0

O resultado é então $1110010_{(c2)}$ (*Confirmação*: –64+32+16+2=–14)

O resultado é *válido*: volve-se na representação em c2 de –14, que é o resultado de 41+(–55).

Para o representar, repare-se: podem remover-se os '**1**'s à *esquerda*, até restar apenas um único, isto é: $1110010_{(c2)}$, que ocupa **7**-bit, significa o mesmo que $10010_{(c2)}$ numa representação com apenas **5**-bit (Confirmação: –16+2=–14). '**5**' é o número mínimo de bits necessário para realizar correctamente a operação em causa.

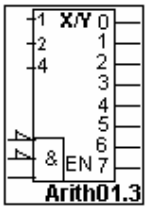
Nota: Repare o leitor na praxis de *confirmar* cada etapa: ela não é supérflua, convém mesmo fazê-lo - nem que seja para evitar que as respostas às interrogações finais *"o resultado é válido? e qual o número mínimo...?"* fiquem de antemão ameaçadas de insucesso. Dada a facilidade e rapidez em cada confirmação, não há a menor desculpa para que as respostas a tais interrogações estejam erradas porque se basearam no resultado erróneo de uma *simplérrima* adição…

Ainda que aparente ser puro desperdício de tempo, *Confirmar um resultado* cai por conseguinte sob o signo da **lei do menor esforço**: fazer (**bem**) a disciplina em, apenas, **um** semestre...

Circuitos Aritméticos

{Arith_r.doc}

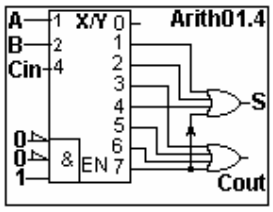
1. [10T1.7] A partir do descodificador da figura Arith01.3, e utilizando o mínimo de portas lógicas adicionais necessárias, desenhe o logigrama de um circuito que realize um somador completo de 1 bit.



R: A tabela de verdade de um *somador completo* de 1 bit {A+B} vem a ser a seguinte:

Resultado (**S**)					Transporte (**Cout**)				
	0	1	0	1	0	1	0	1	**A**
0	0	1	1	**0**	0	0	0	**1**	
1	1	0	0	1	0	1	1	1	
Cin					**B**				

Por exemplo, quando {A=**1**, B=**1**, Cin=**0**}, em que Cin é o Transporte *precedente*, o **Resultado** é S=**0**, e o **Transporte** é Cout=**1** (Se não está a ver: a soma, na base 10, de **1**+**1**+**0** volve-se em $2_{(10)}$; na base 2, isso escreve-se $10_{(2)}$; o '**1**' é o Transporte, o '**0**' é o Resultado).



Em Arith01.4, apresenta-se o logigrama do somador completo…

Processo mental: pressupôs-se *lógica positiva*, isto é, a correspondência: {'**0**'→**Low**, '**1**'→**High**}. As entradas do circuito {C,B,A} são aplicadas às entradas de selecção do *decoder*.

- Pela tabela acima, o transporte **Cout** é '**1**' quando, e só quando, {Cin**=0**,B=**1**,A=**1**} **ou** {Cin**=1**,B=**0**,A=**1**} **ou** {Cin**=1**,B=**1**,A=**0**} **ou** {Cin**=1**,B=**1**,A=**1**}; repare-se que **011, 101**, **110** e **111** são as representações binárias de **3**, **5**, **6**, e **7** (respectivamente). Acontece que, aquando da combinação **011**, a saída '**3**' (e só ela) fica *activa* – concretamente, fica '**1**'; nos restantes casos, ela fica '**0**'; idem para as combinações **101**, **110** e **111**. Então, o transporte, **Cout**, obtém-se por OR das saídas '**3**', '**5**', '**6**' e '**7**': **Cout** ficará '**1**' quando, e só quando, ao menos uma delas for '**1**'…

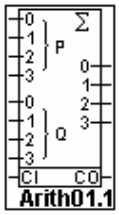
- O resultado **S** é '**1**' quando, e só quando, {Cin**=0**,B=**0**,A=**1**} **ou** {Cin**=0**,B=**1**,A=**0**} **ou** {Cin**=1**,B=**0**,A=**0**} **ou** {Cin**=1**,B=**1**,A=**1**}: quando está a '**1**' ao menos *umas das saídas* '**1**' **ou** '**2**' **ou** '**4**' **ou** '**7**'… Então, **S** será um OR dessas saídas;

- Para que o descodificador *funcione*, é necessário activar o *Enable*; isso acontece forçando '**0**' (*Low*) nas suas duas entradas "de cima", e '**1**' (*High*) na "de baixo".

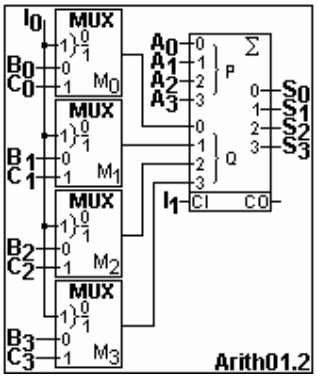
2. [10T1.6] Considere um somador idêntico ao indicado na figura Arith01.1.

2. 1. A partir desse somador e de um máximo de 4 multiplexers com uma variável de selecção, concretize um circuito que, dada uma instrução de 2 bits $<I_1,I_0>$ e 3 números inteiros com sinal representados em complemento para 2 de 4 bits $<A_3A_2A_1A_0,B_3B_2B_1B_0,C_3C_2C_1C_0>$, realize uma das 4 funções seguintes:

I_1 I_0	Função
00	S = A+B
01	S = A+C
10	S = A+B+1
11	S = A+C+1



2. 2. Dados I_1=1, I_0=0, e ainda os números decimais A = 4, B = -8, C = 7, indique quais os valores em cada uma das saídas do somador assumindo que os números presentes nas entradas utilizam a representação de complemento para 2. O resultado da operação realizada é válido?



R1: S pode considerar-se uma *soma de três operandos*:

– o primeiro é, *sempre*, A;

– o segundo é uma *escolha* – seleccionada por $\mathbf{I_0}$ - entre B e C (repare-se: quando $\mathbf{I_0}$=0, S envolve a soma A+B, e quando $\mathbf{I_0}$=1, envolve a soma A+C);

– o terceiro é uma *escolha* – seleccionada por $\mathbf{I_1}$ - entre 0 e 1 (quando e apenas quando, $\mathbf{I_1}$=1, há que somar **1** ao resultado de A+B ou A+C).

Isso conduz ao logigrama em Arith01.2:

- a **P** aplica-se **A**, **ordenado A_0→'0', …, A_3→'3'** (i.e.: 'bit de *menor* peso de A' → 'entrada P de *menor* peso');
- a **Q** aplica-se a saída dos multiplexers, **ordenados M_0→'0', M_3→'3'** (i.e.: 'saída do multiplexer de *menor* peso' → 'entrada Q de *menor* peso'); a entrada de *selecção* dos multiplexers é $\mathbf{I_0}$ – e nas entradas de *dados* aplica-se B e C, B sendo aplicado às de menor peso: quando $\mathbf{I_0}$=0, S envolve a soma A+B…
- a **CI** (*carry-in*) aplica-se a própria variável $\mathbf{I_1}$: quando, e apenas quando, $\mathbf{I_1}$=1, há que somar 1 para perfazer S…

R2: Para {I_1=1, I_0=0}, S volve-se em A+B+1, que no caso vem a ser:

A (+4)	0	1	0	0	Representação em complemento para 2 de +4
B (-8)	1	0	0	0	Representação em complemento para 2 de -8
+1	0	0	0	1	
S=A+B+1	1	1	0	1	Representação em complemento para 2 de -3

Nas saídas do somador observa-se então: S_3=1, S_2=1, S_1=0, S_0=1.

Este resultado é válido: é a representação em complemento para 2 de -3, que é o valor correcto de 4-8+1.

3. [10E3.3,10T3.6] São dados A = 001111 e B = 011011 representados em complemento para 2 de 6 bits
   3. 1. Execute em binário as operações A+B e A−B, indique qual o resultado das operações em binário e indique se os resultados das operações efectuadas são ou não válidos;
   3. 2. A partir de 2 somadores de 4 bits, desenhe o logigrama de um circuito que permita somar quaisquer 2 números representados em complemento para 2 de 6 bits. Desenhe no logigrama quais os valores presentes nas entradas e saídas do circuito quando nas entradas são aplicados os valores de A e –B indicados na alínea anterior.

R1:

A(+15):	0	0	1	1	1	1
B(+27):	0	1	1	0	1	1
A+B=	**1**	**0**	**1**	**0**	**1**	**0**
−B(−27)	**1**	**0**	**0**	**1**	**0**	**1**
A−B=A+(-B)=	**1**	**1**	**0**	**1**	**0**	**0**

Soma dos pesos	Representação decimal
−32+8+2=	$-22_{(10)}$
-32+4+1=	$-27_{(10)}$
−32+16+4=	$-12_{(10)}$

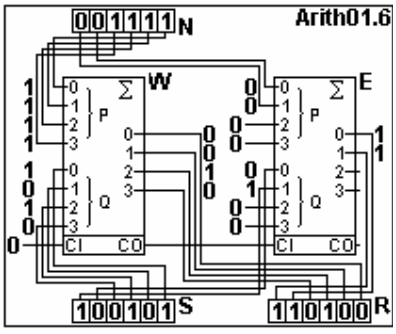
$A+B=101010_{(c2)}$, não é válido pois representa (em c2) um valor *negativo*, e os valores de partida são *positivos*;

$A-B=110100_{(c2)}$, é válido pois é resultado da *adição de dois valores de sinal contrário*.

Confirmação: os transportes gerados nos dois últimos bits são: {0,1} e {0,0} (respectivamente na soma e subtracção) – e somente quando são iguais (o que só sucede em A−B) é que o resultado (em c2) vem a ser válido.

Reparo: para converter, em *decimal*, os resultados das operações em c2, o mais imediato será somar os *pesos* dos '1's, os pesos em c2 de 6-bit sendo {−32,16,8,4,2,1})

R2: O logigrama encontra-se em Arith01.6. Pressuposto que se pretende a soma de dois números em c2 de 6-bit, contidos nos registos **N** e **S**, envolve dois somadores, **W** e **N**: **W** efectua a soma dos 4-bit menos significativos de **N** e **S**, **E** efectua a soma dos 2-bit mais significativos (de **N** e **S**) *com o transporte CO produzido por* **W**. O resultado final, em **R**, é um agregado dos 2-bit menos significativos produzidos por **E** com o resultado de 4-bit produzido por **W**.



Para o caso de $N=A=001111_{(c2)}$ e $S=-B=100101_{(c2)}$, **W** ficará somando 1111 com 0101, que se volve em 10100 (em termos da base 10, trata-se de somar $15_{(10)}$ com $5_{(10)}$, que resulta em $20_{(10)}$, i.e., $14_{(16)}$: **W** produz a soma 0100, **e** o transporte C0=**1**. **E** ficará então somando 0000 com 0010 - mais o transporte (**1**) proveniente de **W**; isso volve-se em 0011: o Resultado final de tudo isso será então $R=110100_{(c2)}$.

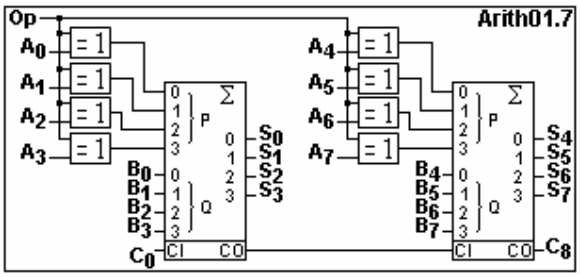
Cuidados: *não deve "deixar-se no ar" nenhuma entrada* (dos somadores) – daí a fixação da entrada CI de **W** (em '**0**') e das entradas '**2**' e '**3**' de P e Q de **E** (em '**0**').

Solução exótica: A mente humana é por vezes muito fértil… e pode chegar a *pseudo-soluções* - que funcionam mesmo, mas que *devem ser descartadas*… Por ex., e em vez de repartir os valores **N** e **S** em {2 + 4} bits, poder-se-ia reparti-los em {4 + 2} bits (ou, quem sabe, em {3+3}): os 2-bits menos significativos seriam aplicados a **W**, e os 4-bit mais significativos seriam aplicados a **E**, *juntamente com o 3º bit do resultado em W*. Para o ex. entre mãos, **W** ficaria somando 00**11** com 00**01**, que resulta em 0**100**; os 2-bit menos significativos do resultado final seriam então **00** – e aproveitar-se-ia o 3º bit, '**1**', como *transporte* a injectar na entrada CI de **E**: **E** ficaria somando os 4-bit mais significativos de **N** e **S** (0011 e 1001) - com **1** -, resultando 1101… Engenhoso, sem dúvida, mas *pode* significar algumas nuvens debaixo do chapéu de quem o subscreve…

4. [11P5.2] Considere o circuito da figura Arith01.7. Indique, em cada caso, qual a operação a ser realizada e quais os valores lógicos nas saídas do circuito. Se existir *overflow*: como o detectaria? Considere, em todos os casos, A e B números inteiros de 8-bits com sinal, representados em complemento para 2.

   A=0011 1001, B=0100 0110.

   4. 1. Op=1, C0=1
   4. 2. Op=1, C0=0
   4. 3. Op=0, C0=0
   4. 4. Op=0, C0=1



R: Repare que A≡+57$_{(10)}$, B≡+70$_{(10)}$, *vidé* [11P5.1]

Op	C0	Operação	A+B	A+B+1	-A+B-1	-A+B
0 ⇒ P=A	0	A+B	0011 1001	0011 1001	1100 0110	1100 0110
0 ⇒ P=A	1	A+B+1	0100 0110	0100 0110	0100 0110	0100 0110
1 ⇒ P= $\overline{A}$	0	(-A)+B-1		1		1
1 ⇒ P= $\overline{A}$	1	(-A)+B	0111 1111	1000 0000	0000 1100	0000 1101
			+127	*overflow*	12	13

Para detectar o *overflow*, basta comparar os *carry* gerados no 7º e 8º bit: se forem diferentes, há overflow...

Nomeadamente, na operação A+B+1, os transportes gerados no 7º e 8º bit são, respectivamente, '1' e '0', isto é: são diferentes; então, há overflow... Já na operação –A+B, eles vêm a ser '1' e '1', isto é: iguais; então, não há *overflow*

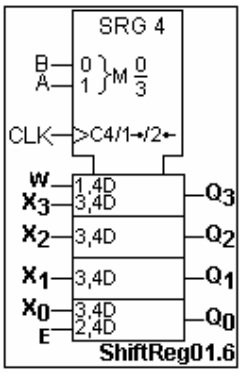
Registos de Deslocamento

{ShiftReg_r.doc}

Preâmbulo:

P1: ***SRG 4***

Como o denota a mnemónica, SRG 4 (*Shift Register*), o circuito figurado vem a ser um Registo de Deslocamento de 4-bit: ao ritmo dos impulsos de relógio, o seu conteúdo desloca-se para a *direita* (**W**→**E**) ou para a *esquerda* (**W**←**E**):



A informação registada é apresentada nas saídas {Q_3, Q_2, Q_1, Q_0}.

Para o sinal de relógio, CLK, é disponibilizada uma entrada – nomeada C**4**; o registo reage ao flanco ascendente do CLK: repare-se no símbolo (⊳) que acompanha a entrada C4. O ***sufixo*** '**4**' significa que *todas as entradas com* ***prefixo '4'*** *estão sincronizadas pelo sinal de relógio*.

Além de *deslocar a informação num dos sentidos*, o registo viabiliza o *seu carregamento com um valor arbitrário* escolhido pelo projectista. Formalmente, diz-se que o registo tem 4 *modos de funcionamento* {M_0, M_1, M_2 e M_2} – tantas quantas as possibilidades distintas de combinação das entradas {A, B}.

O que {M_0, ..., M_2} significam (*deslocamento* – e em que sentido? -, ou *carregamento*, ou *não fazer nada*?) é determinado pelos ***prefixos*** de **D**, '←' e '→':

A	B	M	Funcionamento		Q_3	Q_2	Q_1	Q_0
0	0	**0**	o conteúdo do registo mantém-se inalterado		q_3	q_2	q_1	q_0
0	1	**1**	o conteúdo do registo desloca-se *à direita*	repare-se na notação em C4: '/**1**→'	**W**	q_3	q_2	q_1
1	0	**2**	o conteúdo do registo desloca-se *à esquerda*	repare-se na notação em C4: '/**2**←'	q_2	q_1	q_0	**E**
1	1	**3**	é feito o carregamento *em paralelo* do registo	repare-se na notação **3**,4D em 4 entradas	X_3	X_2	X_1	X_0

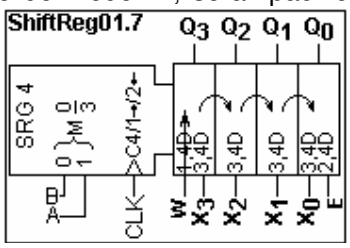
Em ordem a proporcionar o *carregamento com um valor arbitrário* {X_3, X_2, X_1, X_0}, o registo disponibiliza 4 entradas de *dados* '**4**D', todas elas sincronizadas pelo CLK: basta aplicar tal valor a essas entradas, seleccionar o Modo=**3** (nas entradas {A,B}), e provocar um impulso de relógio.

Repare-se que, dessas 4 entradas, apenas a superior se encontra referenciada 3,4D: subentende-se que as que lhe estão por baixo exibem um comportamento idêntico, não há que *repetir* a símbólica…

O registo figurado tem ainda:

- uma entrada ***1,4D***; o ***sufixo*** '**1**' significa que *ela tem sentido no contexto do Modo '**1**';* neste Modo, o registo desloca-se (aquando do impulso de relógio), à *direita (é o que significa '→' na* notação C4: '/**1**→'*)*. Isso significa o seguinte: o valor, **W**, nessa entrada '***1,4D***', vai ocupar a posição Q_3 – e como que isso *empurra para baixo todos os bits adiante* (formalmente: para a posição Q_0, emigra o que *estava* na posição Q_1; para a posição Q_1 emigra o que *estava* na posição Q_2; para a posição Q_2 emigra o que estava na posição Q_3).

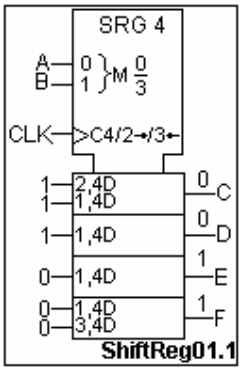
- uma entrada ***2,4D***; o ***sufixo*** '**2**' significa que *ela tem sentido no contexto do Modo '**2**',* neste Modo, o registo desloca-se (aquando do impulso de relógio), à *esquerda (é o que significa '←' na* notação C4: '/**2**←'*)*... Para quem entendeu o lugar da entrada ***1,4D*** no funcionamento do Modo '**1**', será pacífico adivinhar o lugar da entrada ***2,4D*** no funcionamento do Modo '**2**'...



(Parêntesis: pode fazer alguma confusão ao leitor as referências à *direita* e à *esquerda*, quando de facto o que se observa (em ShiftReg01.6) são bits a deslizar à maneira de um elevador – de *cima* para *baixo* ou vice-versa... Se for o caso, experimente mentalmente dar um piparote ao registo, para ele ficar como em ShiftReg01.7: no Modo '**1**', Q_0 evapora-se..., [$Q_3Q_2Q_1$] deslocam-se todos para a *direita* e **W** vai ocupar a posição mais à esquerda {Q_3})

O uso de ***prefixos*** (em **D**, '←' e '→'), esclarece a finalidade de cada Modo. Entretanto, o fabricante de um registo pode decidir que ele se desloca à direita no Modo 3 (e, não, no Modo 1, como acontece em ShiftReg01.6)! Assim, antes de tudo há que determinar quais os ***sufixos***/***prefixos*** usados (no caso: '0', ...,'3') – e interpretá-los!

1. [10T2.2,10E1.6] Considere o circuito em ShiftReg01.1.
   1. 1. De que tipo de circuito se trata? Quais as suas características principais?
   1. 2. Para cada combinação de valores de A e B indicados na tabela, indique qual o valor das saídas do circuito após um impulso de relógio (Nota: considere que para cada linha da tabela se mantêm sempre os valores indicados na figura original).



A	B	C	D	E	F
0	0				
0	1				
1	0				
1	1				

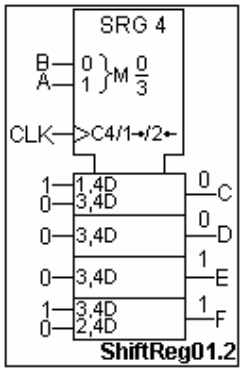
R: Trata-se de um Registo de Deslocamento de 4-bit, como o denota a mnemónica, SRG 4 (*Shift-Register*). As suas saídas são {C, D, E, F}; dispõe de 6 entradas de dados '4D', todas elas sincronizadas pelo CLK – a que está associada a entrada C4. Reage ao flanco ascendente do CLK: repare-se no símbolo (triângulo interno) que o acompanha. Tem 4 modos de funcionamento {M: 0, 1, 2, 3}, seleccionados pelo par {B, A} (No par, A é o selector de menor peso, '1', e B o de maior, '2'):

A	B	M	Funcionamento		C	D	E	F
0	0	0	o conteúdo do registo mantém-se inalterado		0	0	1	1
0	1	2	o conteúdo do registo desloca-se *à direita*	repare-se na notação em C4: '/**2**→'	1	0	0	1
1	0	1	é feito o carregamento *em paralelo* do registo	repare-se na notação **1**,4D em 4 entradas	1	1	0	0
1	1	3	o conteúdo do registo desloca-se *à esquerda*	repare-se na notação em C4: '/**3**←'	0	1	1	0

As ultimas colunas da tabela explicitam as saídas do circuito para as combinações distintas que o par {B,A} pode assumir. Para a sua percepção, será suficiente compreender que, na prática,

- no modo M=**2**, o conteúdo do registo desloca-se de *cima para baixo*, de C para F: para F passa o conteúdo de E, para E passa o conteúdo de D, etc. Na 'posição' deixada vaga (em C), entra o bit de dados mais acima, '**1**': repare-se na notação **2**,4D;
- no modo M=**3**, o conteúdo do registo desloca-se de *baixo para cima*, de F para C: para C passa o conteúdo de D, para D passa o conteúdo de E, etc. Na 'posição' deixada vaga (em F), entra o bit de dados mais abaixo, '**0**', repare-se na notação **3**,4D.

