COMPARADORES

Pretende-se através de uma operação a que se chama Comparação, determinar qual a relação existente entre dois números A e B dados em numeração binária.

$A = B$
$A > B$
$A < B$

$$A = \sum_{0}^{n} a_K 2^K$$

$$B = \sum_{0}^{n} b_K 2^K$$

Utilizar-se-ão para tal, processos combinacionais paralelo e série. Os processos referidos não pertencem ao estado.

1- Comparação Paralelo

Os n pares de bits de A e B, a_K e b_K, são comparados simultaneamente sendo a decisão tomada conforme se verifica igualdade ou desigualdade entre os pares referidos.

2- Comparação Série

Os n pares de bits de A e B, a_K e b_K, são comparados par após par começando pelos bits mais significativos.

Naturalmente que as implementações correspondentes aos dois processos não são triviais.

Considerem-se dois números A e B formados por um único bit A_0 e B_0.

Estabeleçam-se as tabelas de verdade e respectivas equações lógicas para os três casos possíveis.

1- $A = B$ 2- $A < B$ 3- $A > B$

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

$X = \bar{A}\bar{B} + AB = \overline{A \oplus B}$

A	B	X
0	0	0
0	1	1
1	0	0
1	1	0

$X = \bar{A}B$

A	B	X
0	0	0
0	1	0
1	0	1
1	1	0

$X = A\bar{B}$

É evidente que a combinação de $A = B$ com qualquer das outras relações em duas a:

$$A \geq B \rightarrow X = \bar{A}\bar{B} + AB + A\bar{B} = A + \bar{B} \equiv \overline{A < B}$$

$$A \leq B \rightarrow X = \bar{A}\bar{B} + AB + \bar{A}B = \bar{A} + B \equiv \overline{A > B}$$

Vai-se-á agora como se pode estabelecer a comparação paralelo entre dois números binários, A e B.

Seja: $A = \sum_{0}^{n} a_k 2^k \qquad B = \sum_{0}^{n} b_k 2^k$

1- $A = B$

Para que $A = B$ terá que ser $a_k = b_k$ para todo o k.

Como $a_k = b_k \Rightarrow \overline{a_k \oplus b_k} \equiv 1$, é necessário que a intersecção de todas as comparações parciais tome o valor 1.

Portanto:

$$X = (\overline{a_n \oplus b_n})(\overline{a_{n-1} \oplus b_{n-1}}) \ldots (\overline{a_0 \oplus b_0}) = \prod_{0}^{n} X_k$$

em que $X_k = \overline{a_k \oplus b_k}$

A implementação correspondente encontra-se na fig. 1.

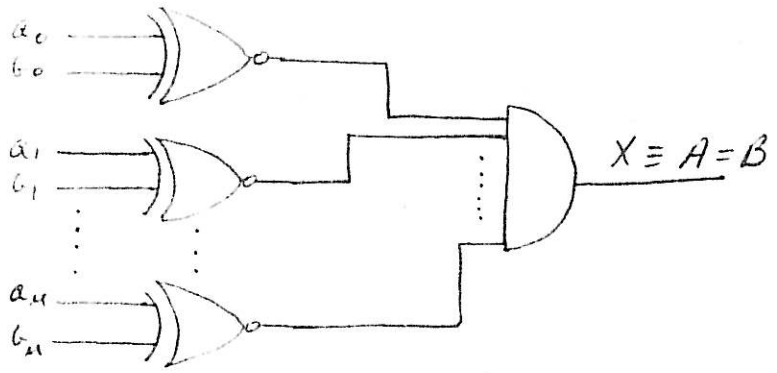


fig. 1

Le n for grande, poderá haver necessidade de optar por soluções intermediárias uma vez que existe limitação do número de variáveis de entrada nos dispositivos lógicos existentes (máximo de 8 entradas utilizando TTL)