2. [10E2.7] Considere o circuito da figura ShiftRe01.2. Para cada combinação de valores de A e B indicados na tabela, indique qual o valor das saídas do circuito após um impulso de relógio (Nota: ao preencher cada linha da tabela considere que se têm sempre os valores indicados na figura original).



A	B	C	D	E	F
0	0				
0	1				
1	0				
1	1				

R: A tabela abaixo explicita as saídas para as combinações que o par {B,A} pode assumir

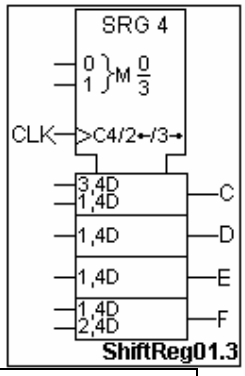
Considerandos: Trata-se de um Registo de Deslocamento de 4-bit, com 4 modos de funcionamento {M: 0, 1, 2, 3}, seleccionados pelo par {A, B} (No par, A é o selector de *maior* peso, '2', e B o de *menor*, '1'):

A	B	M	Funcionamento		C	D	E	F
0	0	0	o conteúdo do registo mantém-se inalterado		0	0	1	1
0	1	1	o conteúdo do registo desloca-se *à direita*	repare-se na notação em C4: '/**1**→'	**1**	0	0	1
1	0	2	o conteúdo do registo desloca-se *à esquerda*	repare-se na notação em C4: '/**2**←'	0	1	1	**0**
1	1	3	é feito o carregamento *em paralelo* do registo	repare-se na notação **3**,4D em 4 entradas	**0**	**0**	**0**	**1**

Na prática,

- no modo M=**1**, o conteúdo do registo desloca-se de C para F, e em C entra o bit de dados mais acima, '**1**';
- no modo M=**2**, o conteúdo do registo desloca-se de F para C; e em F entra o bit de dados mais abaixo, '**0**'

3. [10E4.6] A partir do circuito representado na figura ShiftReg01.3, e utilizando no máximo uma porta lógica adicional, desenhe um circuito que implemente o seguinte ciclo de estados … 9 → 12 → 4 → 9 … Considere que **C** é a saída de maior peso.



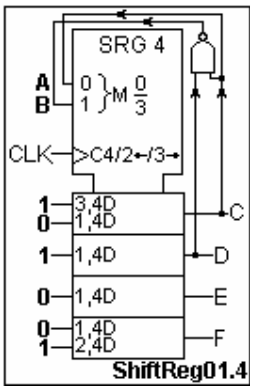
R: O circuito encontra-se em ShiftReg01.4

Processo mental, com a ajuda de um quadro de Karnaugh: ShiftReg01.3 esquematiza um Registo de Deslocamento de 4-bit, cujas saídas são {C, D, E, F}. Tem 4 modos de funcionamento {M: 0, 1, 2, 3}, seleccionados por um par de entradas, designem-se de {B, A} (No par, A é o selector de menor peso, '**0**', e B o de maior, '**1**'):

B	A	M	Funcionamento	
0	0	0	o conteúdo do registo mantém-se inalterado	
0	1	1	é feito o carregamento *em paralelo* do registo	repare-se na notação **1**,4D em 4 entradas
1	0	2	o conteúdo do registo desloca-se *à esquerda*	repare-se na notação em C4: '**/2**←'
1	1	3	o conteúdo do registo desloca-se *à direita*	repare-se na notação em C4: '**/3**→'

Com vista à resolução da questão enunciada, quiçá seja mais cómodo escrever o ciclo como sucessão de linhas em binário:

C	D	E	F
1	**0**	**0**	**1**
1	**1**	**0**	**0**
0	**1**	**0**	**0**
1	0	0	1



A questão transforma-se em: estando memorizado no Registo um dos membros do ciclo, qual o *Modo* que se deve forçar para que, no impulso de relógio seguinte, o Registo passe a conter o membro seguinte do ciclo? Não custa ver que:

- se obtém '1100' de '1001', por deslocamento à *direita* (Modo 3, entrando '**1**' pela esquerda);
- se obtém '0100' de '1100', por carregamento *em paralelo* do registo de **0100** (Modo 1);
- se obtém '1001' de '0100', por deslocamento à *esquerda* (Modo 2, entrando '**1**' pela direita)…

Exprimindo o Modo como função do que está no registo, e na forma de quadro de Karnaugh:

Modo		0	0	1	1
		0	1	1	0
0	0	x	x	x	x
0	1	**10 (2)**	x	x	x
1	1	**01 (1)**	x	x	x
1	0	x	**11 (3)**	x	x
C	**D**				

⇒

0	0	1	1
0	1	1	0
x	x	x	x
1	x	x	x
0	x	x	x
x	**1**	x	x

B

e

0	0	1	1	**E**
0	1	1	0	**F**
x	x	x	x	
0	x	x	x	
1	x	x	x	
x	**1**	x	x	

A

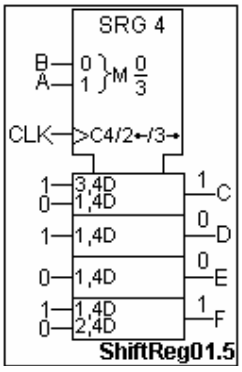
De que se deduz:

$A = C$

$B = \overline{C} + \overline{D} = \overline{C\,D}$ (solução só com um NAND, a que está figurada em ShiftReg01.4);

(Outra solução possível: $B = C\,\overline{D} + \overline{C}\,D = C \oplus D$, que envolve apenas um XOR)

4. [10E3.6] Considere o circuito em ShiftReg01.5. Para cada combinação de valores de A e B indicados na tabela, indique qual o valor das saídas do circuito após um impulso de relógio (Nota: ao preencher cada linha da tabela, considere que se têm sempre os valores indicados na figura original).



A	B	C	D	E	F
0	0				
0	1				
1	0				
1	1				

R: Trata-se de um Registo de Deslocamento de 4-bit, com 4 modos de funcionamento {M: 0, 1, 2, 3}, seleccionados pelo par {A,B} (No par, A é o selector de maior peso, '1', e B o de menor, '0'):

A	B	M	Funcionamento		C	D	E	F
0	0	0	o conteúdo do registo mantém-se inalterado		1	0	0	1
0	1	1	é feito o carregamento *em paralelo* do registo	repare-se na notação **1**,4D em 4 entradas	0	1	0	1
1	0	2	o conteúdo do registo desloca-se *à esquerda*	repare-se na notação em C4: '/**2**←'	0	0	1	0
1	1	3	o conteúdo do registo desloca-se *à direita*	repare-se na notação em C4: '/**3**→'	1	1	0	0

As ultimas colunas da tabela explicitam as saídas do circuito para as combinações distintas que o par {A,B} pode assumir. Na prática,

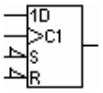
- no modo M=**3**, o conteúdo do registo desloca-se de C para F: para F passa o conteúdo de E, para E passa o conteúdo de D, etc. Na 'posição' deixada vaga (em C), entra o bit de dados mais acima, '**1**': repare-se na notação **3**,4D;
- no modo M=**2**, o conteúdo do registo desloca-se de F para C: para C passa o conteúdo de D, para D passa o conteúdo de E, etc. Na 'posição' deixada vaga (em F), entra o bit de dados mais abaixo, '**0**', repare-se na notação **2**,4D.

Desenho de Diagramas de Estado

{StateDiags_r.doc}

Preâmbulo:

P1: **Flip-Flop D**

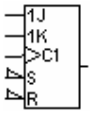


O dispositivo figurado vem a ser um *flip-flop* D. Em ordem a proporcionar o *carregamento com um valor arbitrário*, disponibiliza uma entrada 'D'.

Tabela de excitação do *flip-flop* D (valores a impor à entrada, para lograr uma transição):

Transição que se pretende	Entrada a aplicar
Estado *actual* → Estado *seguinte*	**D**
x → 0	0
x → 1	1

P2: **Flip-Flop JK**



O dispositivo figurado vem a ser um *flip-flop* JK. Em ordem a proporcionar o *carregamento com um valor arbitrário*, disponibiliza um par de entradas 'J,K',

Tabela de excitação do *flip-flop* JK (valores a impor à entrada, para lograr uma transição):

Transição que se pretende	Entradas a aplicar	
Estado *actual* → Estado *seguinte*	**J**	**K**
0 → 0	0	X
0 → 1	1	X
1 → 0	X	1
1 → 1	X	0

Mnemónica: "quando Q_n=0, Q_{n+1}=J; quando Q_n=1, Q_{n+1}=K , ou seja: se o Estado de *partida* é '0', força-se em J o Estado *seguinte*; se for '1', força-se em K o *complemento* do Estado seguinte".

Atributos comuns dos dispositivos figurados:

Para o sinal de relógio, CLK, é disponibilizada uma entrada, C; o *flip-flop* reage ao flanco ascendente do CLK: repare-se no símbolo (▷) que acompanha essa entrada. As entradas ('D' e 'J,K') são sincronizadas pelo CLK (*vidé* ***sufixo*** '**1**' em C**1** e ***prefixo*** '**1**' em **1**D e em **1**J,**1**K)

Os *flip-flops* figurado têm ainda duas entradas *assíncronas*:

- uma entrada ***S*** (Set) – que, quando a *Low*, força a passagem imediata do *flip-flop* ao estado '1';
- uma entrada ***R*** (Reset) – que, quando a *Low*, força a passagem imediata do *flip-flop* ao estado '0'.

P3: Seja um diagrama de estados, e analise-se um estado qualquer, seja A; considere-se uma qualquer das *setas que partem de* A; ela terá um rótulo com o formato genérico ***in/out*** (porém, *vidé* Preâmbulo P4) e apontará a um estado, seja B. O significado – da ***seta + rótulo*** – é o seguinte:

Admita-se que o *estado actual* do circuito é A – i.e., que em dado momento o circuito ingressou no estado A (e ainda não o abandonou); então,

- no *estado actual*, A, enquanto a entrada se mantiver em *in*, a saída do circuito será *out*.
- se, no impulso de relógio seguinte, a *entrada* for *in*, o circuito passará ao estado B – que por isso se diz "*seguinte*";

Nota 1: o estado *seguinte* não tem que ser distinto do *actual*; pode suceder que, estando em A, e ao menos para uma dada entrada, o circuito aí continue: a seta dirige-se de A para A - vulgo *pescadinha de rabo-na-boca*...

P4: **Modelos de Mealy e Moore**

Um *diagrama-de-estados/circuito* pode seguir um de dois *modelos*, a saber: ***Mealy*** e ***Moore***...:

- diz-se seguir o modelo de Mealy se, *ao menos para um* estado, a saída, ***out***, associada a *ao menos uma* entrada é distinta da saída associada a *alguma outra* entrada: *a saída depende desse estado* ***e*** *do valor da entrada,* ***in***;

- diz-se seguir o modelo de *Moore* se, para *todos* os estados, se verificar o seguinte: a todas as setas que partem de cada estado está associada a *mesma saída*, ***out*** – independentemente da entrada, ***in***. Nesse caso, os rótulos são reduzidos a ***in***, e a terminação ***/out*** é transferida para o seio *do círculo que representa o estado*.

1. [10T2.4,10E1.9] Desenhe o diagrama de estados de um circuito sequencial que detecte quando ocorre uma contagem ascendente ou descendente completa nas suas entradas. O circuito deverá ter 2 entradas, A e B, e uma saída, Z. Ou seja, pretende-se um circuito cuja saída só é activa após a ocorrência de uma das seguintes sequências de valores nas entradas: 00→01→10→11; 11→10→01→00. O circuito deverá funcionar continuamente. Utilize uma máquina de Mealy.

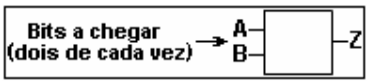
R: O diagrama de estados encontra-se em SeqSynth01.6.

Processo mental: Convirá porventura gastar algum tempo a esmiuçar o enunciado… O contexto é o de uma "caixa preta" com duas entradas, A e B, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sequência (em que os primeiros bits a chegar são os *dois da primeira coluna à esquerda*, {0, 0}):

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

(Menor peso) *(Maior peso)*

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *4 pares de bits mais recentes* perfizerem uma *contagem crescente ou decrescente completa*:



A	Crescente:	0	1	0	1	ou	Decrescente:	1	0	1	0
B		0	0	1	1			1	1	0	0

Pense o leitor que lhe pedem para fazer de "caixa preta"… Dão-lhe um par de bits, {00} – e o leitor provavelmente anotará "Boa! Já tenho o 1º par da sequência (crescente)!". Depois, dão-lhe outro par de bits, no caso {10} – e o leitor anotará "Azar! A sequência foi *abortada*, tenho que esperar que me apareça outro {00}, ou então {11}, para começar de novo!...". Depois, dão-lhe outro par de bits, no caso {11} – e o leitor anotará "Boa! Já tenho o 1º par da sequência (decrescente)!...". Para facilitar a vida, transcreve-se a sequência acima, todavia em decimal, que será mais cómodo acompanhá-la:

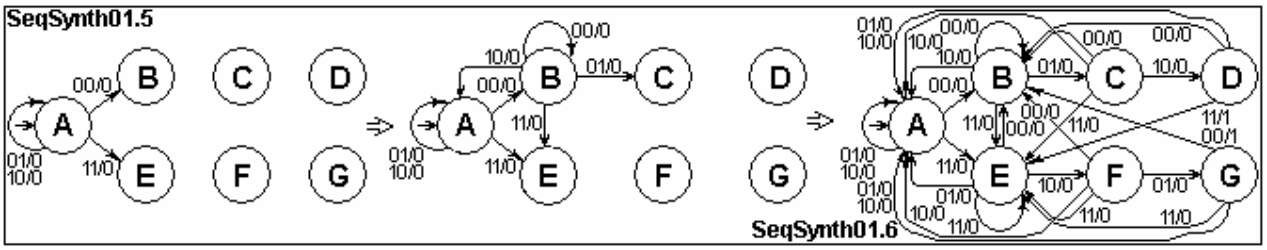
	0	2	3	0	1	2	2	3	1	3	0	**3**	**2**	**1**	**0**	1	3	0	**0**	**1**	**2**	**3**	**2**	**1**	**0**	2	2	1	3	3
Instantes:	1º	2º	**3º**	**4º**	5º	6º	7º	**8º**	9º	**10º**	**11º**	**12º**	13º	14º	**15º**	16º	**17º**	**18º**	**19º**	20º	21º	**22º**	23º	24º	**25º**	26º	27º	28º	**29º**	**30º**

Na tabela, realçaram-se os instantes em que se detectou o *começo de uma sequência* e bem assim as *sequências completas detectadas*: ocorrem nos instantes {12º a 15º}, {19º a 22º} e {22º a 25º}. Isto é: após alguns recomeços (nos instantes 3º, 4º, de novo no 8º, 10º e 11º), os pares de bits {12º a 15º} perfizeram uma sequência decrescente; etc…. Repare-se que, nos instantes 15º e 22º, não apenas chegaram ao fim duas sequências – como, ademais, começaram outras; aquela que começou no instante 15º veio a abortar no instante 17º, mas aquela que começou no instante 22º veio a completar-se com sucesso, no instante 25º!

Este "faz de conta" quiçá não seja tempo inútil, no sentido em que faz vir ao de cima *o que o leitor precisa lembrar-se* para exclamar: "Boa! A sequência acabou: o par que recebi era mesmo aquele que faltava!...". Será provavelmente o seguinte:

A	- Tenho que esperar que me apareça {00}, ou {11}, para *começar* uma sequência…
B	- Boa! Já tenho o 1º par de uma sequência crescente!
C	- Boa! Já tenho o 2º par de uma sequência crescente!
D	- Boa! Já tenho o 3º par de uma sequência crescente!
E	- Boa! Já tenho o 1º par de uma sequência decrescente!
F	- Boa! Já tenho o 2º par de uma sequência decrescente!
G	- Boa! Já tenho o 3º par de uma sequência decrescente!

Na coluna à esquerda, inscreveram-se letras *maiúsculas*; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…



- a primeira coisa a fazer é *desenhar sete batatas* (portuguesas, é claro – que são as melhores!), *rotuladas com as maiúsculas associadas a cada estado*: {A, B, C, …, G};
- considere-se agora o estado A; aguarda-se um *par de bits* – pelo que há que tratar cada uma das hipóteses possíveis:
  - suponha-se que se recebe {00}; então, "Boa!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 00/0 dirigida de A para B (no modelo de Mealy, o formato do rótulo é: *entrada/saída*);
  - suponha-se que se recebe {01}; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 01/0 dirigida de A para A;
  - suponha-se que se recebe {10}; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 10/0 dirigida de A para A;
  - suponha-se que se recebe {11}; então, "Boa!": ingressa-se no estado E – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 11/0 dirigida de A para E (*vidé* diagrama em SeqSynth01.5)
- adivinha-se o resto: estado a estado {B, C, …, G}, exploram-se as quatro combinações possíveis que o par de bits recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes…

Resumidamente, ir-se-á verificar o seguinte:

- o par {00} dá sempre azo a uma seta dirigida para B: está-se começando uma nova sequência ascendente…
- o par {11} dá sempre azo a uma seta dirigida para E: está-se começando uma nova sequência descendente…
- o par {01} dá azo a uma seta dirigida para A – excepto em dois casos: foi recebido em B (e então a seta é dirigida a C) ou em F (e então a seta é dirigida a G)…
- o par {10} dá azo a uma seta dirigida para A – excepto em dois casos: foi recebido em C (e então a seta é dirigida a D) ou em E (e então a seta é dirigida a F)…

Quanto à saída Z, ela é sempre '**0**' excepto quando, estando em D, se receber {11}, ou, estando em G, se receber {00}…

2. [10E2.10] Determine o diagrama de estados de uma máquina de Moore que detecta se um inteiro não negativo, representado por um bloco de 3 bits, pertence ao intervalo 0 a 4. Os bits são apresentados em série à máquina numa única entrada, começando pelo de maior peso. Ao receber o bit menos significativo a máquina activa a saída durante um período de relógio se o número estiver na gama pretendida, e fica pronta a processar o próximo bloco de 3 bits (ou seja, o sistema opera sem sobreposição). Justifique as opções tomadas.

R: O diagrama de estados encontra-se em SeqSynth01.13.

Processo mental: O contexto é o de uma "caixa preta" com uma entrada, A, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sequência (em que o primeiro bit a chegar é o *da primeira coluna à esquerda*, {0}):

A	**0**	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	0	0	1	1	1
Z	**0**	0	0	**1**	0	0	**1**	0	0	**1**	0	0	**0**	0	0	**1**	0	0	**0**	0	0	**1**	0	0	**0**	0	0	**1**	0	0

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *3 bits mais recentes* perfizerem um *inteiro não superior a 4* (considera-se que, em cada 3-bits, o primeiro a chegar é o de maior peso).

Pense o leitor que lhe pedem para fazer de "caixa preta"… Olha o primeiro bit, {0} – e o leitor provavelmente anotará "O número está entre 0 e 3 – mas recebi somente o 1º!". Depois, recebe outro bit – e o leitor anotará "Seja o que for que tenha chegado, o número estará entre 0 e 3 – mas recebi somente até ao 2º". Depois, recebe outro bit – e o leitor anotará "O número está entre 0 e 3, já posso assinalá-lo!"…

Com este "faz de conta", vem ao de cima *o que o leitor precisa lembrar-se* para exclamar: "O número está entre 0 e 3 – e já recebi os 3-bits". Será provavelmente o seguinte:

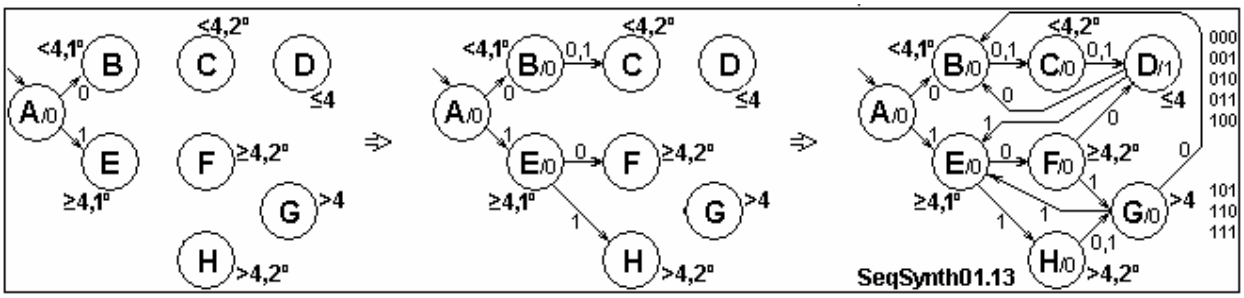
A	- Tenho que esperar que me apareça um bit…
B	- O número está entre 0 e 3 – mas recebi somente o 1º!
C	- O número está entre 0 e 3 – mas recebi somente os dois primeiros!
D	- O número está entre 0 e 3, já posso assinalá-lo!
E	- O número excede 3 – mas recebi somente o 1º!
F	- O número excede 3 – mas recebi somente os dois primeiros!
G	- O número excede 4, já recebi os 3-bits!!
H	- O número excede 4 – mas recebi somente os dois primeiros!