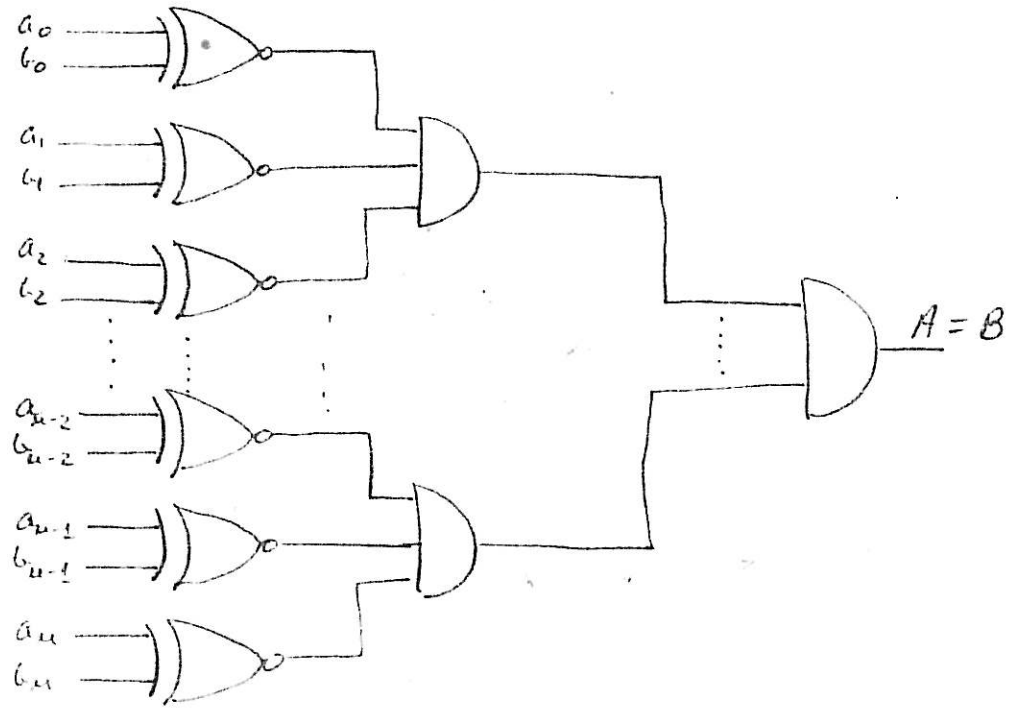


fig. 2

Estas soluções envolvem o mesmo número de funções "Ou" exclusivo havendo apenas desdobramento na função "E" de saída, tendo o inconveniente de o atraso entre as entradas e a saída ser maior.

Na implementação da figura 1 o atraso Δ será:

$\Delta = \Delta_1 + t_d$ em que Δ_1 é o atraso de um "ou" exclusivo negado e t_d o atraso de uma gate "E".

Na implementação da fig. 2 em que se desdobrou uma vez a função "E", o atraso será: $\Delta = \Delta_1 + 2\,t_d$

Uma implementação em que se utilizem n desdobramentos o atraso será: $\Delta = \Delta_1 + (n+1)\,t_d$.

2 - A obtenção da implementação que permitisse determinar se $A < B$ ou $A > B$ pode-se do seguinte algoritmo:

Analisam-se os [redacted] bits mais significativos:

Se $a_n > b_n \Longrightarrow A > B$

Se $a_n < b_n \Longrightarrow A < B$

Se $a_n = b_n \Longrightarrow$ nada pode ser concluído pelo que se passa à comparação de a_{n-1} e b_{n-1}

a) $A > B$ $\qquad A = \sum_{0}^{n} a_k 2^k \qquad B = \sum_{0}^{n} b_k 2^k$

$a_n > b_n \longrightarrow$ decidido

$a_n = b_n \longrightarrow \begin{cases} a_{n-1} > b_{n-1} \longrightarrow \text{decidido} \\ a_{n-1} = b_{n-1} \longrightarrow \begin{cases} a_{n-2} > b_{n-2} \longrightarrow \text{decidido} \\ a_{n-2} = b_{n-2} \longrightarrow \text{etc.} \end{cases} \end{cases}$

Isto traduz-se do seguinte modo:

$$X = a_n > b_n + (a_n = b_n)\left[a_{n-1} > b_{n-1} + (a_{n-1} = b_{n-1})\left[a_{n-2} > b_{n-2} + (a_{n-2} = b_{n-2}) \text{ etc.}\right]\right]$$