Na coluna à esquerda, inscreveram-se letras maiúsculas; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…

- a primeira coisa a fazer é *desenhar oito batatas*, *rotuladas com as maiúsculas associadas a cada estado*: {A, B, …, H};
- considere-se o estado A; aguarda-se o primeiro de 3-*bits* – pelo que há que tratar cada uma das hipóteses possíveis:
  - suponha-se que se recebe {0}; então, "O número está entre 0 e 3 – mas recebi somente o 1º!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 0 dirigida de A para B (no modelo de Moore, o formato do rótulo é: *entrada*), e rotulando esse *estado seguinte*, B, com '/0';
  - suponha-se que se recebe {1}; então, "O número excede 3 – mas recebi somente o 1º": ingressa-se no estado E – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 1 dirigida de A para E, e rotulando E com '/0';
- adivinha-se o resto: estado a estado {B, C, …, H}, exploram-se os valores possíveis que cada bit recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes, e rotulam-se os estados com a saída que devem apresentar…

No diagrama, afixaram-se, à direita, os 3-bits passíveis de conduzir o circuito aos estados **D** e **G**; são, respectivamente, as representações binárias (de 3-bit) dos "inteiros não negativos entre 0 e 4", e "superiores a 4". Constata-se que, não por acaso, as sequências de rótulos que de A conduzem a D ou G são um desses 3-bits…

Encontrando-se o circuito em **D** ou **G**, a recepção dum novo bit fá-lo mudar de estado: *recomeça a análise a um 3-bit*; em termos práticos, o resultado será análogo àquele havido no estado inicial, A: irá ingressar em B ou E, consoante esse bit for, respectivamente, '0' ou '1'…



3. [10E4.9] Desenhe o diagrama de estados de um circuito sequencial que detecta um dos dois padrões "1101" ou "1110", isto é: a saída do circuito estará normalmente a "0", ficando a "1" se e apenas se os últimos 4 bits tiverem sido "1101" ou "1110".

R: O diagrama de estados encontra-se em SeqSynth01.17.

Processo mental: Convirá porventura gastar algum tempo a esmiuçar o enunciado… O contexto é o de uma "caixa preta" com uma entrada, A, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sucessão de bits (em que o primeiro a chegar é o *primeiro à esquerda*, '**0**'):

A: 0 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *4 bits mais recentes* perfizerem uma *das sequências: 1101 ou 1110*:

Pense o leitor que lhe pedem para fazer de "caixa preta"… Dão-lhe um bit, '**0**' – e o leitor provavelmente anotará "Pouca sorte! Nenhuma das sequências alvo começa por '**0**'! Depois, dão-lhe outro bit, no caso '**1**' – e o leitor anotará "Boa! Já tenho o 1º bit de uma sequência, embora não saiba ainda qual!"

Com o bit seguinte, '**0**', o leitor provavelmente anotará" Azar! A sequência foi *abortada*, tenho que esperar que me apareça outro '**1**', para começar de novo!..."

Com o bit seguinte, novamente '**1**', o leitor voltará a anotar: "Boa! Já tenho o 1º bit de uma das sequências!..."

Para facilitar a vida, transcreve-se a sucessão recebida, realçando os instantes em que se detectou o *começo de uma sequência* e bem assim as *sequências completas detectadas*: ocorrem nos instantes {4º a 7º}, {7º a 10º}, {8º a 11º}, {14º a 17º] e [15º a 18º]

	0	1	0	1	1	0	1	1	1	0	1	0	1	1	1	1	0	1	1	0
Instantes:	1º	**2º**	3º	**4º**	5º	6º	**7º**	**8º**	9º	10º	**11º**	12º	**13º**	**14º**	**15º**	16º	17º	**18º**	19º	20º

Repare-se que, no instante 7º, não apenas chegou ao fim a sequência "**1101**" – como, ademais, começou outra; quando aquela que começou no instante 7º acabou, já havia começado uma outra – a que começou no instante 8º! Etc…

Este "faz de conta" quiçá não seja tempo inútil, no sentido em que faz vir ao de cima *o que o leitor precisa lembrar-se* para exclamar: "Os deuses sorriram-me! Detectei uma das sequências, o bit que acabei de receber era mesmo aquele que faltava para terminar uma das sequências!...". Será provavelmente o seguinte:

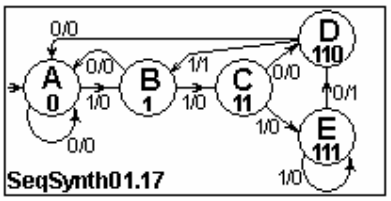
A - Tenho que esperar que me apareça '**1**', para *começar* uma sequência…

B - Boa! Já tenho o primeiro '**1**' de uma sequência!

C - Boa! Já tenho os dois primeiros '**1**' de uma sequência!

D - Boa! Já tenho o trio '**110**' com que uma das sequências começa!

E - Boa! Já tenho o trio '**111**' com que a outra sequência começa!



Á esquerda, inscreveram-se letras *maiúsculas*; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…

- a primeira coisa a fazer é desenhar cinco batatas, rotuladas com as maiúsculas associadas a cada estado: {A, B, C, …, E};

- considere-se agora o estado A; aguarda-se um *bit* – pelo que há que tratar cada uma das hipóteses possíveis:

- suponha-se que se recebe '**1**'}; então, "Boa!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 1/0 dirigida de A para B (no modelo de Mealy, o formato do rótulo é: *entrada/saída*);

- suponha-se que se recebe '**0**'; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 0/0 dirigida de A para A;

- adivinha-se o resto: estado a estado {B, C, …, E}, exploram-se as duas combinações possíveis que o bit recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes…

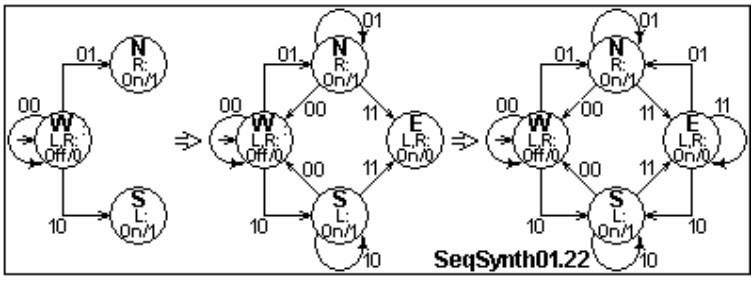
Quanto à saída Z, ela é sempre '**0**' excepto quando, estando em D, se receber '**1**', ou, estando em E, se receber '**0**'…

4. [10E3.9] Determine o diagrama de estados de uma máquina de Moore que comanda uma lâmpada utilizando 2 interruptores distintos, tal como acontece numa escada em que existe um interruptor no fundo da escada e outro no cimo da escada. De cada vez que se acciona um dos interruptores, a lâmpada vai trocar o seu estado, i.e., se estava apagada, acende, se estava acesa, apaga. Assuma que cada interruptor tem 2 posições possíveis e que nunca é possível mudar os 2 interruptores simultaneamente. Assuma igualmente que o estado inicial do circuito corresponde a ter ambos os interruptores na posição '0' e que nessa situação a lâmpada está apagada.

R: SeqSynth01.22 apresenta um diagrama de estados possível (incluindo, à esquerda, as primeiras etapas na sua construção). Assumindo que a escada sobe da esquerda para a direita, os dois interruptores foram designados de {**L**eft,**R**ight}, cada um podendo ter as posições **0** (Off) ou **1** (On).

Processo mental:

- De início, {L,R} estão Off, o circuito encontra-se no estado inicial **W** – a sua saída estando '**0**' (apagada); continuando {L,R} Off, o estado não se altera (o que se representa com uma seta saindo de **W** e a **W** retornando); se {L} ou {R} passarem a On, o estado muda (o que se representa com uma seta de **W** para, respectivamente **S** ou **N**); em **W**, nunca acontece {L,R}



ficarem simultaneamente On...

- Em **N** ou **S**, os interruptores podem voltar ambos a Off (o que se representa por setas saindo desses estados e dirigidas para **W**); ou aquele que ainda estava Off pode passar a On (o que se representa por setas saindo desses estados e dirigidas para **E**); ou podem não se alterar (o que se representa por setas saindo desses estados e a eles retornando)...
- Em **E**, um dos interruptores pode voltar a Off (o que se representa por setas saindo de **E** e dirigidas para **N** ou **S**, consoante o caso); ou podem não se alterar (o que se representa por um seta saindo de **E** e a ele retornando)...
- Relativamente à *saída*, ela é '**0**' nos estados **W** e **E**, e é '**1**' nos estados **N** e **S**. Essa especificação representa-se pela inscrição, nesses estados, de '**/**' seguido da saída que esse estado deve produzir...

Curiosidade: Admita-se que se pretende desenhar o diagrama de estados – e que para o efeito se escolhe uma *codificação em que os códigos de* **N** *e* **S** *são adjacentes*, seja N→11 e S→10, ficando, por ex., W→00 e E→01. A tabela de estados, e as subsequentes tabelas de excitação dos *flip-flops* (D, para simplificar) escrever-se-iam (já sob a forma de quadro de Karnaugh):

			0	0	1	1	0	0	1	1		L
			0	1	1	0	0	1	1	0		R
W	0	0	0	1	X	1	0	1	X	0	0	
E	0	1	X	1	0	1	X	1	1	0	0	
N	1	1	0	1	0	X	0	1	1	X	1	
S	1	0	0	X	0	1	0	X	1	0	1	
	Q_1	Q_0	D_1				D_0				Z	

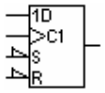
Por conseguinte, viria: $Z = Q_1$, $D_1 = L \oplus R$ (e $D_0 = R$). Isto é: Z depende apenas da saída dum *flip-flop*, e as entradas desse *flip-flop* não dependem do outro: essoutro, Q_0, é *supérfluo* – e será pacífico chegar o leitor a um diagrama que, ao invés de apresentar os quatro estados em SeqSynth01.22, comporte apenas, um par deles, seja {Apagado} e {Aceso}... Reflectindo sobre isso: poderá alguém aventar que até mesmo Q_1 é dispensável, o circuito ficando reduzido a uma malha combinatória: um XOR de L e R produzindo imediatamente a saída Z... Mas não é bem assim: a saída desse XOR mudaria logo que alguma das entradas mudasse – enquanto que na solução "circuito sequencial" a saída só mudaria no impulso de relógio seguinte.

Síntese de Circuitos Sequenciais Síncronos

{SeqSynth_r.doc}

Preâmbulo:

P1: **Flip-Flop D**

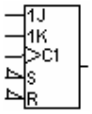


O dispositivo figurado vem a ser um *flip-flop* D. Em ordem a proporcionar o *carregamento com um valor arbitrário*, disponibiliza uma entrada 'D'.

Tabela de excitação do *flip-flop* D (valores a impor à entrada, para lograr uma transição):

Transição que se pretende	Entrada a aplicar
Estado *actual* → Estado *seguinte*	**D**
x → 0	0
x → 1	1

P2: **Flip-Flop JK**



O dispositivo figurado vem a ser um *flip-flop* JK. Em ordem a proporcionar o *carregamento com um valor arbitrário*, disponibiliza um par de entradas 'J,K',

Tabela de excitação do *flip-flop* JK (valores a impor à entrada, para lograr uma transição):

Transição que se pretende	Entradas a aplicar	
Estado *actual* → Estado *seguinte*	**J**	**K**
0 → 0	0	X
0 → 1	1	X
1 → 0	X	1
1 → 1	X	0

Mnemónica: "quando Q_n=0, Q_{n+1}=J; quando Q_n=1, Q_{n+1}=K , ou seja: se o Estado de *partida* é '0', força-se em J o Estado *seguinte*; se for '1', força-se em K o *complemento* do Estado seguinte".

Atributos comuns dos dispositivos figurados:

Para o sinal de relógio, CLK, é disponibilizada uma entrada, C; o *flip-flop* reage ao flanco ascendente do CLK: repare-se no símbolo (▷) que acompanha essa entrada. As entradas ('D' e 'J,K') são sincronizadas pelo CLK (*vidé* ***sufixo*** '**1**' em C**1** e ***prefixo*** '**1**' em **1**D e em **1**J,**1**K)

Os *flip-flops* figurado têm ainda duas entradas *assíncronas*:

- uma entrada ***S*** (Set) – que, quando a *Low*, força a passagem imediata do *flip-flop* ao estado '1';
- uma entrada ***R*** (Reset) – que, quando a *Low*, força a passagem imediata do *flip-flop* ao estado '0'.

P3: Seja um diagrama de estados, e analise-se um estado qualquer, seja A; considere-se uma qualquer das *setas que partem de* A; ela terá um rótulo com o formato genérico ***in/out*** (porém, *vidé* Preâmbulo P4) e apontará a um estado, seja B. O significado – da ***seta + rótulo*** – é o seguinte:

Admita-se que o *estado actual* do circuito é A – i.e., que em dado momento o circuito ingressou no estado A (e ainda não o abandonou); então,

- no *estado actual*, A, enquanto a entrada se mantiver em *in*, a saída do circuito será *out*.
- se, no impulso de relógio seguinte, a *entrada* for *in*, o circuito passará ao estado B – que por isso se diz "*seguinte*";

Nota 1: o estado *seguinte* não tem que ser distinto do *actual*; pode suceder que, estando em A, e ao menos para uma dada entrada, o circuito aí continue: a seta dirige-se de A para A - vulgo *pescadinha de rabo-na-boca*...

P4: **Modelos de Mealy e Moore**

Um *diagrama-de-estados/circuito* pode seguir um de dois *modelos*, a saber: ***Mealy*** e ***Moore***…:

- diz-se seguir o modelo de Mealy se, *ao menos para um* estado, a saída, ***out***, associada a *ao menos uma* entrada é distinta da saída associada a *alguma outra* entrada: *a saída depende desse estado* ***e*** *do valor da entrada,* ***in***;

- diz-se seguir o modelo de *Moore* se, para *todos* os estados, se verificar o seguinte: a todas as setas que partem de cada estado está associada a *mesma saída*, ***out*** – independentemente da entrada, ***in***. Nesse caso, os rótulos são reduzidos a ***in***, e a terminação ***/out*** é transferida para o seio *do círculo que representa o estado*.

Desenho de Diagramas de Estado

1. [10T2.4,10E1.9] Desenhe o diagrama de estados de um circuito sequencial que detecte quando ocorre uma contagem ascendente ou descendente completa nas suas entradas. O circuito deverá ter 2 entradas, A e B, e uma saída, Z. Ou seja, pretende-se um circuito cuja saída só é activa após a ocorrência de uma das seguintes sequências de valores nas entradas: 00→01→10→11; 11→10→01→00. O circuito deverá funcionar continuamente. Utilize uma máquina de Mealy.

R: O diagrama de estados encontra-se em SeqSynth01.6.

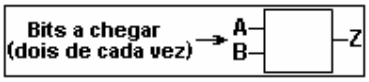
Processo mental: Convirá porventura gastar algum tempo a esmiuçar o enunciado… O contexto é o de uma "caixa preta" com duas entradas, A e B, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sequência (em que os primeiros bits a chegar são os *dois da primeira coluna à esquerda*, {0, 0}):

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

(Menor peso)

(Maior peso)

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *4 pares de bits mais recentes* perfizerem uma *contagem crescente ou decrescente completa*:



A	Crescente:	0	1	0	1	ou	Decrescente:	1	0	1	0
B		0	0	1	1			1	1	0	0

Pense o leitor que lhe pedem para fazer de "caixa preta"… Dão-lhe um par de bits, {00} – e o leitor provavelmente anotará "Boa! Já tenho o 1º par da sequência (crescente)!". Depois, dão-lhe outro par de bits, no caso {10} – e o leitor anotará "Azar! A sequência foi *abortada*, tenho que esperar que me apareça outro {00}, ou então {11}, para começar de novo!...". Depois, dão-lhe outro par de bits, no caso {11} – e o leitor anotará "Boa! Já tenho o 1º par da sequência (decrescente)!...". Para facilitar a vida, transcreve-se a sequência acima, todavia em decimal, que será mais cómodo acompanhá-la:

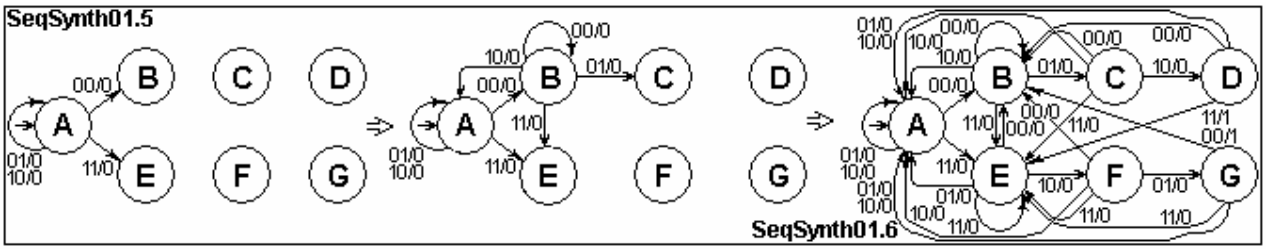
	0	2	3	0	1	2	2	3	1	3	0	**3**	**2**	**1**	**0**	1	3	0	**0**	**1**	**2**	**3**	**2**	**1**	**0**	2	2	1	3	3
Instantes:	1º	2º	**3º**	**4º**	5º	6º	7º	**8º**	9º	**10º**	**11º**	**12º**	13º	14º	**15º**	16º	**17º**	**18º**	**19º**	20º	21º	**22º**	23º	24º	**25º**	26º	27º	28º	**29º**	**30º**

Na tabela, realçaram-se os instantes em que se detectou o *começo de uma sequência* e bem assim as *sequências completas detectadas*: ocorrem nos instantes {12º a 15º}, {19º a 22º} e {22º a 25º}. Isto é: após alguns recomeços (nos instantes 3º, 4º, de novo no 8º, 10º e 11º), os pares de bits {12º a 15º} perfizeram uma sequência decrescente; etc…. Repare-se que, nos instantes 15º e 22º, não apenas chegaram ao fim duas sequências – como, ademais, começaram outras; aquela que começou no instante 15º veio a abortar no instante 17º, mas aquela que começou no instante 22º veio a completar-se com sucesso, no instante 25º!

Este "faz de conta" quiçá não seja tempo inútil, no sentido em que faz vir ao de cima *o que o leitor precisa lembrar-se* para exclamar: "Boa! A sequência acabou: o par que recebi era mesmo aquele que faltava!...". Será provavelmente o seguinte:

A	- Tenho que esperar que me apareça {00}, ou {11}, para *começar* uma sequência…
B	- Boa! Já tenho o 1º par de uma sequência crescente!
C	- Boa! Já tenho o 2º par de uma sequência crescente!
D	- Boa! Já tenho o 3º par de uma sequência crescente!
E	- Boa! Já tenho o 1º par de uma sequência decrescente!
F	- Boa! Já tenho o 2º par de uma sequência decrescente!
G	- Boa! Já tenho o 3º par de uma sequência decrescente!

Na coluna à esquerda, inscreveram-se letras *maiúsculas*; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…



- a primeira coisa a fazer é *desenhar sete batatas* (portuguesas, é claro – que são as melhores!), *rotuladas com as maiúsculas associadas a cada estado*: {A, B, C, …, G};

- considere-se agora o estado A; aguarda-se um *par de bits* – pelo que há que tratar cada uma das hipóteses possíveis:

  - suponha-se que se recebe {00}; então, "Boa!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 00/0 dirigida de A para B (no modelo de Mealy, o formato do rótulo é: *entrada/saída*);

  - suponha-se que se recebe {01}; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 01/0 dirigida de A para A;

  - suponha-se que se recebe {10}; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 10/0 dirigida de A para A;

  - suponha-se que se recebe {11}; então, "Boa!": ingressa-se no estado E – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 11/0 dirigida de A para E (*vidé* diagrama em SeqSynth01.5)

- adivinha-se o resto: estado a estado {B, C, …, G}, exploram-se as quatro combinações possíveis que o par de bits recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes…

Resumidamente, ir-se-á verificar o seguinte:

- o par {00} dá sempre azo a uma seta dirigida para B: está-se começando uma nova sequência ascendente…

- o par {11} dá sempre azo a uma seta dirigida para E: está-se começando uma nova sequência descendente…

- o par {01} dá azo a uma seta dirigida para A – excepto em dois casos: foi recebido em B (e então a seta é dirigida a C) ou em F (e então a seta é dirigida a G)…

- o par {10} dá azo a uma seta dirigida para A – excepto em dois casos: foi recebido em C (e então a seta é dirigida a D) ou em E (e então a seta é dirigida a F)…

Quanto à saída Z, ela é sempre '**0**' excepto quando, estando em D, se receber {11}, ou, estando em G, se receber {00}…

2. [10E2.10] Determine o diagrama de estados de uma máquina de Moore que detecta se um inteiro não negativo, representado por um bloco de 3 bits, pertence ao intervalo 0 a 4. Os bits são apresentados em série à máquina numa única entrada, começando pelo de maior peso. Ao receber o bit menos significativo a máquina activa a saída durante um período de relógio se o número estiver na gama pretendida, e fica pronta a processar o próximo bloco de 3 bits (ou seja, o sistema opera sem sobreposição). Justifique as opções tomadas.

R: O diagrama de estados encontra-se em SeqSynth01.13.

Processo mental: O contexto é o de uma "caixa preta" com uma entrada, A, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sequência (em que o primeiro bit a chegar é o *da primeira coluna à esquerda*, {0}):

A	**0**	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	0	0	1	1	1
Z	**0**	0	0	**1**	0	0	**1**	0	0	**1**	0	0	**0**	0	0	**1**	0	0	**0**	0	0	**1**	0	0	**0**	0	0	**1**	0	0

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *3 bits mais recentes* perfizerem um *inteiro não superior a 4* (considera-se que, em cada 3-bits, o primeiro a chegar é o de maior peso).

Pense o leitor que lhe pedem para fazer de "caixa preta"… Olha o primeiro bit, {0} – e o leitor provavelmente anotará "O número está entre 0 e 3 – mas recebi somente o 1º!". Depois, recebe outro bit – e o leitor anotará "Seja o que for que tenha chegado, o número estará entre 0 e 3 – mas recebi somente até ao 2º". Depois, recebe outro bit – e o leitor anotará "O número está entre 0 e 3, já posso assinalá-lo!"…

Com este "faz de conta", vem ao de cima *o que o leitor precisa lembrar-se* para exclamar: "O número está entre 0 e 3 – e já recebi os 3-bits". Será provavelmente o seguinte:

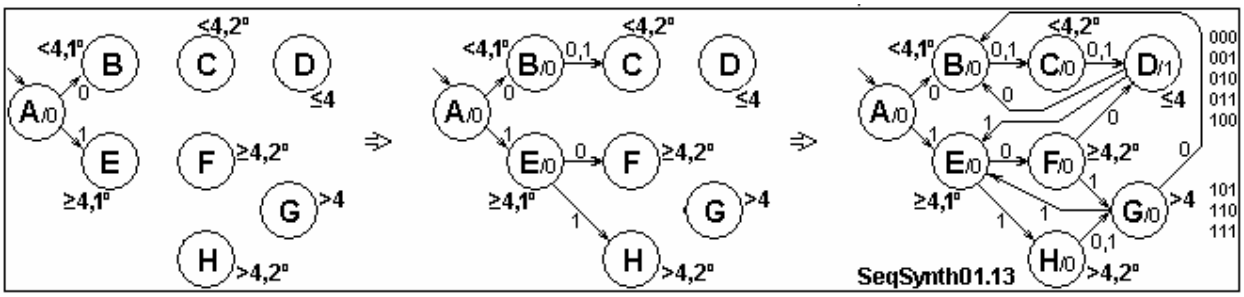
A	- Tenho que esperar que me apareça um bit…
B	- O número está entre 0 e 3 – mas recebi somente o 1º!
C	- O número está entre 0 e 3 – mas recebi somente os dois primeiros!
D	- O número está entre 0 e 3, já posso assinalá-lo!
E	- O número excede 3 – mas recebi somente o 1º!
F	- O número excede 3 – mas recebi somente os dois primeiros!
G	- O número excede 4, já recebi os 3-bits!!
H	- O número excede 4 – mas recebi somente os dois primeiros!

Na coluna à esquerda, inscreveram-se letras maiúsculas; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…

- a primeira coisa a fazer é *desenhar oito batatas*, *rotuladas com as maiúsculas associadas a cada estado*: {A, B, …, H};
- considere-se o estado A; aguarda-se o primeiro de 3-*bits* – pelo que há que tratar cada uma das hipóteses possíveis:
  - suponha-se que se recebe {0}; então, "O número está entre 0 e 3 – mas recebi somente o 1º!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 0 dirigida de A para B (no modelo de Moore, o formato do rótulo é: *entrada*), e rotulando esse *estado seguinte*, B, com '/0';
  - suponha-se que se recebe {1}; então, "O número excede 3 – mas recebi somente o 1º": ingressa-se no estado E – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 1 dirigida de A para E, e rotulando E com '/0';
- adivinha-se o resto: estado a estado {B, C, …, H}, exploram-se os valores possíveis que cada bit recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes, e rotulam-se os estados com a saída que devem apresentar…

No diagrama, afixaram-se, à direita, os 3-bits passíveis de conduzir o circuito aos estados **D** e **G**; são, respectivamente, as representações binárias (de 3-bit) dos "inteiros não negativos entre 0 e 4", e "superiores a 4". Constata-se que, não por acaso, as sequências de rótulos que de A conduzem a D ou G são um desses 3-bits…

Encontrando-se o circuito em **D** ou **G**, a recepção dum novo bit fá-lo mudar de estado: *recomeça a análise a um 3-bit*; em termos práticos, o resultado será análogo àquele havido no estado inicial, A: irá ingressar em B ou E, consoante esse bit for, respectivamente, '0' ou '1'…



3. [10E4.9] Desenhe o diagrama de estados de um circuito sequencial que detecta um dos dois padrões "1101" ou "1110", isto é: a saída do circuito estará normalmente a "0", ficando a "1" se e apenas se os últimos 4 bits tiverem sido "1101" ou "1110".

R: O diagrama de estados encontra-se em SeqSynth01.17.

Processo mental: Convirá porventura gastar algum tempo a esmiuçar o enunciado… O contexto é o de uma "caixa preta" com uma entrada, A, onde estão chegando bits à cadência dos impulsos de relógio. Para exemplificar, considere-se que está chegando a seguinte sucessão de bits (em que o primeiro a chegar é o *primeiro à esquerda*, '**0**'):

A: 0 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0

A saída, Z, está em geral *apagada*; deve *acender-se* quando e só quando os *4 bits mais recentes* perfizerem uma *das sequências: 1101 ou 1110*:

Pense o leitor que lhe pedem para fazer de "caixa preta"… Dão-lhe um bit, '**0**' – e o leitor provavelmente anotará "Pouca sorte! Nenhuma das sequências alvo começa por '**0**'! Depois, dão-lhe outro bit, no caso '**1**' – e o leitor anotará "Boa! Já tenho o 1º bit de uma sequência, embora não saiba ainda qual!"

Com o bit seguinte, '**0**', o leitor provavelmente anotará" Azar! A sequência foi *abortada*, tenho que esperar que me apareça outro '**1**', para começar de novo!..."

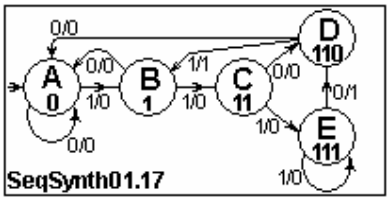
Com o bit seguinte, novamente '**1**', o leitor voltará a anotar: "Boa! Já tenho o 1º bit de uma das sequências!..."

Para facilitar a vida, transcreve-se a sucessão recebida, realçando os instantes em que se detectou o *começo de uma sequência* e bem assim as *sequências completas detectadas*: ocorrem nos instantes {4º a 7º}, {7º a 10º}, {8º a 11º}, {14º a 17º] e [15º a 18º]

	0	1	0	1	1	0	1	1	1	0	1	0	1	1	1	1	0	1	1	0
Instantes:	1º	**2º**	3º	**4º**	5º	6º	**7º**	**8º**	9º	10º	**11º**	12º	**13º**	**14º**	**15º**	16º	17º	**18º**	19º	20º

Repare-se que, no instante 7º, não apenas chegou ao fim a sequência "**1101**" – como, ademais, começou outra; quando aquela que começou no instante 7º acabou, já havia começado uma outra – a que começou no instante 8º! Etc…

Este "faz de conta" quiçá não seja tempo inútil, no sentido em que faz vir ao de cima *o que o leitor precisa lembrar-se* para exclamar: "Os deuses sorriram-me! Detectei uma das sequências, o bit que acabei de receber era mesmo aquele que faltava para terminar uma das sequências!...". Será provavelmente o seguinte:



A - Tenho que esperar que me apareça '**1**', para *começar* uma sequência…
B - Boa! Já tenho o primeiro '**1**' de uma sequência!
C - Boa! Já tenho os dois primeiros '**1**' de uma sequência!
D - Boa! Já tenho o trio '**110**' com que uma das sequências começa!
E - Boa! Já tenho o trio '**111**' com que a outra sequência começa!

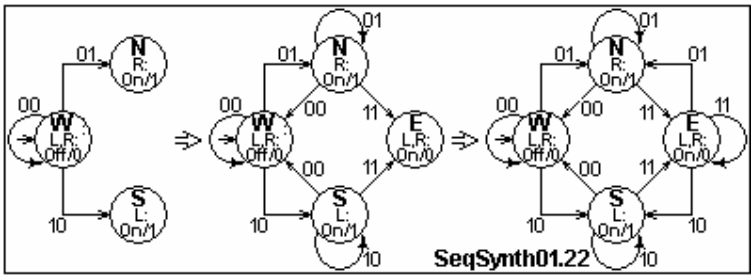
Á esquerda, inscreveram-se letras *maiúsculas*; a sua finalidade é representar as *lembranças* em causa. A partir daqui, é pacífico traçar o diagrama de estados…

- a primeira coisa a fazer é desenhar cinco batatas, rotuladas com as maiúsculas associadas a cada estado: {A, B, C, …, E};
- considere-se agora o estado A; aguarda-se um *bit* – pelo que há que tratar cada uma das hipóteses possíveis:
- suponha-se que se recebe '**1**'}; então, "Boa!": ingressa-se no estado B – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 1/0 dirigida de A para B (no modelo de Mealy, o formato do rótulo é: *entrada/saída*);
- suponha-se que se recebe '**0**'; então, "Azar!": continua-se no estado A – a saída Z continuando apagada… Isso representa-se por uma seta rotulada 0/0 dirigida de A para A;
- adivinha-se o resto: estado a estado {B, C, …, E}, exploram-se as duas combinações possíveis que o bit recebido pode assumir – e traçam-se as setas (e respectivos rótulos) que forem pertinentes…

Quanto à saída Z, ela é sempre '**0**' excepto quando, estando em D, se receber '**1**', ou, estando em E, se receber '**0**'…

4. [10E3.9] Determine o diagrama de estados de uma máquina de Moore que comanda uma lâmpada utilizando 2 interruptores distintos, tal como acontece numa escada em que existe um interruptor no fundo da escada e outro no cimo da escada. De cada vez que se acciona um dos interruptores, a lâmpada vai trocar o seu estado, i.e., se estava apagada, acende, se estava acesa, apaga. Assuma que cada interruptor tem 2 posições possíveis e que nunca é possível mudar os 2 interruptores simultaneamente. Assuma igualmente que o estado inicial do circuito corresponde a ter ambos os interruptores na posição '0' e que nessa situação a lâmpada está apagada.

R: SeqSynth01.22 apresenta um diagrama de estados possível (incluindo, à esquerda, as primeiras etapas na sua construção). Assumindo que a escada sobe da esquerda para a direita, os dois interruptores foram designados de {**L**eft,**R**ight}, cada um podendo ter as posições **0** (Off) ou **1** (On).



Processo mental:

- De início, {L,R} estão Off, o circuito encontra-se no estado inicial **W** – a sua saída estando '**0**' (apagada); continuando {L,R} Off, o estado não se altera (o que se representa com uma seta saindo de **W** e a **W** retornando); se {L} ou {R} passarem a On, o estado muda (o que se representa com uma seta de **W** para, respectivamente **S** ou **N**); em **W**, nunca acontece {L,R} ficarem simultaneamente On...
- Em **N** ou **S**, os interruptores podem voltar ambos a Off (o que se representa por setas saindo desses estados e dirigidas para **W**); ou aquele que ainda estava Off pode passar a On (o que se representa por setas saindo desses estados e dirigidas para **E**); ou podem não se alterar (o que se representa por setas saindo desses estados e a eles retornando)...
- Em **E**, um dos interruptores pode voltar a Off (o que se representa por setas saindo de **E** e dirigidas para **N** ou **S**, consoante o caso); ou podem não se alterar (o que se representa por um seta saindo de **E** e a ele retornando)...
- Relativamente à *saída*, ela é '**0**' nos estados **W** e **E**, e é '**1**' nos estados **N** e **S**. Essa especificação representa-se pela inscrição, nesses estados, de '**/**' seguido da saída que esse estado deve produzir...

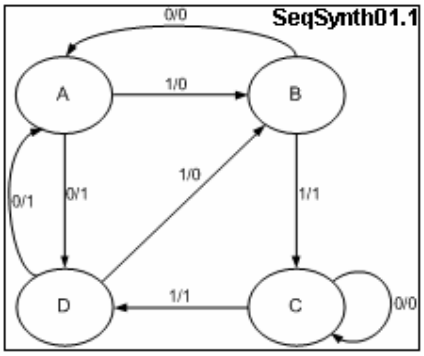
Curiosidade: Admita-se que se pretende desenhar o diagrama de estados – e que para o efeito se escolhe uma *codificação em que os códigos de **N** e **S** são adjacentes*, seja N→11 e S→10, ficando, por ex., W→00 e E→01. A tabela de estados, e as subsequentes tabelas de excitação dos *flip-flops* (D, para simplificar) escrever-se-iam (já sob a forma de quadro de Karnaugh):

			0	0	1	1	0	0	1	1		L
			0	1	1	0	0	1	1	0		R
W	0	0	0	1	X	1	0	1	X	0	0	
E	0	1	X	1	0	1	X	1	1	0	0	
N	1	1	0	1	0	X	0	1	1	X	1	
S	1	0	0	X	0	1	0	X	1	0	1	
	Q_1	Q_0	D_1				D_0				Z	

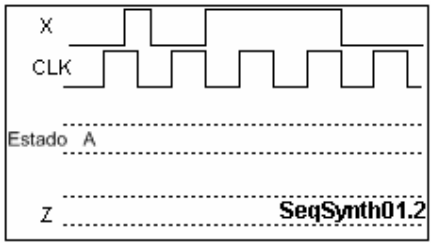
Por conseguinte, viria: $Z = Q_1$, $D_1 = L \oplus R$ (e $D_0 = R$). Isto é: Z depende apenas da saída dum *flip-flop*, e as entradas desse *flip-flop* não dependem do outro: essoutro, Q_0, é *supérfluo* – e será pacífico chegar o leitor a um diagrama que, ao invés de apresentar os quatro estados em SeqSynth01.22, comporte apenas, um par deles, seja {Apagado} e {Aceso}... Reflectindo sobre isso: poderá alguém aventar que até mesmo Q_1 é dispensável, o circuito ficando reduzido a uma malha combinatória: um XOR de L e R produzindo imediatamente a saída Z... Mas não é bem assim: a saída desse XOR mudaria logo que alguma das entradas mudasse – enquanto que na solução "circuito sequencial" a saída só mudaria no impulso de relógio seguinte.

Síntese Clássica

5. [10E2.9] Considere o diagrama de estados em SeqSynth01.1.



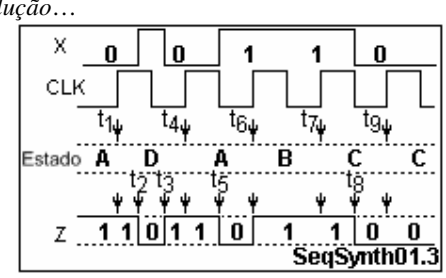
5. 1. Complete o diagrama temporal em SeqSynth01.2, indicando o estado do circuito e o valor da saída Z em cada instante. Considere que inicialmente o circuito se encontra no estado A, que Z = 1 e que os FF a utilizar reagem ao flanco ascendente do relógio:



5. 2. Utilizando a síntese clássica de circuitos sequenciais síncronos, obtenha as equações da entrada dos *Flip-Flops* e de Z, dum circuito que implemente a máquina de estados apresentada. Codifique os estados da seguinte forma (A=00, B=01, C=10, D=11).

R1: SeqSynth01.3 mostra a evolução de {Estado, Z}.

Processo mental: convirá, antes de desenhar o diagrama, traçar uma *tabela da evolução*…



- Numa primeira linha, inscrevem-se (*encostados à direita*) os instantes significativos – que no caso são: aqueles em que ocorre o *flanco ascendente do relógio* (t_1, t_4, t_6, t_7, t_9) e aqueles em que *a entrada de dados, X, se altera* (t_2, t_3, t_5, t_8), *vidé* SeqSynth01.3.

- Na segunda linha, inscrevem-se os valores que X tem *entre esses instantes* (mais precisamente: abaixo de, por ex., t_2, inscreve-se o valor de X *antes* de t_2);

- Nas duas linhas seguintes, inscrevem-se os estados e a saída do circuito, *após*/*entre* esses instantes significativos.

No que diz respeito ao estado, ele pode variar só aquando do *flanco ascendente do relógio*; entre esses flancos, ele permanece o mesmo. Para se saber o estado em o circuito ingressa *após* esses instantes, basta conhecer duas coisas: o estado em que ele se encontra – e a entrada X *imediatamente antes desse instante*. Quanto à saída *entre* dois instantes, ela depende do estado e da entrada entre esses dois instantes (*vidé* passos detalhados a este respeito mais abaixo).

	CLK: t_1	t_2	t_3	CLK: t_4	t_5	CLK: t_6	CLK: t_7	t_8	CLK: t_9	
Entrada X:	0	0	1	0	0	1	1	1	0	0
Estado	A	D			A		B	C		C
Saída Z:	1	1	0	1	1	0	1	1	0	0

Elaborada a tabela, é uma tarefa trivial preencher o diagrama SeqSynth01.3…

R2: Primeiro, converte-se o diagrama de estados em tabelas de estados/saída, já sob a forma "Karnaugh":

			Estado/Z		Estado		Z		Q_1		Q_0		D_1		D_0		
			0	1	0	1	0	1	0	1	0	1	0	1	0	1	X
A	0	0	D/1	B/0	11	01	1	0	1	0	1	1	1	0	1	1	
B	0	1	A/0	C/1	00	10	0	1	0	1	0	0	0	1	0	0	
D	1	1	A/1	B/0	00	01	1	0	0	0	0	1	0	0	0	1	
C	1	0	C/0	D/1	10	11	0	1	1	1	0	1	1	1	0	1	
	Q_1	Q_0															

Deduz-se $Z = \overline{X}(\overline{Q_1}\ \overline{Q_0} + Q_1Q_0) + X(\overline{Q_1}Q_0 + Q_1\overline{Q_0}) = \overline{X} \oplus Q_1 \oplus Q_0$

e, considerando o uso de *flip-flops* D (em que o estado seguinte assume o valor na sua entrada actual, $Q_{n+1}=D_n$), as equações de entrada dos *flip-flops*, ditas de *excitação*, vêm a ser:

$$D_1 = \overline{X}\ \overline{Q_0} + X\overline{Q_1}Q_0 + Q_1\overline{Q_0} = Q_0 \oplus (\overline{X} + Q_1) \ \textit{e}\ D_0 = \overline{Q_1}\ \overline{Q_0} + Q_1X$$

Comentário a R1: a *tabela de evolução* constrói-se através de 2 etapas:

- Primeiro, traça-se a evolução do Estado… Os instantes significativos são *aqueles em que ocorre o flanco ascendente do relógio* (CLK):

- em t_1, o estado é A, e X=0, então o estado *seguinte* será D (*vidé* seta rotulada **0/** saindo de **A** - terminando em D);
- em t_4, o estado é D, e X=0, então o estado seguinte será A (*vidé* seta rotulada **0/** saindo de **D** - terminando em A);
- em t_6, o estado é A, e X=1, então o estado seguinte será B (*vidé* seta rotulada **1/** saindo de **A** - terminando em B);
- em t_7, o estado é B, e X=1, então o estado seguinte será C (*vidé* seta rotulada **1/** saindo de **B** - terminando em C);
- em t_9, o estado é C, e X=0, então o estado seguinte será C (*vidé* seta rotulada **0/** saindo de **C** - terminando em C);

- Depois, traça-se a evolução da saída, Z… Os instantes significativos são {t_1 a t_9}, isto é, *todos aqueles em que ocorre uma mudança no Estado **ou** na entrada, X*:

- antes de t_1, o estado é A, e X=0, então a saída é 1 (*vidé* seta rotulada **0/** saindo de **A** – especificando /1);
- entre t_1 e t_2, o estado é D, e X=0, então a saída é 1 (*vidé* seta rotulada **0/** saindo de **D** – especificando /1);
- entre t_2 e t_3, o estado é D, e X=1, então a saída é 0 (*vidé* seta rotulada **1/** saindo de **D** – especificando /0);
- entre t_3 e t_4, o estado é D, e X=0, então a saída é 1 (*vidé* seta rotulada **0/** saindo de **D** – especificando /1);
- entre t_4 e t_5, o estado é A, e X=0, então a saída é 1 (*vidé* seta rotulada **0/** saindo de **A** – especificando /1);
- entre t_5 e t_6, o estado é A, e X=1, então a saída é 0 (*vidé* seta rotulada **1/** saindo de **A** – especificando /0);
- entre t_6 e t_7, o estado é B, e X=1, então a saída é 1 (*vidé* seta rotulada **1/** saindo de **B** – especificando /1);
- entre t_7 e t_8, o estado é C, e X=1, então a saída é 1 (*vidé* seta rotulada **1/** saindo de **C** – especificando /1);
- entre t_8 e t_9, o estado é C, e X=0, então a saída é 0 (*vidé* seta rotulada **0/** saindo de **C** – especificando /0);
- após t_9, o estado é C, e X=0, então a saída é 0 (*vidé* seta rotulada **0/** saindo de **C** – especificando /0);

Comentário a R2: Antes de mais, há que entender o que está em jogo na pergunta – e para tal quiçá SeqSynth01.11 seja um bom ponto de partida; representa um circuito cujo comportamento – aceite o leitor por ora – é o descrito pelo diagrama de estados dado, SeqSynth01.1. Um breve relance permite descortinar:

- que tem apenas uma entrada, **X** (no canto superior esquerdo), e uma saída, **Z** (próximo do canto superior direito);
- que o circuito se estrutura em *dois blocos*: um à esquerda, composto por apenas dois *flip-flops*-D, e um à direita, que é uma malha combinatória – composta de elementos lógicos triviais: NOTs, ANDs, ORs, XORs…

A função dos *flip-flops* {Q_1 Q_0} é óbvia: registam a "*lembrança do que já sucedeu até ao momento*"… Ou, mais formalmente: guardam o *estado do circuito*. Quando ocorre um impulso de relógio, e dependendo dos valores que as entradas {D_1 D_0} tiverem nesse momento, o circuito passa para um outro estado: as saídas {Q_1 Q_0} passam a ser outras…

A evolução dessas saídas {Q_1 Q_0} não é arbitrária: tem que estar conforme a SeqSynth01.1. Seja, como exemplo, o caso de {Q_1=1 Q_0=0} (isto é, o circuito estar no estado C) e a entrada ser X=1; as saídas dos *flip-flops* devem – no impulso de relógio seguinte – passar a ser {Q_1=**1** Q_0=**1**} (é o que o diagrama de estados estabelece: o circuito deve passar ao estado D). Como se consegue isso? Resposta: fazendo com que, *nesse instante*, as entradas dos *flip-flops* sejam precisamente, {D_1=**1** D_0=**1**}. Uma das funções da malha combinatória é precisamente forçar esse comportamento: gerar as entradas {D_1 D_0} – a partir do estado dos *flip-flops*, {Q_1 Q_0}, e da entrada do circuito, X. A outra função da malha combinatória é, naturalmente, gerar a própria saída do circuito, Z… A pergunta feita volve-se então quais as expressões booleanas dessa malha combinatória? E resolve-se da maneira geral, e trivial, de desenhar uma malha combinatória: determinar a tabela de verdade, simplificar…

As tabelas de estados/saída constroem-se através das seguintes etapas:

- Antes de mais, há que reparar que o diagrama envolve apenas 4 estados; pela *síntese clássica* (e pois que 2^2=4), bastam então apenas 2 *flip-flops*, designem-se de {Q_1, Q_0}. A primeira coisa a fazer é construir uma *tabela com 4 linhas* - isto é, tantas quantas as combinações possíveis que esses *flip-flops* podem assumir; antevendo a posterior aplicação do método de Karnaugh, convém rotulá-las com o *código binário reflectido*: {00, 01, 11, 10}. Antecedendo tais rótulos, convém inscrever as maiúsculas {A,B,D,C} que representam aqueles estados.