Por exemplo, para $n = 3$, vinha:

$$A = a_2 2^2 + a_1 2^1 + a_0 2^0 \qquad B = b_2 2^2 + b_1 2^1 + b_0 2^0$$

$$X = a_2 > b_2 + (a_2 = b_2)\left[a_1 > b_1 + (a_1 = b_1)(a_0 > b_0)\right]$$

Ou seja:

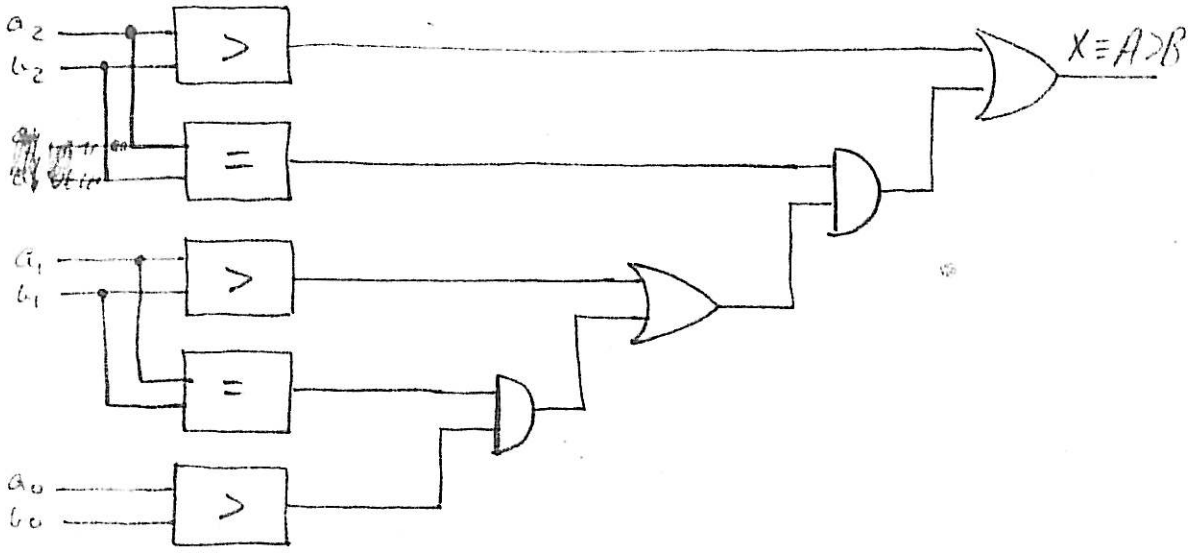


fig. 3

Como já se sabe que:

- a função lógica $a_n \bar{b}_n$ determina $a_n > b_n$
- " " " $\overline{a_n \oplus b_n}$ " $a_n = b_n$

facilmente se implementaria um circuito que decidisse se $A > B$.

Uma implementação deste tipo tem o inconveniente de o tempo de atraso aumentar quando aumenta n. Assim pode-se pensar noutro tipo de implementação que minimize o atraso entre as saídas e a saída.

Retomando a expressão:

$$X = (a_2 > b_2) + (a_2 = b_2)\left[(a_1 > b_1) + (a_1 = b_1)(a_0 > b_0)\right]$$

ou seja:

$$X = a_2 \bar{b}_2 + (\bar{a}_2 \bar{b}_2 + a_2 b_2)\left[a_1 \bar{b}_1 + (\bar{a}_1 \bar{b}_1 + a_1 b_1) a_0 \bar{b}_0\right]$$

Desenvolvendo:

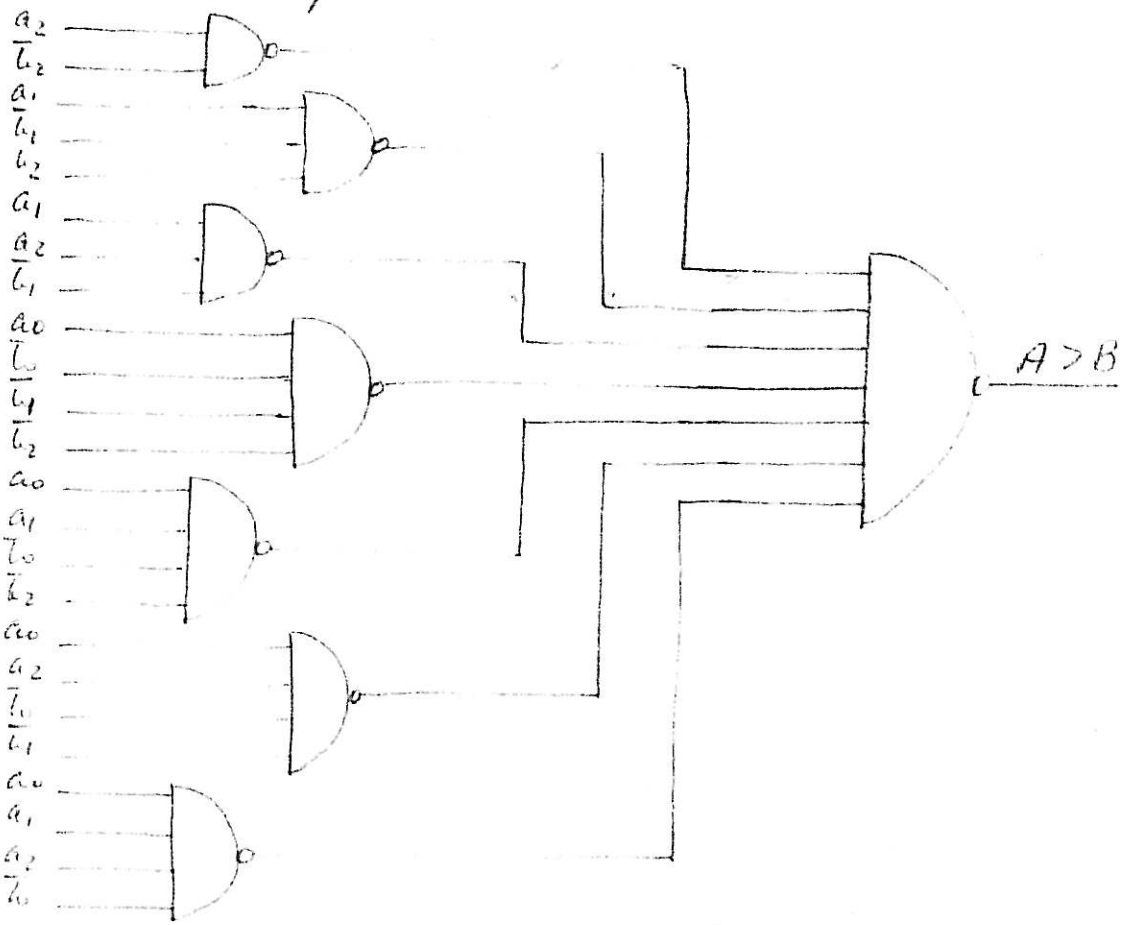
$$X = a_2 \bar{b}_2 + (\bar{a}_2 \bar{b}_2 + a_2 b_2)(a_1 \bar{b}_1 + a_0 \bar{a}_1 \bar{b}_0 \bar{b}_1 + a_0 a_1 \bar{b}_0 b_1)$$

$$X = a_2 \bar{b}_2 + a_1 \bar{a}_2 \bar{b}_1 \bar{b}_2 + a_0 \bar{a}_1 \bar{a}_2 \bar{b}_0 \bar{b}_1 \bar{b}_2 + a_0 a_1 \bar{a}_2 \bar{b}_0 b_1 \bar{b}_2 +$$
$$+ a_1 a_2 \bar{b}_1 b_2 + a_0 \bar{a}_1 a_2 \bar{b}_0 \bar{b}_1 b_2 + a_0 a_1 a_2 \bar{b}_0 b_1 b_2$$

Simplificando esta função por qualquer dos métodos de simplificação, obtem-se uma expressão mais simples. Por exemplo uma simplificação pelo método de Karnaugh, poderia conduzir a:

$$X = a_2 \bar{b}_2 + a_1 \bar{b}_1 b_2 + a_1 a_2 \bar{b}_1 + a_0 \bar{b}_0 \bar{b}_1 \bar{b}_2 + a_0 a_1 \bar{b}_0 \bar{b}_2 +$$
$$+ a_0 a_2 \bar{b}_0 \bar{b}_1 + a_0 a_1 a_2 \bar{b}_0$$

ou seja:



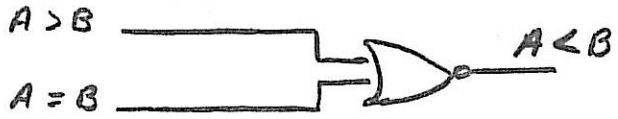
Repare-se que à medida que o número de bits aumenta também o número máximo de entradas aumenta pelo que se terá que fazer, igualmente, desdobramentos, aumentando o tempo de atraso. De qualquer modo o atraso introduzido será menor que no tipo de implementação anteriormente apresentado.

b) $A < B$

Por exclusão de hipóteses:

Se $A \neq B$ ou $A \not> B$, então $A < B$.