- Após o que, directamente a partir do diagrama de estados, se constrói a tabela ***Estado/Z***: esta desenrola-se por 2 colunas, rotuladas {0, 1} - isto é, tantas quantas as combinações possíveis que a *entrada* X pode assumir.

- na quadrícula de intersecção da linha **A** com a coluna **0**, inscrevem-se – sob a forma D/1 - características da seta que sai de **A** rotulada **0/**, a saber: que termina em **D** e que a saída é **1**.

- na quadrícula de intersecção da linha **A** com a coluna **1**, inscrevem-se – sob a forma B/0 - características da seta que sai de **A** rotulada **1/**, a saber: que termina em **B** e que a saída é **0**;

- e assim por diante, até esgotar todas as quadrículas – o mesmo é dizer: todas as *setas*…

- Em seguida, desdobra-se a tabela ***Estado/Z*** assim obtida em duas outras tabelas, **Estado** e **Z**:

- a tabela ***Estado*** contem só o Estado seguinte (na prática, quadrícula a quadrícula, retém-se só a informação *antes* do separador '/' – com uma alteração importante: converte-se a letra que representa o Estado na sua codificação (A→00, etc));

- a tabela **Z** contem a saída Z (na prática, quadrícula a quadrícula, retém-se só a informação *após* o separador '/')…

Seja um exemplo: na Tabela ***Estado/Z***, a quadrícula de intersecção da linha **A** com a coluna **0**, contém D/1; então, nas correspondentes quadrículas das tabelas ***Estado*** e ***Z*** há que inscrever, respectivamente, **11** (que é a codificação de D) e **1**…

Obtida a tabela ***Z***, e por aplicação do método de Karnaugh, é já possível escrever a expressão, acima, da saída Z em função de Q_1, Q_0 e X - deixando-se ao leitor a tarefa, trivial, de confirmar que, *LaPalisse dixit*, salvo erro, a solução está certa

- Relativamente às expressões relativas às entradas dos *flip-flops*, ainda é preciso pelo menos mais duas etapas – a primeira consistindo em desdobrar a tabela ***Estado*** em duas outras, concernindo respectivamente $\mathbf{Q_1}$ e $\mathbf{Q_0}$:
  - a tabela Q_1 contem só a *1ª* componente do Estado seguinte (quadrícula a quadrícula, aproveita-se o bit à *esquerda*);
  - a tabela Q_0 contem só a *2ª* componente do Estado seguinte (quadrícula a quadrícula, aproveita-se o bit à *direita*).

Seja um exemplo: na Tabela ***Estado***, a quadrícula de intersecção da linha **A** com a coluna **0**, contém **11**; então, nas correspondentes quadrículas das tabelas $\boldsymbol{Q_1}$ e $\boldsymbol{Q_0}$ há que inscrever, respectivamente, **1** e **1**…

- Após o que se constroem as tabelas finais, ditas de *excitação*, a saber: esclarecendo as entradas que deverão ser aplicadas aos *flip-flops* para lograr a transição pretendida (quadrícula a quadrícula, inscreve-se a resposta à pergunta: *quais devem ser as entradas deste flip-flop para que, partindo do estado actual, se chegue ao estado seguinte?*)

Para a solução mais simples, a de se usarem *flip-flops* D, acontece que $Q_{n+1}=D_n$; então, tais tabelas de *excitação*, designem-se de $\boldsymbol{D_1}$ e $\boldsymbol{D_0}$, acabam por se volver em *cópias* das tabelas de $\{Q_1,Q_0\}$; a partir delas, e por aplicação do método de Karnaugh, deduzem-se as expressões, acima, das entradas dos *flip-flops*…

Nota. Considere-se o uso de *flip-flops* JK. A última etapa seria mais demorada: as tabelas acima volver-se-iam em:

			Estado/Z	
			0	1
A	0	0	D/1	B/0
B	0	1	A/0	C/1
D	1	1	A/1	B/0
C	1	0	C/0	D/1
	Q_1	Q_0		

Estado	
0	1
11	01
00	10
00	01
10	11

Z	
0	1
1	0
0	1
1	0
0	1

Q_1	
0	1
1	0
1	0
0	0
1	1

Q_0	
0	1
1	1
0	0
0	1
0	1

JK_1	
0	1
1x	0x
0x	1x
x1	x1
x0	x0

JK_0	
0	1
1x	1x
x1	x1
x1	x0
0x	1x

J_1	
0	1
1	0
0	1
x	x
x	x

K_1	
0	1
x	x
x	x
1	1
0	0

J_0	
0	1
1	1
x	x
x	x
0	1

K_0		
0	1	
x	x	X
x	x	
1	1	
x	x	
1	0	
x	x	

Como se chegou aqui? Resposta: varrem-se as quadrículas das tabelas de $\{Q_1, Q_0\}$, e, para cada uma, responde-se à questão elementar: *quais as entradas JK para que, partindo do estado actual, se chegue ao estado seguinte?*

Considere-se, nomeadamente, a tabela de $\boldsymbol{Q_1}$ – o desafio sendo a construção de, por exemplo, a tabela $\boldsymbol{JK_1}$:

- para a primeira quadrícula (linha A – em que Q_1=**0** - e coluna X=0), é visível que o estado seguinte deve ser Q_1=**1**. Pergunta: *quais as entradas JK para que, partindo do estado '**0**', se chegue a '**1**'?* Resposta, *vidé* Preâmbulo P2: deve ser J=**1**, K=**x** (qualquer). Então, inscreve-se, na correspondente quadrícula da tabela $\boldsymbol{JK_1}$ o par **1x**;
- para a quadrícula abaixo (linha B – em que Q_1=**0** - e coluna X=0), é visível que o estado seguinte deve ser Q_1=**0**. Pergunta: *quais as entradas JK para que, partindo do estado '**0**', se permaneça em '**0**'?* Resposta: deve ser J=**0**, K=**x** (qualquer). Então, inscreve-se, na correspondente quadrícula da tabela $\boldsymbol{JK_1}$ o par **0x**;
- e assim por diante, até esgotar todas as quadrículas (de $\boldsymbol{Q_1}$ e $\boldsymbol{Q_0}$) – o mesmo é dizer: todas as *transições de estado*…

Em seguida, desdobra-se as tabelas de *excitação* $\boldsymbol{JK_1}$ *e* $\boldsymbol{JK_0}$ assim obtidas em dois pares de tabelas, $\{\mathbf{J_1}, \mathbf{K_1}\}$ e $\{\mathbf{J_0}, \mathbf{K_0}\}$.

- a tabela $\mathbf{J_1}$ contem apenas a informação à esquerda da correspondente quadrícula da tabela $\boldsymbol{JK_1}$;
- a tabela $\mathbf{K_1}$ contem apenas a informação à direita da correspondente quadrícula da tabela $\boldsymbol{JK_1}$;
- idem para as tabelas $\mathbf{J_0}$, $\mathbf{K_0}$**…**

Seja um exemplo: na Tabela $\boldsymbol{JK_1}$, a quadrícula de intersecção da linha **A** com a coluna **0**, contém **1x**; então, nas correspondentes quadrículas das tabelas $\boldsymbol{J_1}$ e $\boldsymbol{K_1}$ há que inscrever, respectivamente, **1** e **x**…

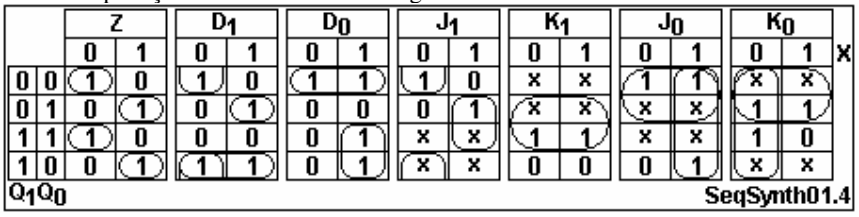
A partir destas 4 tabelas, e por aplicação do método de Karnaugh, deduzem-se as expressões das entradas dos *flip-flops JK*:

$$J_1 = \overline{X}\,\overline{Q_0} + X\,Q_0,\ K_1 = Q_0,\ J_0 = \overline{Q_1} + X \text{ e } K_0 = \overline{Q_1} + \overline{X},$$

a que corresponde o circuito SeqSynth01.12.

(SeqSynth01.12, e aliás SeqSynth01.11, incluem uma entrada, INIT_L, que, quando a *Low*, inicia os *flip-flops* a {0 0})

Adiante, esquematiza-se a aplicação do método de Karnaugh aos vários casos abordados:

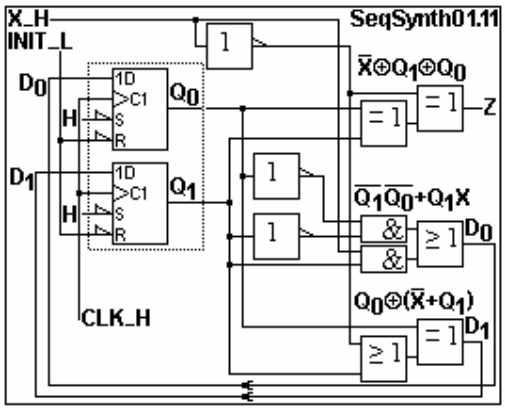


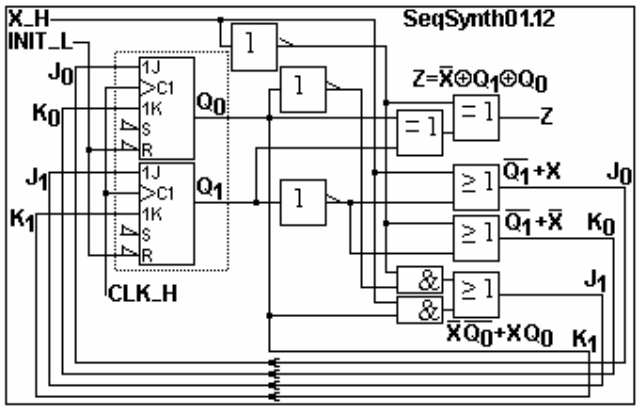
Observação a quem acha que "são tabelas a mais":

- obviamente, não é absolutamente imperioso transitar pela tabela ***Estado/Z***: com algum treino, o leitor pode escrever as tabelas ***Estado*** (ou mesmo $\mathbf{Q_1}$ e $\mathbf{Q_0}$) e ***Z***, *directamente* a partir do diagrama de Estados fornecido; se for o caso, quiçá seja boa ajuda inscrever, nas elipses que representam os estados no diagrama, os pares de bits que os codificam....

- também obviamente, não é absolutamente imperioso transitar pela tabelas $\boldsymbol{JK_1}$ ***e*** $\boldsymbol{JK_0}$: com algum treino, o leitor pode escrever as tabelas $\{\mathbf{J_1, K_1}\}$ e $\{\mathbf{J_0, K_0}\}$, *directamente* a partir das tabelas de $\mathbf{Q_1}$ e $\mathbf{Q_0}$....

De cronómetro na mão, o leitor poderá verificar por si próprio quantos segundos poupa se o fizer…





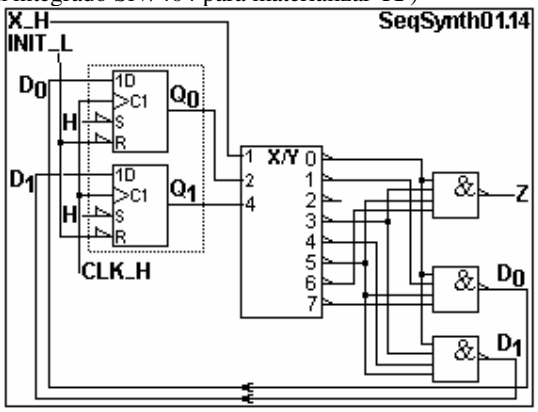
Nota a propósito:

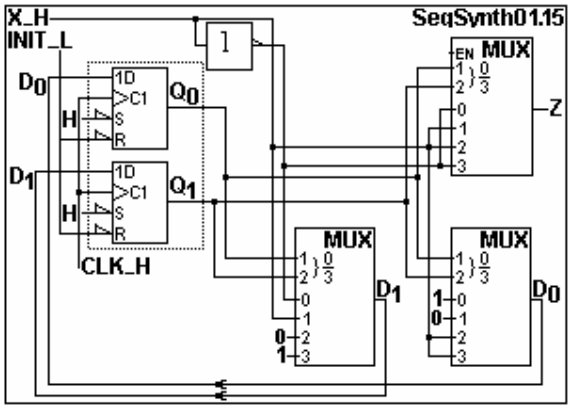
O enunciado remete explicitamente para as "equações da entrada dos *Flip-Flops* e de Z". *A ter-se omitido* essa condicionante (como é, genericamente, o caso de Trabalhos de Laboratório, em que a liberdade de concepção é maior), poder-se-iam ter congeminado outros circuitos. Adiante, apresentam-se dois deles:

- em SeqSynth01.14, as funções Z, D1 e D0 são produzidas através de um *decoder* 3:8 - com saídas activas a *Low* - e NANDs; nas entradas de selecção do *decoder* aplicam-se $\{\mathbf{Q_1Q_0X}\}$ – *ordenadas* $\mathbf{X}\rightarrow$'1', $\mathbf{Q_0}\rightarrow$'2', $\mathbf{Q_1}\rightarrow$'4'. Por ex., pelas tabelas acima, Z deve ser '**1**' quando, e só quando, $\{Q_1$**=0** ***e*** Q_0**=0** ***e*** X**=0**$\}$ ***ou*** $\{Q_1$**=0** ***e*** Q_0**=1** ***e*** X**=1**$\}$ ***ou*** $\{Q_1$**=1** ***e*** Q_0**=0** ***e*** X**=1**$\}$ ***ou*** $\{Q_1$**=1** ***e*** Q_0**=1** ***e*** X**=0**$\}$; então, Z será a saída de uma NAND cujas entradas são as saídas '0', '3', '5' e '6' do *decoder*.

- em SeqSynth01.15, as funções Z, D1 e D0 são produzidas através de *multiplexers* 4:1; nas entradas de selecção aplicam-se $\{\mathbf{Q_1Q_0}\}$ – *ordenadas* $\mathbf{Q_0}\rightarrow$'1', $\mathbf{Q_1}\rightarrow$'2'. Por ex., pelas tabelas acima, e para as quatro combinações possíveis dessas entradas, $\{Q_1$**=0** ***e*** Q_0**=0**$\}$, $\{Q_1$**=0** ***e*** Q_0**=1**$\}$, $\{Q_1$**=1** ***e*** Q_0**=0**$\}$ e $\{Q_1$**=1** ***e*** Q_0**=1**$\}$, Z deve ser, respectivamente, $\overline{X}$, X, X e $\overline{X}$; então, aplicando estes valores às entradas de dados de um multiplexer, obtém-se, na saída, Z.

(Na prática, recorre-se a *dois* integrados SN74153 – com, cada um, dois *multiplexers* 4:1, suficiente para produzir $\{\mathbf{D_1D_0Z}\}$ – e a um integrado SN7404 para materializar $\overline{X}$)





O Diagrama temporal à lupa…

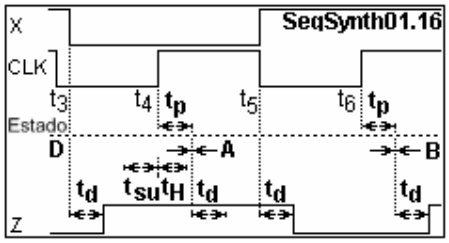
Quiçá seja oportuna uma revisão ao diagrama SeqSynth01.3, para uma percepção mais exacta do que está sucedendo…

Inicialmente, a entrada é X**=0**, e o circuito encontra-se no estado A, isto é: as saídas dos *flip-flops* são $\{Q_1$**=0**, Q_0**=0**$\}$. Essa entrada e essas saídas aplicam-se à malha combinatória (à direita dos *flip-flops*) em SeqSynth01.11 - produzindo um conjunto de valores (D_1D_0), nas entradas dos *flip-flops*… Em t_1, ocorre o flanco ascendente do impulso de relógio; isso provoca a

mudança do estado dos *flip-flops*: passa a ser D, isto é, {Q_1=**1**, Q_0=**1**}; *tal não acontece instantaneamente*: demora um tempo, t_pLH ou t_pLH (tempo de passar de *Low* a *High* ou de *High* a *Low*).

Encontrando-se o circuito no estado 'D', X sofre mudanças: passa de '**0**' a '**1**' (em t_2) e depois retorna a '**0**' (em t_3) – mas isso não se repercute nos *flip-flops*: eles só vêm a mudar aquando de um novo flanco ascendente do relógio – o que sucede em t_4. De acordo com o diagrama de estados, isso provoca a mudança de estado dos *flip-flops*: para 'A' – e por aqui nada de novo.

É entretanto hora de referir que, para um bom funcionamento, as entradas dos *flip-flops* têm que permanecer *estáveis* ao menos um tempo t_{su} (*Setup-time*) *antes* de tal flanco ocorrer; e têm que continuar estáveis mais um tempo t_H (*Hold-time*) *depois* desse flanco! SeqSynth01.16 esquematiza a cronologia dos eventos:

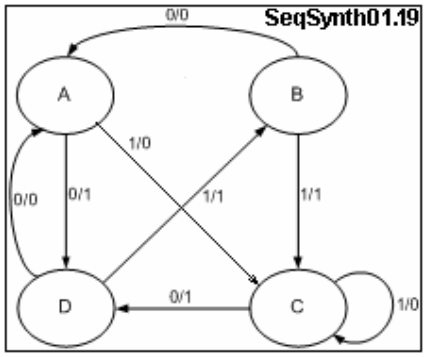


- em t_3, a entrada X passa a '**0**'; a malha combinatória reage a essa mudança: após um certo tempo, seja t_d, produz novos valores para D_1D_0 (e Z);
- desde que decorra t_{su} sem que D_1D_0 se tenham alterado, pode o sinal de relógio experimentar um flanco ascendente, em t_4;
- as entradas dos *flip-flops* não se devem alterar t_H após esse flanco;
- algum tempo t_p (t_pLH ou t_pHL) após esse flanco, os *flip-flops* mudam de estado… e t_d depois há novos valores nas entradas dos *flip-flops*…
- no instante t_5, a entrada X passa a '**1**';
- a malha combinatória reage a essa mudança: após um certo tempo, seja t_d, produz novos valores para D_1D_0 (e Z)…

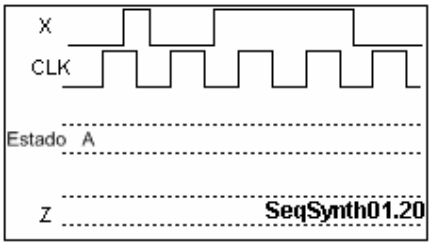
Resumindo e concluindo:

- as mudanças de estado acontecem (se acontecerem) t_p (t_pLH ou t_pHL) *após os flancos* ascendentes do relógio;
- podem acontecer mudanças nas saídas $t_p + t_d$ após os flancos de relógio;
- ademais, numa máquina de *Mealy*, podem acontecer mudanças nas saídas t_d após as mudanças na entrada.

6. [10E3.8] Considere o diagrama de estados em SeqSynth01.19.
   6. 1. Complete o diagrama temporal em SeqSynth01.20, indicando o estado do circuito e o valor da saída Z em cada instante. Considere que inicialmente o circuito se encontra no estado A, que Z=1 e que os FF a utilizar reagem ao flanco ascendente do relógio
   6. 2. Utilizando a síntese clássica de circuitos sequenciais síncronos, obtenha as equações de entrada dos Flip-flops e de Z, de um circuito que implemente a máquina de estados apresentada. Codifique os estados da seguinte forma (A=00, B=01, C=11, D=10).

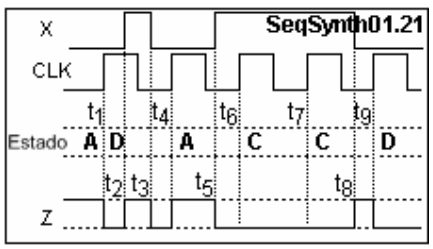


R1: SeqSynth01.21 mostra a evolução de {Estado, Z}



Processo mental, com a ajuda da tabela adiante, construída de modo análogo ao seguido em [10E2.9]:

- antes de mais, convém assinalar, no diagrama temporal, os *instantes significativos* (*vidé verticais a ponteado*):
  - relativamente à *mudança de estado dos flip-flops*, os instantes significativos são os do *flanco ascendente do relógio* {t_1, t_4, t_6, t_7, t_9};
  - já quanto à *mudança na saída* (e pois se trata de uma máquina de *Mealy*) os instantes significativos são esses e, *também*, aqueles em que *a entrada de dados, X, se altera* (isto é, e no total: t_1, ..., t_9);
- a segunda linha da tabela assinala os valores de X *entre esses instantes*;
- na linha seguinte inscrevem-se os estados do circuito, *após* o *flanco ascendente do relógio* (t_1, t_4, t_6, t_7, t_9). Por ex., em t_1 o estado é o inicial, **A**, e entrada é **X=0**; o diagrama de estados exibe uma seta saindo de **A** com etiqueta '**0**' dirigida para **D** – então, o estado seguinte é **D**, e é isso que se inscreve abaixo de t_2 (aí permanecendo até ao flanco seguinte, t_4);
- na linha seguinte inscreve-se a saída Z, *entre* os instantes significativos. Por ex., antes de t_1 o estado é o inicial, **A**, e a entrada é **X=0**); a seta que sai de **A** com etiqueta '**0**' denota uma saída Z=1 – então, é '**1**' o que se inscreve abaixo de t_1 (e que significa: a saída do circuito *até* t_1);



	CLK: t_1	t_2	t_3	CLK:t_4	t_5	CLK:t_6	CLK: t_7	t_8	CLK:t_9	
Entrada X:	0	0	1	0	0	1	1	1	0	0
Estado	**A**	**D**			**A**		**C**	**C**		**D**
Saída Z:	1	0	1	0	1	0	0	0	1	0

R2: Convertendo o diagrama de estados em tabelas de estados/saída, já sob a forma "Karnaugh":

			Estado/Z		Estado		Z		Q_1, D_1		Q_0, D_0		
			0	1	0	1	0	1	0	1	0	1	X
A	0	0	D/1	C/0	10	11	1	0	1	1	0	1	
B	0	1	A/0	C/1	00	11	0	1	0	1	0	1	
C	1	1	D/1	C/0	10	11	1	0	1	1	0	1	
D	1	0	A/0	B/1	00	01	0	1	0	0	0	1	
	Q_1	Q_0											

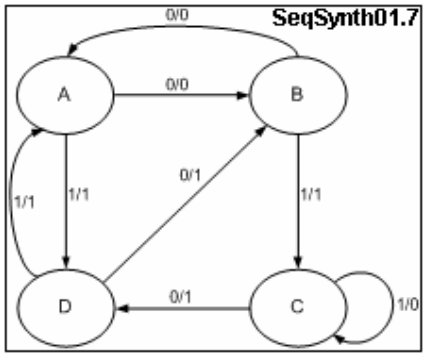
Deduz-se $Z = \overline{X}(\overline{Q_1}\ \overline{Q_0} + Q_1 Q_0) + X(\overline{Q_1} Q_0 + Q_1 \overline{Q_0}) = \overline{X} \oplus Q_1 \oplus Q_0$

e, considerando o uso de *flip-flops* D, as equações de entrada dos *flip-flops* vêm a ser:

$$D_1 = \overline{Q_1}\,\overline{Q_0} + Q_1 Q_0 + X Q_0 = \overline{Q_1 \oplus Q_0} + X Q_0 \ e\ D_0 = X$$

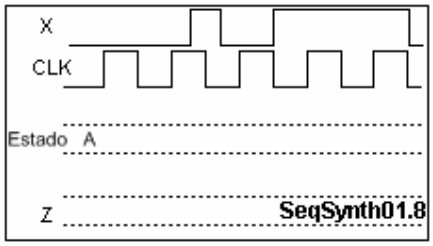
Síntese "1 *flip-flop* por Estado"

7. [10T2.5,10E1.8] Considere o diagrama de estados SeqSynth01.7.



7. 1. Trata-se de uma máquina de Mealy ou de Moore?

7. 2. Complete o diagrama temporal SeqSynth01.8, indicando o estado do circuito e a saída Z em cada instante. Considere que inicialmente o circuito se encontra no estado A, que Z = 0 e que os FF a usar reagem ao flanco ascendente do relógio:



7. 3. Encontre as equações de activação de entrada de cada um dos *Flip-flops* e a equação da saída quando se utiliza uma codificação "*1-hot-encoding*", ou 1 FF por estado (simplifique as equações encontradas).

R1: Trata-se duma máquina de Mealy, *vidé* Preâmbulo P4: a saída depende do estado, mas também da entrada. Por ex., em A, a saída será 0 se a entrada for '0', mas será '1' se a entrada for '1': *se, em A, a entrada variar, a saída também varia*…

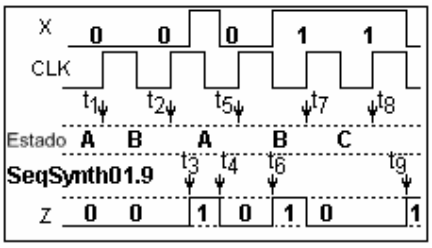
R2: SeqSynth01.9 mostra a evolução de {Estado, Z}

Processo mental: convirá, antes de desenhar o diagrama, traçar uma *tabela da evolução*…

(A tabela é construída de modo análogo ao seguido em [10E2.9])

	CLK: t_1	CLK: t_2	t_3	t_4	CLK: t_5	t_6	CLK: t_7	CLK: t_8	t_9	
Entrada X:	0	0	0	1	0	0	1	1	1	0
Estado	A	B	A			B		C		
Saída Z:	0	0	0	1	0	0	1	0	0	1

R3: Eis *dois* métodos para cavalgar o problema:



O método mais moroso: Primeiro, converte-se o diagrama de estados na tabela de estados/saída, *vidé* adiante… O diagrama envolve tão somente 4 estados; a *síntese 1-ff/estado* remete então para 4 *flip-flops*, designem-se de {A, B, C, D}. A primeira etapa é construir uma *tabela com outras tantas linhas* - rotuladas com as maiúsculas que representam aqueles estados; na tabela, incluíram-se também as correspondentes combinações daqueles *flips-flops*…

Após o que, directamente a partir do diagrama de estados, se constrói a tabela ***Estado/Z***: esta desenrola-se por 2 colunas, {0, 1} - isto é, tantas quantas as combinações possíveis que X pode assumir.

- na quadrícula de intersecção da linha **A** com a coluna **0**, inscrevem-se – sob a forma B/0 - características da seta que sai de **A** rotulada **0/**, a saber: que termina em **B** e que a saída é **0**;

- na quadrícula de intersecção da linha **A** com a coluna **1**, inscrevem-se – sob a forma D/1 - características da seta que sai de **A** rotulada **1/**, a saber: que termina em **D** e que a saída é **1**;

- e assim por diante, até esgotar todas as quadrículas – o mesmo é dizer: todas as *setas*…

Enfim, e similarmente a [10E2.9], desdobra-se a tabela ***Estado/Z*** assim obtida em duas outras tabelas, **Estado** e **Z**:

Flip-Flops					**Estado/Z**	
					0	1
A	0	0	0	1	B/0	D/1
B	0	0	1	0	A/0	C/1
C	0	1	0	0	D/1	C/0
D	1	0	0	0	B/1	A/1
	D	**C**	**B**	**A**		

⇒

Estado	
0	1
B	D
A	C
D	C
B	**A**

Z	
0	1
0	1
0	1
1	0
1	1

⇒

D	
0	1
0	1
0	0
1	0
0	0

C	
0	1
0	0
0	1
0	1
0	0

B	
0	1
1	0
0	0
0	0
1	0

A	
0	1
0	0
1	0
0	0
0	**1**

X

A_{antes}='**1**', os outros a '**0**'
B_{antes}='**1**', os outros a '**0**'
C_{antes}='**1**', os outros a '**0**'
D_{antes}='**1**', os outros a '**0**'

Parêntesis inadiável: salta à vista que a filosofia com que agora se identificam *estados*, *flip-flops* (e respectivas *entradas*) difere da seguida em [10E2.9]. Então, usaram-se os identificadores {A, B, C e D} (para representar *estados*), {Q_1, Q_0} (para representar *flip-flops*) e {D_1 D_0} ou {JK_1 JK_0} (para representar as suas *entradas*). Mas agora o *mesmo* identificador, seja A, é usado para denotar um *estado* (*vidé* maiúsculas que rotulam as linhas) ou um *flip-flop* (*vidé* rótulos da 2ª-a-5ª colunas à esquerda e rótulos das tabelas à direita da tabela **Z**); de facto, e porquanto se usam *flip-flops* D, A até será usado para denotar a

entrada do *flip-flop* A… (*vidé* rótulos das tabelas à direita da tabela **Z**). Isso faz-se, é claro, por comodidade – crendo que ao leitor não será difícil inferir, pelo contexto, a acepção com que, a dado momento, se está usando a maiúscula A…

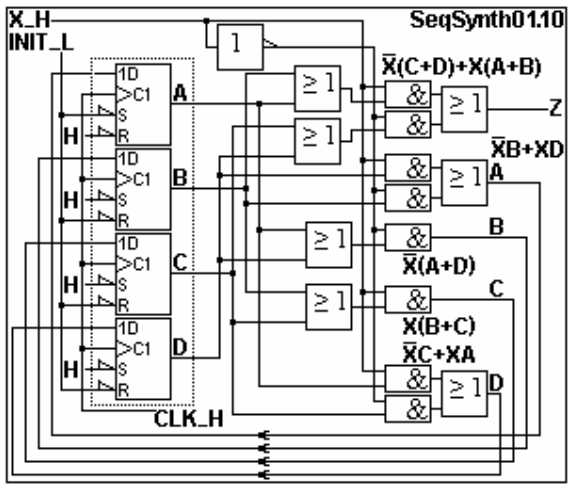
A etapa seguinte é desdobrar a a tabela ***Estado*** em quatro tabelas, {A, B, C, D}.

- a tabela de **A** é toda preenchida a '**0**'s – excepto nas quadrículas em que a tabela ***Estado*** assume o valor **A**; concretamente, nesta última, a maiúscula A ocorre, apenas, nas quadrículas de intersecção da linha-B/coluna-0 e linha-D/coluna-1; então, e apenas nestas quadrículas, a tabela A volve-se em '**1**' – e nas demais fica sendo '**0**'.

- as tabelas de {B, C e D} constroem-se de modo análogo…

Observação a quem acha que "são tabelas a mais": obviamente, não é absolutamente imperioso transitar pela tabela ***Estado/Z***, nem sequer pela tabela ***Estado***… É pacífico que a tabela **A** é a tabela de verdade da função de activação do *flip-flop* A: especifica em que circunstâncias é que ele transita para o estado '**1**'. Ora, verifica-se que só existem duas *setas dirigidas para A*: uma, rotulada **0**/0 partindo de B, e outra rotulada **1**/1 partindo de **D**. Então, pode-se construir a tabela **A** *directamente a partir do diagrama*: nas quadrículas linha-B/coluna-0 e linha-D/coluna-1 inscreve-se '1', as demais ficam a '**0**'

(Atenção: a praxis comum é *escrever uma tabela de verdade em uma coluna*, ficando as variáveis à *esquerda*, cada qual na sua coluna… Não é isso que agora acontece: as tabelas acima seguem *outra ordenação*! As variáveis são {A, B, C, D, X} – sendo que {A, B, C, D} se dispõem à esquerda, mas {X} dispõe-se *em cima*…)



A modos de apoio à tranquilidade após preencher as tabelas, convém atestar se as tabelas {A, B, C e D} exibem as seguintes propriedades: numa quadrícula em que A (por ex.) é '**1**', nenhuma tabela mais assume também o valor '**1**' (isso, é claro, reflecte *a síntese 1-ff/estado: em cada momento, só um dos flip-flops pode estar a '1'*); por outro lado, o número total de '**1**'s semeado numa linha daquelas tabelas é igual ao número de combinações possíveis das entradas (neste caso, 2).

A partir das tabelas, e pressuposto o uso de *flip-flops* D, são imediatas as expressões algébricas:

$$A = \overline{X}\,B + X\,D\,,\; B = \overline{X}\,(A+D),\; C = X\,(B+C) \text{ e } D = \overline{X}C + X\,A\,.\,Z = \overline{X}\,(C+D) + X(A+B)$$

Nota 1: É claro que $C = X\,(B+C)$ *não é uma equação a uma incógnita*, C ! O que a expressão faz é exprimir a *entrada* do *flip-flop* C em termos de X e das *saídas* dos *flip-flops*. Nos termos da filosofia de identificação usada em [10E2.9], ter-se-ia antes escrito algo como $D_C = X(Q_B + Q_C)$ - mas o leitor certamente concordará que esta expressão é *menos cómoda*…

Nota 2: a escrita *imediata* das expressões acima poderá parecer uma passada comprida demais – pelo que *é mister* dedicar-lhe algum tempo… É que, de facto, da tabela de A, parece que se deveria concluir $A = \overline{X}\,\overline{D}\,\overline{C}\,B\,\overline{A} + X\,D\overline{C}\,\overline{B}\,\overline{A}$...

Para compreender o que está acontecendo, faça-se uma viagem ao passado: à boa maneira de Karnaugh, como se deduziria a expressão de A? Bem entendido, haveria que traçar um quadro em que as combinações {0,1} das variáveis {A,B,C,D} seriam dispostas segundo o código binário reflectido, etc. Eis como ficaria:

		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	D
	A	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	C
		0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	B
		0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	A
0		x	0	X	**1**	x	x	x	0	x	x	x	x	x	x	x	0	
1		x	0	X	0	x	x	x	0	x	x	x	x	x	x	x	**1**	
X																		

Repare-se: conquanto as cinco variáveis {A,B,C,D,**X**} se volvam em 32 combinações possíveis de '**0**'s e '**1**'s, na sua *esmagadora maioria o quadro de A volve-se em indiferenças, 'x'*! Isso reflecte o peculiar da *síntese 1-ff/estado*: quando um *flip-flop* se encontra no estado '**1**', os outros estão a '**0**'! Na vida real de "*1-ff/estado*", nunca acontece estarem ao mesmo tempo dois ou mais *flip-flops* a '**1**' (nem tampouco todos a '**0**') – e daí as *indiferenças* registadas no quadro. Isso habilita a grandes simplificações – cabendo ao leitor conferir que as expressões que obtém são precisamente aquelas acima.Uma outra linha de pensamento passa por analisar os membros da expressão $\overline{X}\,\overline{D}\,\overline{C}\,B\,\overline{A} + X\,D\overline{C}\,\overline{B}\,\overline{A}$… Seja o caso do produto $\overline{D}\,\overline{C}\,B\,\overline{A}$, que faz parte do primeiro deles, - e veja-se qual o valor que assume quando B=0 e quando B=1. Vem:

- para B=0, vem $\overline{D}\,\overline{C}\,B\,\overline{A} = 0$

- para B=1 – e porque então *necessariamente sera* A=C=D=0 – vem $\overline{D}\,\overline{C}\,B\,\overline{A}=1$

Isto é, $\overline{D}\,\overline{C}\,B\,\overline{A}$ volve-se, *sempre*, no valor de B: $\overline{D}\,\overline{C}\,B\,\overline{A}=B$! Quanto a $D\,\overline{C}\,\overline{B}\,\overline{A}$, volve-se, *sempre*, no valor de D: $D\,\overline{C}\,\overline{B}\,\overline{A}=D$! Ou seja, $\overline{X}\,\overline{D}\,\overline{C}\,B\,\overline{A}+X\,D\overline{C}\,\overline{B}\,\overline{A}=\overline{X}\,B+X\,D$: *os complementos dos estados evaporaram-se!*

O método mais rápido:

A *lei do menor esforço* manda que se conclua com um método (de chegar às expressões para {A, B, C, D}) que evite o *bué-chatérrimo* preenchimento (com '**0**'s e '**1**'s) das tabelas {A, B, C, D}. Trata-se apenas de preencher uma *tabela indicando as circunstâncias necessárias para se ir ter a algum estado particular*: estado a estado, consideram-se *as setas que para ele convergem* – e regista-se: de que estado provêm, e qual a entrada a que correspondem:

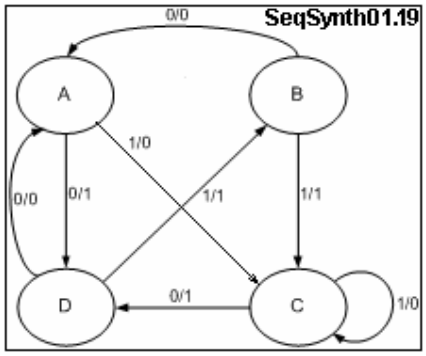
Para ingressar no Estado…	é preciso (Estado actual ***e*** entrada):				
A	{B ***e*** X=0)	***ou***	{D ***e*** X=1}	⇒	$A=\overline{X}\,B+X\,D$
B	{A ***e*** X=0)	***ou***	{D ***e*** X=0)		$B=\overline{X}\,(A+D)$
C	{B ***e*** X=1}	***ou***	{C ***e*** X=1}		$C=X\,(B+C)$
D	{A ***e*** X=1}	***ou***	{C ***e*** X=0)		$D=\overline{X}C+X\,A$

Depois, é aplicar a velha substituição: '***ou***' por OR e '***e***' por AND – e já está!

O circuito correspondente encontra-se esquematizado em SeqSynth01.10 Como já sucedia com SeqSynth01.11, ele estrutura-se em *dois blocos*: um à esquerda, composto por apenas quatro *flip-flops*-D, e um à direita, que é uma malha combinatória – com a função de gerar a saída do circuito e as entradas dos *flip-flops*…

O circuito inclui obrigatoriamente uma entrada, INIT_L, que, quando a *Low*, coloca os *flip-flops* no estado {0 0 0 1}.

8. [10E4.8] Considere o diagrama de estados em SeqSynth01.19. Obtenha as equações simplificadas da entrada dos *flip-flops* e da saída Z de um circuito que implemente a máquina de estados utilizando a codificação "*One-Hot*", um FF por estado.



R: Preâmbulo: Usando a codificação 1FF/estado, e espraiando-se o diagrama por quatro estados, o circuito comportará 4 *flip-flops* – tais que, em cada momento, *apenas um deles* está a '**1**'! Convém designá-los pelos mesmos nomes dos estados: {A, B, C , D} – e de tal modo que dizer que o circuito está no estado '**A**' (por ex.) seja equivalente a dizer que é o *flip-flop* '**A**' que está a '**1**'!

A saída Z é '**1**' apenas para a seguinte combinação de estados/saída:

Estado actual:	A	B	C	D
Entrada X:	0	1	0	1

Para bom entendedor, a primeira coluna da tabela (por ex.) quer dizer o seguinte: conforme ao preâmbulo acima, quando o *flip-flop* '**A**' estiver a '**1**' e a entrada X for '**0**', a saída do circuito é '**1**'; idem para as demais.

De que se deduz imediatamente: $Z=\overline{X}(A+C)+X(B+D)$

Quanto às equações de entrada dos *flip-flops* (também elas designadas de {A, B, C, D}), convém preencher uma *tabela indicando as circunstâncias necessárias para se ir ter a algum estado particular*: estado a estado, consideram-se *as setas que para ele convergem* – e regista-se: de que estado provêm, e qual a entrada a que correspondem:

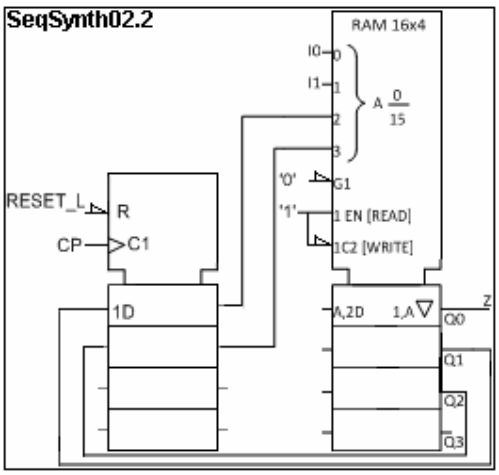
Depois, é aplicar a velha substituição: '***ou***' por OR e '***e***' por AND – e já está!