Portanto:



Por outro lado trocando A com B em qualquer das implementações anteriores obter-se-ia o mesmo resultado. De facto se $A < B$ então é $B > A$.

Poder-se-ia ainda desenvolver um algoritmo semelhante ao tratado para $A > B$, ou seja:

$$X \equiv A < B \equiv (a_n < b_n) + (a_n = b_n)\left[a_{n-1} < b_{n-1} + \text{etc.}\right.$$

Ou:

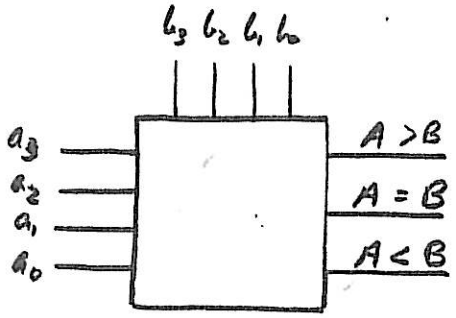
$$X = \bar{a}_n b_n + \overline{a_n \oplus b_n}\left[\bar{a}_{n-1} b_{n-1} + \text{etc.}\right.$$

Conclusão:

O atraso permitido entre as entradas e as saídas determina a escolha da implementação. A tecnologia existente permite qualquer das soluções, desde que se tome em consideração as limitações apontadas (máximo número de entradas permitido).

———— // ————

Existem no mercado comparadores integrados utilizando técnicas de M S I (Medium Scale Integration). Os mais vulgares comparam dois números de quatro bits.

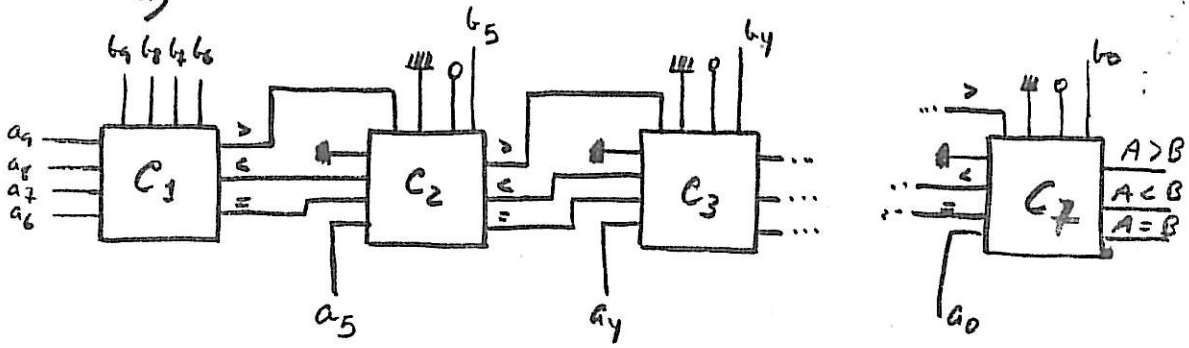


Associando estes blocos de base consegue-se comparadores de dois números formados por qualquer número de bits. As possibilidades de implementação são várias. A título exemplificativo apresenta-se em seguida alguma, em que os números a comparar são constituídos por 10 bits:

$$A = a_9 2^9 + a_8 2^8 + a_7 2^7 + \dots + a_0$$

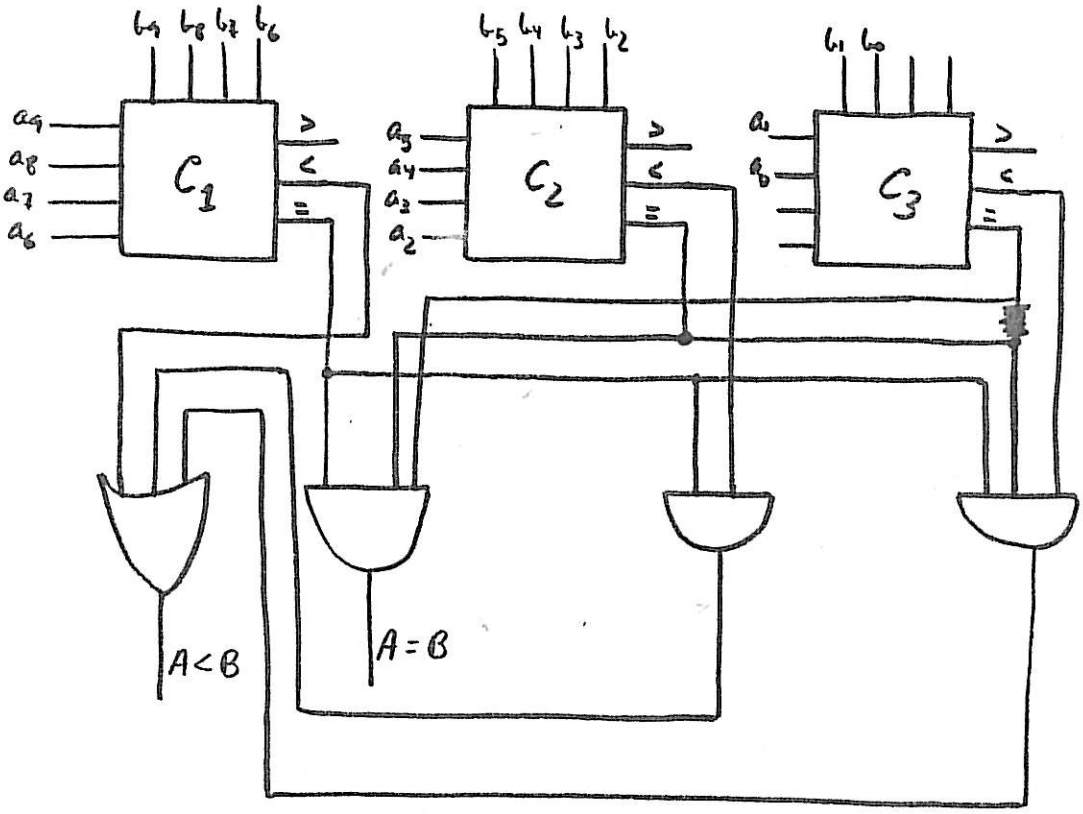
$$B = b_9 2^9 + b_8 2^8 + b_7 2^7 + \dots + b_0$$

a)

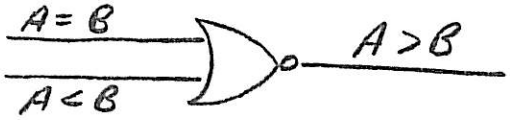


Esta solução consome um número desproporcionado de comparadores e é dificilmente recomendável, sendo apresentada apenas com fins didácticos.

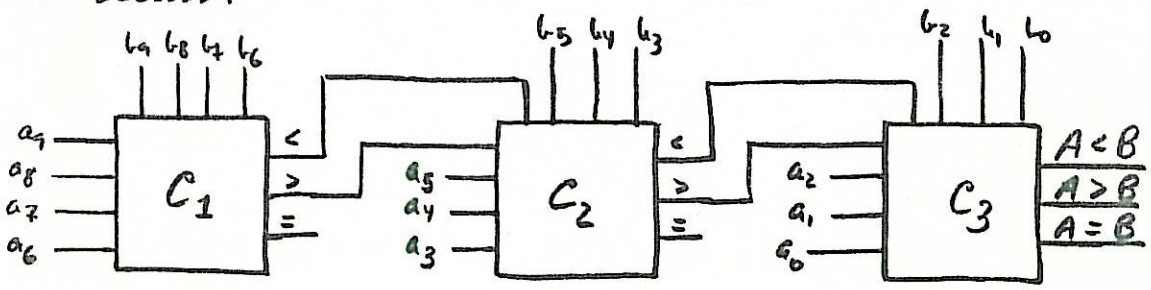
b) Uma implementação no extremo oposto será:



Para se obter A>B poder-se-ia implementar algo semelhante a A<B ou então, por exclusão de partes:



c) Uma solução que nesta caso (10 bits) parece de aconselhar será intermédia destas duas:



Se C_1 indicar $A<B$ (ou $A>B$) o bit mais significativo de C_2 da entrada B (ou A) fica em "1", e o mais significativo da entrada A (ou B) em "0" o que obriga C_2 a indicar $<$ ($>$). Por sua vez isto obriga a que C_3 indique $A<B$ ($A>B$), que é o resultado correcto.