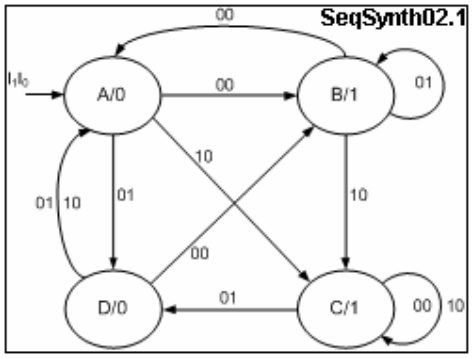
Para ingressar no Estado…	é preciso (Estado actual ***e*** entrada):						
A	{B ***e*** X=0)	***ou***	{D ***e*** X=0}			⇒	$A=\overline{X}\,B+\overline{X}\,D$
B	{D ***e*** X=1)						$B=X\,D$
C	{A ***e*** X=1}	***ou***	{B ***e*** X=1}	***ou***	{C ***e*** X=1}		$C=X\,(A+B+C)$
D	{A ***e*** X=0}	***ou***	{C ***e*** X=0)				$D=\overline{X}(C+A)$

Síntese por Memórias Integradas

9. [10T2.6] Considere o diagrama de estados SeqSynth02.1 e o circuito representado em SeqSynth02.2. O circuito tem 2 entradas I_1 e I_0, e uma saída Z. Preencha a tabela que representa o conteúdo da memória para que o circuito implemente o diagrama de estados indicado. Deixe em branco as posições de memória não utilizadas.

Address	Q3	Q2	Q1	Q0
0000				
0001				
0010				
0011				
0100				
0101				
0110				
0111				
1000				
1001				
1010				
1011				
1100				
1101				
1110				
1111				





R: A tabela preenchida encontra-se lá mais para a frente…

O método mais moroso: A primeira etapa é a costumeira conversão do diagrama de estados numa tabela de estados/saída:

- Antes de mais, há que reparar que o diagrama envolve 4 estados; pois que 2^2=4, podem codificar-se em palavras de 2-bit – designem-se eles de $\{\mathbf{D_1D_0}\}$. Há então que construir uma *tabela com 4 linhas* - isto é, tantas quantas as combinações possíveis que esses *di-bits* podem assumir; antevendo a posterior síntese por RAM, convém rotulá-las com o *código binário natural*: {00, 01, 10, 11}; a anteceder tais rótulos, inscrevem-se as maiúsculas {A, B, C, D} que representam aqueles estados.

- Após o que, directamente a partir do diagrama de estados, se constrói a tabela ***Estado(Seguinte)***: esta desenrola-se por 4 colunas, rotuladas {00, 01, 10, 11} - isto é, tantas quantas as combinações possíveis que as *entradas* $\{I_1\ I_0\}$ podem assumir.

- na quadrícula de intersecção da linha **A** com a coluna **00**, inscreve-se – sob a forma B – o facto de a seta que sai de **A** rotulada **00** terminar em **B**;

- na quadrícula de intersecção da linha **A** com a coluna **01**, inscreve-se – sob a forma D – o facto de a seta que sai de **A** rotulada **01** terminar em **D**;

- e assim por diante, até esgotar todas as quadrículas – o mesmo é dizer: todas as *setas*…

Estados	Circuito de Controlo		**Estado seguinte**			
			0	0	1	1
			0	1	0	1
A	0	0	**B**	**D**	C	x
B	0	1	A	B	C	x
C	1	0	C	D	C	x
D	1	1	B	A	A	x
			D_1	**D_0**		

⇒

$\{D_1\ D_0\}$ seguinte							
0		0		1		1	
0		1		0		1	
0	**1**	**1**	**1**	1	0	x	x
0	0	0	1	1	0	x	x
1	1	1	1	1	0	x	x
0	1	0	0	0	0	x	x
D_1	**D_0**	**D_1**	**D_0**	**D_1**	**D_0**	**D_1**	**D_0**

Z
0
1
1
0

D_1					**D_0**				
0	0	1	1		0	0	1	1	**I_1**
0	1	0	1		0	1	0	1	**I_0**
0	**1**	1	x		**1**	**1**	0	x	
0	0	1	x		0	1	0	x	
1	1	1	x		1	1	0	x	
0	0	0	x		1	0	0	x	

Em seguida, transforma-se a tabela ***Estado*** numa outra tabela, $\mathbf{\{D_1D_0\}}$ – no sentido de que a maiúscula que representa cada estado é convertida na sua codificação (A→00, etc)). Seja um ex.: na tabela ***Estado***, a quadrícula de intersecção da linha **A** com a coluna **00** contém B; então, na correspondente quadrícula da tabela $\mathbf{\{D_1D_0\}}$ inscreve-se **01** (que é a codificação de B).

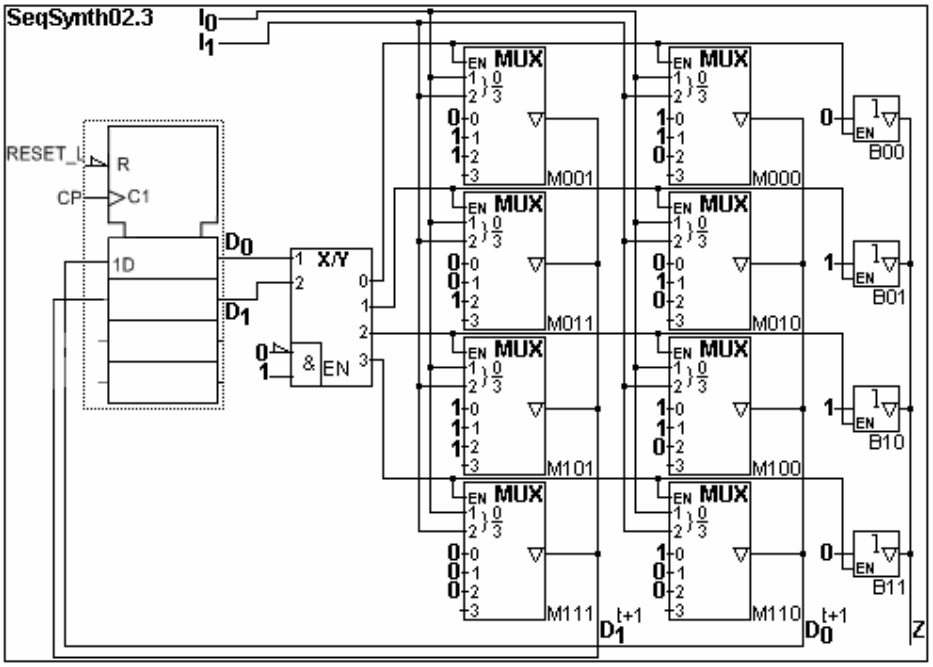
Além disso, constrói-se também a tabela **Z**. No caso, o facto de a saída não estar às cavalitas das setas, antes surge no seio das elipses que representam os estados, leva a que ela se reduz a uma única coluna: *a saída depende exclusivamente do estado* – que não das entradas $\{I_1$ e $I_0\}$. Por ex., quando o circuito se encontra no estado A, o diagrama informa que a saída é '**0**' – e é isso que se inscreve na quadrícula de intersecção da linha **A** com a coluna **Z**.

Enfim, desdobra-se a tabela **{D_1D_0}** em duas outras, concernindo respectivamente **D_1** e **D_0**:

- a tabela **D_1** contem só a *1ª* componente de **{D_1D_0}** (quadrícula a quadrícula, aproveita-se o bit à *esquerda*);
- a tabela **D_0** contem só a *2ª* componente de **{D_1D_0}** (quadrícula a quadrícula, aproveita-se o bit à *direita*).

Seja um exemplo: na Tabela **{D_1D_0}**, a quadrícula de intersecção da linha **A** com a coluna **00**, contém **01**; então, nas correspondentes quadrículas das tabelas **D_1** e **D_0** há que inscrever, respectivamente, **0** e **1**…

Nesta altura do percurso, seria pacífico deduzir as expressões de activação das entradas e das saídas dos *flip-flops* {**D_1D_0**} – se, é claro, se houvera requerido materializar o circuito em *flip-flops*… Mas de facto o que se requere é suportá-lo numa RAM! Isso convida a um *parêntesis* preliminar…



Revendo os logigramas SeqSynth01.11/SeqSynth01.12 e SeqSynth01.10, salta à vista uma *mesma estrutura*: um bloco à esquerda, com apenas *flip-flops*, e um bloco à direita, que é uma malha combinatória trivial – de NOTs, ANDs, ORs, etc… E, para um engenheiro, é forçosa a pergunta: como progredir - no sentido de uma maior compactação/simplificação do circuito?

- quanto ao bloco de *flip-flops*, a resposta adivinha-se: substituir os *dois flip-flops* (que seriam precisos na síntese clássica) por *um registo* de ao menos 2-bit… De facto, o circuito dado, SeqSynth02.2, comporta, do lado esquerdo, um registo de 4-bit, e é quanto basta: designando o bit mais a norte de **D_0**, e o que lhe está por baixo de **D_1**, o *par* de *flip-flops* é substituído por *um* registo, *vidé* SeqSynth02.3;

- e, quanto à malha combinatória, também se adivinha o sentido das pesquisas para a simplificar: recorrer a *blocos* como sejam *multiplexers* e *decoders*... É isso que se verá a seguir…

A materialização de uma função booleana por *multiplexers* e *decoders* suporta-se no conhecimento da sua tabela de verdade. Ora, e centrando a atenção por ora em **D_0**, um ponto de vista sobre a tabela acima é entender **D_0** como uma função de {**I_1 I_0**} que se espraia por *quatro tabelas de verdade distintas*, uma por cada estado; por ex., no estado A, a tabela de verdade é '**110x**' (=Σ m(0,1)); por outras palavras, a modos que **D_0** acaba por ser um OR de várias tabelas de verdade - perdão, de *multiplexers* -, em que, em cada momento está activo um deles apenas. E, se a saída de tais multiplexers for *tri-state*, então nem o OR é preciso, *vidé* SynthSeq02.3:

- **D_1D_0** aplicam-se às entradas de selecção {1,2} de um *decoder* 2:4 – *ordenadas* **D_0**→'1', **D_1**→'2' (i.e.: 'bit de *menor* peso' → 'entrada de selecção de *menor* peso'); então, quando **D_1=0,D_0=1** (por ex.), fica activa a saída '**1**' (repare-se: **01** representa '**1**' em binário) – e apenas ela! E, por consequência, ficam activados os multiplexers M**01**1 e M**01**0 – e apenas eles!

- quanto às entradas **I_1I_0**, elas aplicam-se às entradas de selecção {1,2} de oito multiplexers 4:1 – *ordenadas* I_0→'1', I_1→'2'; … Se, por ex., I_1=**1**,I_0=**0**, a saída que neles irá ficar activa é a **'2'** (**10** representa '**2**' em binário) – e apenas essa!

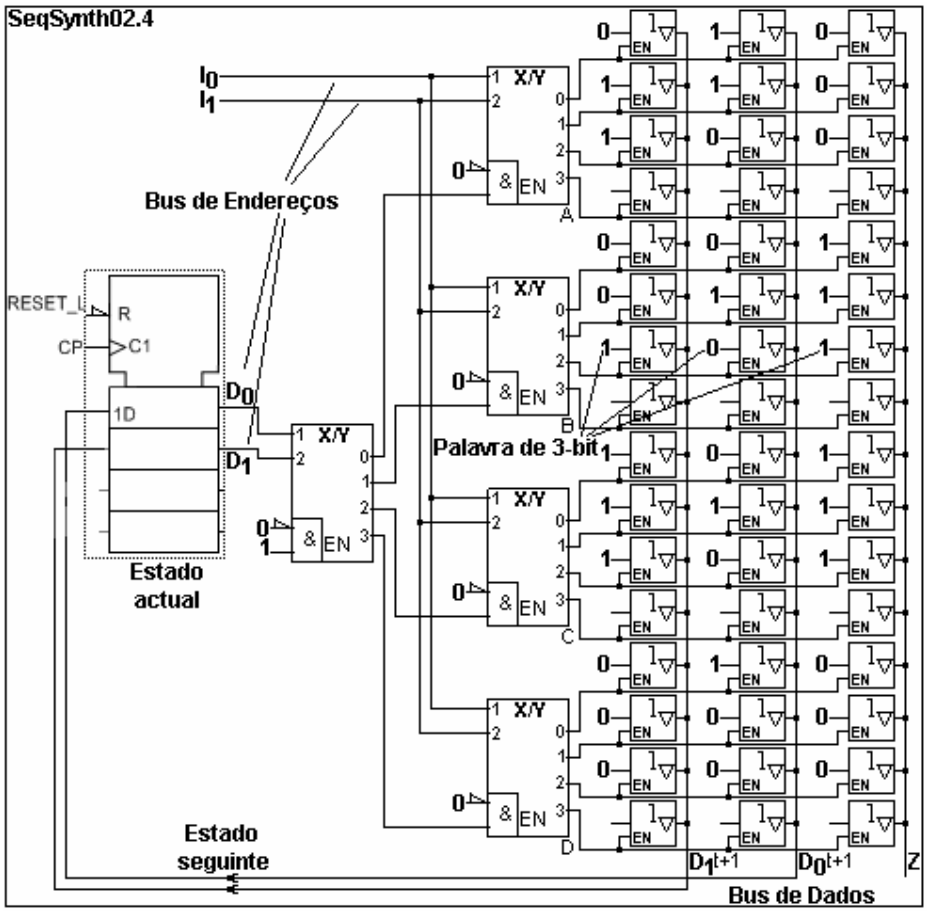
No conjunto, se **D_1=0,D_0=1** e **I_1=1,I_0=0**, irá surgir – como valores *seguintes* de {**D_1D_0**} – o que estiver nas entradas '**2**' dos multiplexers M**01**1 e M**01**0, a saber, e respectivamente: '**1**' e '**0**'. Atente-se que o uso de saídas *3-state* acarreta que, porquanto as entradas *enable* dos restantes multiplexers se encontram então a '**0**', eles estão desactivados – com o que as respectivas saídas como que se encontram soltas 'no ar'…

Quanto à saída do circuito, **Z**, e pois que ela não depende directamente da entradas **I_1I_0**, não são necessários *multiplexers*: são suficientes simples *buffers 3-state*, com um funcionamento que se intui: se **D_1=0,D_0=1** irá surgir em **Z** o que estiver à entrada do *buffer* B**01**, a saber: '**1**'. De novo, o uso de saídas *3-state* acarreta que, porquanto as entradas *enable* dos restantes *buffers* se encontram então a '**0**', eles estão desactivados – com o que as suas saídas como que se encontram 'no ar'…

Globalmente, por conseguinte, se, no estado **D_1=0,D_0=1** (que é o estado B), a entrada for I_1=**1**,I_0=**0**, a malha combinatória gera a saída **Z**=1 e, como estado *seguinte*, **D_1=1,D_0=0** (que é o estado **C**) – e isso é o que o diagrama de estados determina!

A extrapolação para as restantes combinações de entradas e estados é pacífica: basta gravar, nas entradas dos *multiplexers* e *buffers*, '**0**'s e '**1**'s – respeitando o que consta nas tabela acima, {**D_1 D_0 Z**}… Por ex., o *multiplexer* M011 irá conter, nas suas entradas, a tabela de verdade de **D_1** – para a situação em que o estado actual é {**D_1=0**, **D_0=1**}, a saber: '**001x**'.

A um olhar mais atento a SeqSynth02.3 – e pressuposto da parte de alguém que se recorde da constituição íntima de um *multiplexer* – não deverá escapar algum esbanjar de *hardware*: os multiplexers M000 e M001 (por ex.) envolvem, cada um, quatro ANDs, alimentados pelas mesmíssimas entradas, $\mathbf{I_1I_0}$,… Claramente, há aqui uma *duplicação* – que seria bem maior se o circuito houvera que gerar mais saídas… Urge pesquisar por alguma outra solução – e SeqSynth02.4 é uma hipótese:



Em SeqSynth02.4:

– as saídas do *decoder* já presente em SeqSynth02.3 são agora aplicadas às entradas *enable* de quatro *decoders* {nomeados, não por acaso: **A**, …, **D**}; as entradas de selecção de todos eles são $\{I_1I_0\}$ – ordenadas $I_0 \rightarrow$'1', $I_1 \rightarrow$'2'; com isso, se, no estado B, isto é, quando $\mathbf{D_1=0}$, $\mathbf{D_0=1}$, a entrada for $I_1=\mathbf{1},I_0=\mathbf{0}$, fica activada a saída '**2**' do decoder **B** – e apenas essa!

– as saídas desses quatro *decoders* são aplicadas às entradas *enable* de duas "pilhas" de 16 *buffers 3-state*; as respectivas saídas são ligadas a duas colunas, que geram os valores *seguintes* de $\{\mathbf{D_1D_0}\}$. Se se quiser que estes sejam '10' – quando $\mathbf{D_1=0},\mathbf{D_0=1}$ e $I_1=\mathbf{1},I_0=\mathbf{0}$ – bastará então aplicar, nas entradas dos correspondentes *buffers* da saída '2' de B, os valores '10'…

– quanto à produção da saída **Z**, e por mor de conferir a *mesma estrutura* de "pilha" de 16 *buffers*, cada um dos *buffers* que em SeqSynth02.3 suportam os valores de Z é desdobrado em 4 *buffers 3-state* – com a *mesma entrada* e a mesma saída.

SeqSynth02.4 não é uma RAM (nem uma ROM) – mas partilha muitas das suas características, a saber:

– um conjunto de *buffers* para "*armazenar"* um total de 16 * 3 bits – organizados em 16 *palavras* de 3-bit cada;

– um barramento (*bus*) de endereços – $\mathbf{D_1D_0}I_1I_0$ – para seleccionar – perdão, *endereçar* – uma dessas palavras;

– um barramento (*bus*) de dados - $\mathbf{D_1D_0Z}$ – que veicula o conteúdo (3-bit de cada vez) da palavra endereçada.

Para o problema enunciado, é importante compreender que tal *bus* de dados aparece formatado em *dois campos*: um par de fios, $\{\mathbf{D_1D_0}\}$, veiculando o *estado seguinte*, e um terceiro fio, $\{\mathbf{Z}\}$, veiculando a *saída* do circuito. São as saídas duma malha *puramente combinatória*: não dependem de nenhum estado prévio… Imposta uma combinação {Estado-actual + Entradas} no *bus* de endereços, e afora os inevitáveis tempos de propagação, o *bus* de dados oferece uma palavra {Estado-seguinte + Saída}. E, olhando mais de perto, os valores que estão associados aos *buffers* são, coluna-a-coluna, os bits que perfazem as tabelas de verdade das funções $\{\mathbf{D_1D_0Z}\}$! Para responder ao problema enunciado, há então que ter em atenção:

1) as *tabelas de verdade* das funções {Estado-seguinte e Saída};

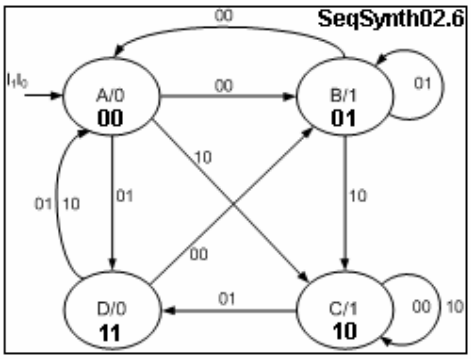
2) o *formato do bus-de-dados,* isto é: a ordenação dos bits em cada palavra. Concretamente, **Z** está mais à direita (é o caso em SeqSynth02.4) ou mais à esquerda? Qual a ordem pela qual estão os fios $\mathbf{D_1D_0}$? Qual está mais a Oeste?...

3) o *formato do bus-de-endereços*: ele também está formatado, há fios que têm a ver com o Estado-actual $\{\mathbf{D_1D_0}\}$, e fios que têm a ver com as entradas do circuito $\{\mathbf{I_1I_0}\}$… Há que distinguir os fios no *bus*-de-endereços: concretamente, às entradas do circuito foram atribuídos os fios de menor peso (é o caso em SeqSynth02.4), ou os de maior peso? etc….

Fechado o parêntesis, constata-se que, no *bus*-de-dados de SeqSynth02.2, **Z** ocupa o fio de "menor-peso" ($\mathbf{Q_0}$), e $\{\mathbf{D_1D_0}\}$ ocupam as posições $\mathbf{Q_2,Q_1}$; quanto ao *bus*-de-endereços, as entradas $\{\mathbf{I_1I_0}\}$ ocupam os fios de menor-peso… Por mor de não correr riscos no preenchimento da tabela, convém registar tais constatações: acrescentaram-se-lhe três colunas à esquerda $\{\mathbf{Estado\text{-}actual},\mathbf{I_1I_0}\}$ e uma linha mais, com os formatos do bus-de-endereços $\{\mathbf{D_1D_0I_1I_0}\}$ e do *bus*-de-dados $\{\mathbf{D_1D_0Z}\}$. O preenchimento da tabela resume-se a inscrever, nas colunas Q_2,Q_1,Q_0, as tabelas de verdade de, respectivamente, $\mathbf{D_1D_0Z}$… Sejam, por ex., as primeiras 4 linhas - para as quais o endereço começa por '**00**', isto é: o estado actual é A; para esse estado, a

tabela de verdade de $\mathbf{D_0}$ regista '**110x**'; então, na coluna '$\mathbf{D_0}$' – que é a coluna $\mathbf{Q_1}$ – inscreve-se '**110x**' nas sucessivas linhas que a constituem (repare-se: na tabela de verdade (*horizontal*), o bit menos significativo está à esquerda, *ela lê-se da esquerda para a direita*; na tabela de memória (*vertical*), o bit menos significativo está no topo, *ela escreve-se de cima-para-baixo*…

Estado actual	I_1	I_0	Address	Q_3	Q_2	Q_1	Q_0
A/00	0	0	**00**00		0	1	0
A/00	0	1	**00**01		1	1	0
A/00	1	0	**00**10		1	0	0
A/00	1	1	**00**11				
B/01	0	0	**01**00		0	0	1
B/01	0	1	**01**01		0	1	1
B/01	1	0	**01**10		1	0	1
B/01	1	1	**01**11				
C/10	0	0	**10**00		1	0	1
C/10	0	1	**10**01		1	1	1
C/10	1	0	**10**10		1	0	1
C/10	1	1	**10**11				
D/11	0	0	**11**00		0	1	0
D/11	0	1	**11**01		0	0	0
D/11	1	0	**11**10		0	0	0
D/11	1	1	**11**11				
Formatos dos *buses*→			**D_1 D_0 I_1 I_0**		**D_1**	**D_0**	**Z**



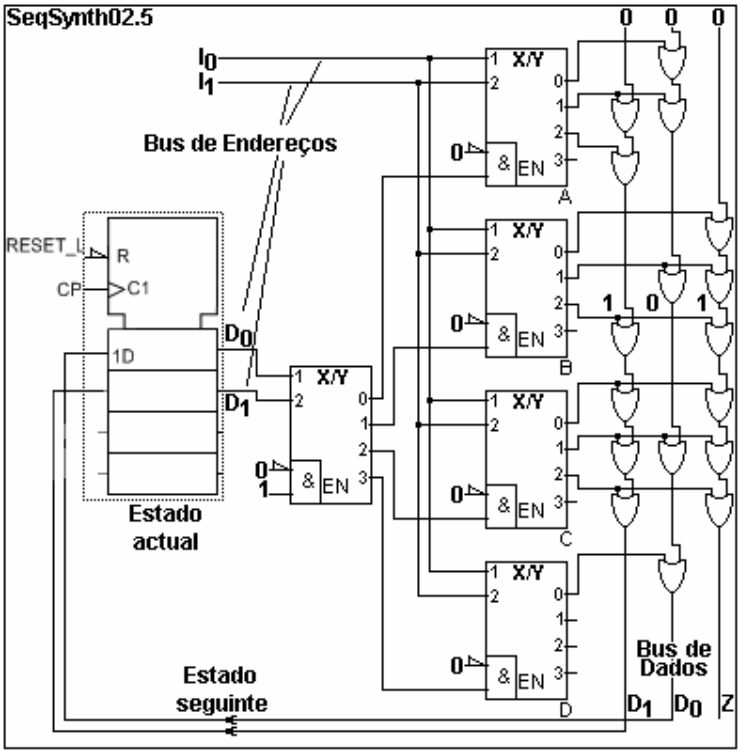
O método mais moroso: Pela abordagem acima, a chave é: *discernir os formatos* dos *buses* de endereços e de dados. Em rigor, pode preencher-se a tabela de memória sem passar pela construção das tabelas de estados/saída! Convirá entretanto *inscrever, no próprio diagrama de estados, a codificação dos estados*, *vid*é SeqSynth02.6; doravante, discorre-se em termos dos estados '**00**', '**01**', … que, não, de A, B…

Suponha-se, então:

- que: se designaram de $\mathbf{D_1D_0}$ os bits do registo dado;
- que se discerniu o formato do *bus* de endereços (é visível na atribuição das posições A(0-3) da RAM: {$\mathbf{D_1D_0}$ I_1 I_0});
- que se discerniu o formato do *bus* de dados (basta reparar que Q_0 produz Z e que Q_1 e Q_2 produzem respectivamente D_0 e D_1).

A etapa seguinte é preencher a linha "Formatos dos *buses*"… e, a partir do diagrama de estados, preencher a tabela de memória, linha a linha:

Seja, como exemplo, a 1ª linha – em que o *Address* é '**0000**'; conclui-se: $\mathbf{D_1D_0}$**=00** e $\mathbf{I_1I_0}$**=00**; procura-se a seta que sai do *estado '00'* e que está *rotulada **00***; ela aponta ao estado **01** – então, nas colunas {$\mathbf{D_1D_0}$} isto é, {$\mathbf{Q_2Q_1}$} inscreve-se **01**; por outro lado, a saída no estado **00** é '**0**'; então, na coluna {**Z**} isto é, {$\mathbf{Q_0}$}, inscreve-se '**0**'. O restante intui-se…



Parêntesis sobre ROMs…

Uma RAM deve ser *aberta* – para poder armazenar amanhã algo diferente… Já uma **RO**M não precisa sê-lo: espera-se de um circuito ***R**ead **O**nly* que preserve inalterado o seu conteúdo (as EPROMs, etc., têm um funcionamento mais flexível, mas isso é irrelevante para a abordagem adiante). Esse carácter *estático* pode levar a questionar o circuito SeqSynth02.4: para quê recorrer a *buffers 3-state* - para armazenar *o que já se conhece de antemão*? SeqSynth02.5 é uma outra solução, em que se suprimem os *buffers* - sendo substituídos por ORs aqueles (e só aqueles) a que estão aplicados '**1**'s.

A solução não será surpresa para quem sabe gerar uma função booleana mediante um *decoder*:

da tabela acima, deduz-se que D_1 se pode exprimir como uma soma de seis mintermos,

$$D_1(D_1, D_0, I_1, I_0) = \sum m(1,2,6,8,9,10)$$

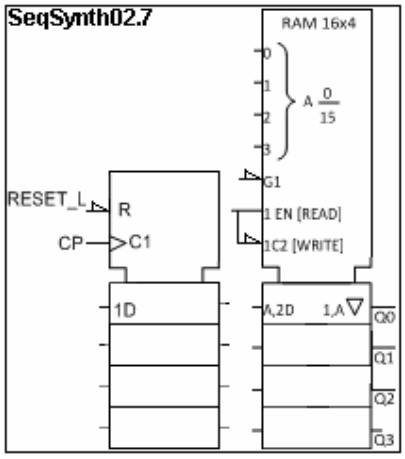
Posto que a cascata de cinco *decoders* produz todos os (16) mintermos possíveis, bastará, para produzir D_1, um OR de seis entradas – ou (pois o OR é associativo), os *seis ORs de duas entradas* explicitados em SeqSynth02.5.

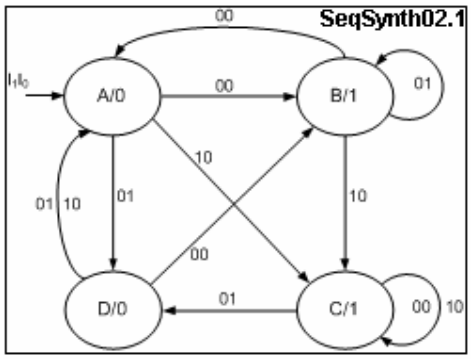
Na prática, se o *Address* for '0110' (por ex.), irá ficar activa – isto é, a '**1**' - a saída '**2**' do *decoder* B (e apenas essa); então, o OR que se encontra na intersecção dessa saída com a coluna D_1 irá recolher esse '**1**', e apresentá-lo em baixo. Em simultâneo, a coluna D_0 apresentará '**0**', e Z apresentará '**1**'. Tudo se passa como se na ROM estivera armazenada a palavra de 3-bit '**101**' – sendo ela agora lograda mediante ligações específicas (ou ausência delas) da saída '2' do *decoder* B às colunas {D_1 D_0 Z}.

A terminar este parêntesis: a ROM figurada é designada de *matriz de AND*s à esquerda (é a massa de que os *decoders* são feitos) e de *matriz de OR*s à direita. Existem outras possibilidades estruturais diferentes – e, para o caso específico das ROMs fornecidas pelos apropriados fabricantes, não espere o leitor vislumbrar aí ORs: encontrará outra coisa, isso dependendo nomeadamente de se é uma PROM, EPROM, EEPROM, etc – mas isso fica para outra noitada…

10. [10E2.11] Considere o diagrama de estados SeqSynth02.1 e o circuito representados em SeqSynth02.7. Assuma que a ROM está programada com o conteúdo indicado na tabela. Complete o circuito de forma a que este implemente o diagrama de estados representado. O circuito tem 2 entradas I_1 e I_0, e uma saída Z.

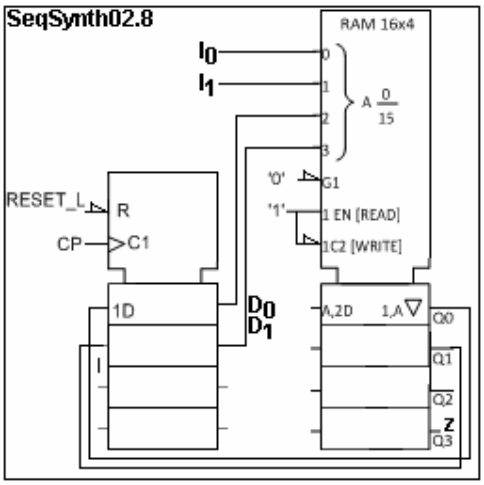
Address	Q3	Q2	Q1	Q0
0000	0	0	0	1
0001	0	0	1	1
0010	0	0	1	0
0011	0	0	0	0
0100	1	0	0	0
0101	1	0	0	1
0110	1	0	1	0
0111	1	0	0	0
1000	1	0	1	0
1001	1	0	1	1
1010	1	0	1	0
1011	1	0	0	0
1100	0	0	0	1
1101	0	0	0	0
1110	0	0	0	0
1111	0	0	0	0





R: O circuito encontra-se em SeqSynth02.8

Processo mental: conforme a resolução [10T2.6], o decisivo é *discernir os formatos dos buses de endereços e de dados*. A esse respeito, Q_2 mantém-se imperturbável em '0'… Pode concluir-se: estão sendo usadas somente as saídas Q_3, Q_1 e Q_0; resta saber: como se distribuem, por elas, Z e os estados seguintes, sejam {D_1D_0}?



O diagrama envolve quatro estados {A,B,C,D}; por cada um, $\mathbf{I_1I_0}$ podem assumir quatro combinações possíveis: {00,01,10,11}. Isso implica que A se desdobra em quatro linhas da tabela da ROM (uma por cada combinação); idem para {B,C,D}. Porquanto a máquina segue o modelo de Moore, é então expectável que a coluna **Z** apresente um número de '**1**'s múltiplo de 4 (e um número de '**0**'s também múltiplo de 4). Isso verifica-se na coluna Q_3 – a conclusão sendo: *Z obtém-se de Q_3*! Por exclusão de partes, {D_1D_0} ocupam as colunas Q_1Q_0!

A saída Z é '1' nos estados {B,C}; por outro lado, Q_3 assume o valor '1' nas linhas {01xx} e {10xx}… A conclusão é: B e C estão codificados '01' e '10' - e os 2-bits à esquerda no *bus-de-endereços* são D_1D_0! Por consequência, os 2-bits à direita são I_1I_0; e, como no diagrama não há setas rotuladas '11', conclui-se: linhas cujo *Address* acaba em '11' são irrelevantes. Falta discernir: qual, entre B e C, estará codificado '01'? No diagrama, a partir de C há duas setas que regressam a C… Ora, na tabela da ROM, para as linhas {01xx}, Q_1Q_0 não se repetem, mas já para as linhas {10xx}, é notória a repetição: nas linhas '1000' e '1010', Q_1Q_0 volvem-se em '10': conclui-se que C se codifica '10' (e B se

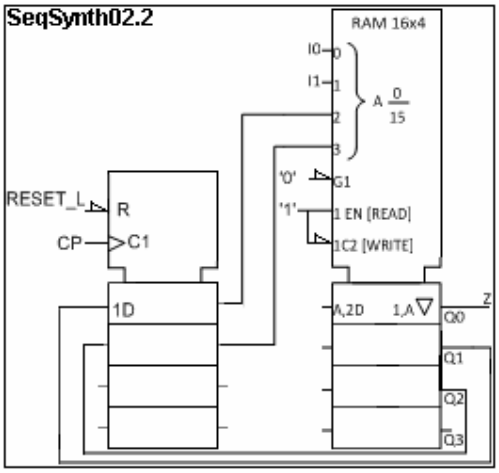
codifica '01') – e que Q_1 *veicula* D_1 (e Q_0 *veicula* D_0), e que o *bit de maior peso do Address veicula* D_1. Enfim, da linha cujo *Address* é '1001', que corresponde à seta que é dirigida de C para fora de C, e que no diagrama está rotulada '01', conclui-se que o *bit de menor peso do Address veicula* I_0. Sob a forma de uma tabela, eis as conclusões:

Estado actual	I_1 I_0	*Address*	Q_3	Q_2	Q_1	Q_0
Formatos dos *buses*→	**D_1 D_0 I_1 I_0**		**Z**		**D_1**	**D_0**

Completar o circuito dado resume-se a um exercício trivial: ligar Q1Q0 às entradas de D_1D_0, e $D_1D_0I_1I_0$ às entradas de endereço da RAM… Ademais, G1 deve ficar *Low*, e EN deve ficar *High*.

11. [10E4.10] Considere o circuito representado em SeqSynth02.2, contendo 2 entradas I_1 e I_0 e uma saída Z. Assuma que a RAM está programada com o conteúdo indicado na tabela.

Address	Q3	Q2	Q1	Q0
0000	1	0	0	1
0001	1	0	1	1
0010	1	0	1	1
0011	1	0	0	1
0100	0	1	1	0
0101	0	1	1	0
0110	0	0	1	0
0111	0	0	0	0
1000	1	1	0	0
1001	1	1	0	0
1010	1	1	0	0
1011	1	1	0	0
1100	0	0	0	1
1101	0	0	1	1
1110	0	0	0	1
1111	0	1	1	1



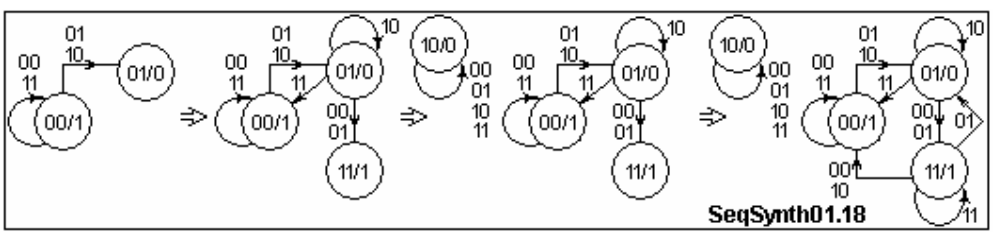
11. 1. Indique justificando se o circuito implementa uma máquina de Mealy ou de Moore.
11. 2. Desenhe o diagrama de estados da máquina implementada pelo circuito

R1: Trata-se de uma máquina de *Moore*. Com efeito, constata-se que, no *bus*-de-dados, **Z** ocupa o fio de "menor-peso" (**Q_0**), e {**D_1D_0**} ocupam as posições **Q_2Q_1**; quanto ao *bus*-de-endereços, as entradas {**I_1I_0**} ocupam os fios de menor-peso, e {**D_1D_0**} ocupam os fios de maior-peso. Acontece que:

- para as primeiras 4 linhas – para as quais o estado é {**D_1=0,D_0=0**} -, a saída é sempre a mesma: Z=Q_0=1;
- para as 4 linhas seguintes – para as quais o estado é {**D_1=0,D_0=1**} -, a saída é sempre a mesma: Z=Q_0=0;
- para as 4 linhas seguintes – para as quais o estado é {**D_1=1,D_0=0**} -, a saída é sempre a mesma: Z=Q_0=0;
- para as 4 linhas seguintes – para as quais o estado é {**D_1=1,D_0=1**} -, a saída é sempre a mesma: Z=Q_0=1.

Quer dizer: *a saída depende unicamente do estado do circuito*: conhecido o estado do circuito, a saída fica totalmente determinada, são irrelevantes os valores nas entradas {**I_1I_0**}. Esta é precisamente a característica de uma máquina de Moore.

R2: SeqSynth01.18 apresenta, à direita, o diagrama de estados do circuito:



Processo mental: Podendo o circuito assumir quatro estados {**D_1D_0**}, então o diagrama de estados irá conter outras tantas batatas, uma por cada um desses estados, designem-se eles de '00', '01', '10' e '11'. As transições entre eles encontram-se especificadas nas linhas '**00**00' a '**00**11', '**01**00' a '**01**11', '**10**00' a '**10**11' e '**11**00' a '**11**11' da RAM, respectivamente:

As primeiras quatro linhas da RAM especificam o estado '**00**': (além da saída que apresenta, '**1**', *vidé* coluna Q_0), dizem (nas colunas {$\mathbf{Q_2Q_1}$}) qual o estado para onde o circuito transita quando, estando em '00', se aplicam os valores '00', '01', '10' e '11' nas entradas I_1I_0:

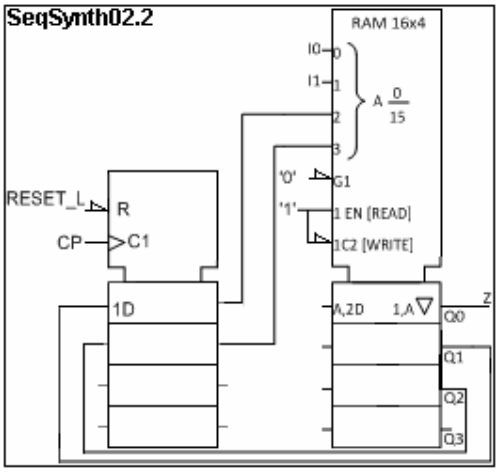
- a linha cujo *address* é '**00**00' estabelece que – para o estado '00' e quando as entradas são '00'- o estado seguinte vem a ser '00' (vide colunas Q_2Q_1); em termos de diagrama de estados, isso representa-se por uma seta saindo da batata '00'e a ela retornando (com uma etiqueta '00'); a linha seguinte, '**00**01' estabelece que – para o mesmo estado '00' mas quando as entradas são '01'- o estado seguinte vem a ser '01'; em termos de diagrama de estado, isso representa-se por uma seta saindo do estado '00' e dirigida para o estado '01' (com uma etiqueta '01'); a linha seguinte, '**00**10' estabelece que – para o mesmo estado '00' mas quando as entradas são '10'- o estado seguinte também vem a ser '01'; em termos de diagrama de estado, isso pode representar-se por mais uma seta saindo do estado '00' e dirigida para o estado '01' (com uma etiqueta '10') – ou, melhor ainda, e posto que já existe uma seta dirigida de '00' para '01', apondo-lhe uma etiqueta mais, '10'; enfim a linha seguinte, '**00**11' estabelece que – para o mesmo estado '00' mas quando as entradas são '11'- o estado seguinte também vem a ser '00'; em termos de diagrama de estado, isso pode representar-se por mais uma seta saindo do estado '00' e a ele retornando (com uma etiqueta '11') – ou, melhor ainda, e posto que já existe uma seta dirigida de '00' para '00', apondo-lhe uma etiqueta mais, '11';

- o procedimento para os restantes estados é, naturalmente, similar – acima se exibindo o filme do desenho…

Enfim, e porquanto se trata de uma máquina de Moore, há que inscrever, em cada batata, a saída que o circuito oferece no estado a que corresponde… No estado '00', isso representa-se inscrevendo '**/1**' imediatamente após o seu *nome*, '00'…

12. [10E3.10] Considere o circuito representado em SeqSynth02.2. Assuma que a RAM está programada com o conteúdo indicado na tabela. O circuito implementa uma máquina de Mealy ou de Moore?

Address	Q3	Q2	Q1	Q0
0000	1	0	0	1
0001	1	0	1	1
0010	1	0	1	0
0011	1	0	0	0
0100	0	1	0	0
0101	0	1	0	1
0110	0	1	1	0
0111	0	1	0	0
1000	1	1	1	0
1001	1	1	1	1
1010	1	1	1	0
1011	1	1	0	0
1100	0	0	0	1
1101	0	0	0	0
1110	0	0	0	0
1111	0	0	0	0



R: Trata-se de uma máquina de *Mealy*. Com efeito, constata-se que, no *bus*-de-dados, **Z** ocupa o fio de "menor-peso" ($\mathbf{Q_0}$), e {$\mathbf{D_1D_0}$} ocupam as posições $\mathbf{Q_2Q_1}$; quanto ao *bus*-de-endereços, as entradas {$\mathbf{I_1I_0}$} ocupam os fios de menor-peso, e {$\mathbf{D_1D_0}$} ocupam os fios de maior-peso. Acontece que para as primeiras 4 linhas – para as quais o estado é {$\mathbf{D_1=0,D_0=0}$} -, a saída *não* é sempre a mesma: ela é, linha a linha, '**1**', '**1**', '**0**' e '**0**'. Quer dizer: *a saída não depende só do estado do circuito*: os valores nas entradas {$\mathbf{I_1I_0}$} não são irrelevantes. Esta é precisamente a característica de uma máquina de Mealy.

(Note-se que a dependência de Z em relação a {$\mathbf{I_1I_0}$} também acontece para os restantes estados {$\mathbf{D_1,D_0}$})