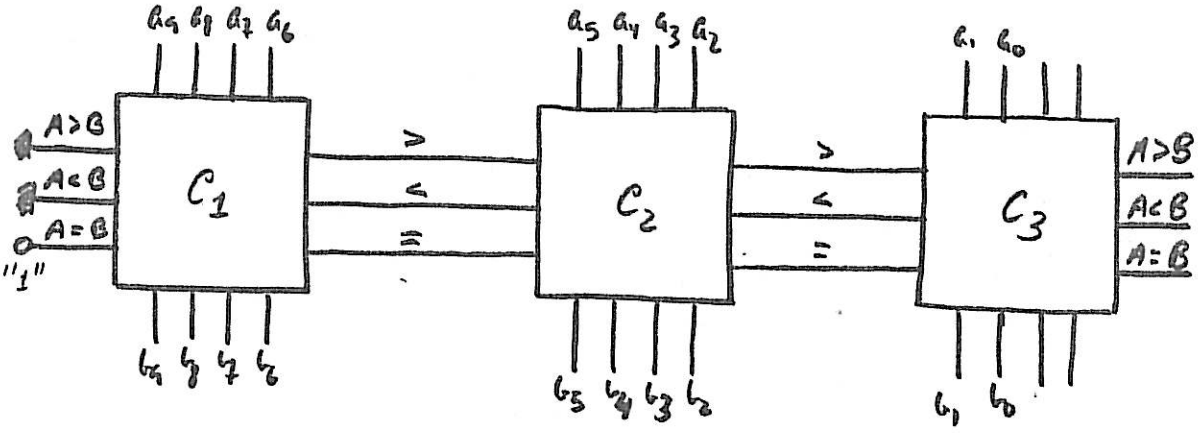
Se C_1 indica $A=B$, a decisão é transferida para C_2. Se este decide $A<B$ ou $A>B$ tudo se passa como anteriormente.

Se este decide $A=B$, é C_3 que fica encarregado da decisão final, apresentando o resultado nos seus terminais $A<B$, $A>B$ e $A=B$.

——— // ———

Outro tipo de comparadores integrados existentes, adicionam mais três entradas que se destinam a receber os sinais $A>B$, $A<B$ ou $A=B$ de comparações anteriores.

Utilizando este tipo de comparadores a resolução do problema anterior (A e B formados por 10 bits) conduziria à implementação mostrada a seguir.



As saídas A>B, A<B e A=B de C_3 dão a ordem de grandeza existente entre A e B.

DESCODIFICAÇÃO

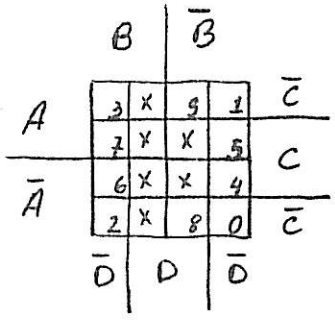
Considere-se um código binário. A descodificação de qualquer palavra do código poderá ser realizada a partir da intersecção ("E" lógico) das variáveis tomadas na sua forma negada ou não consoante na palavra considerada o seu valor seja "zero" ou "um".

Tome-se como exemplo o código BCD

	D C B A	
0	0 0 0 0	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$
1	0 0 0 1	$A\,\bar{B}\,\bar{C}\,\bar{D}$
2	0 0 1 0	$\bar{A}\,B\,\bar{C}\,\bar{D}$
3	0 0 1 1	$A\,B\,\bar{C}\,\bar{D}$
4	0 1 0 0	$\bar{A}\,\bar{B}\,C\,\bar{D}$
5	0 1 0 1	$A\,\bar{B}\,C\,\bar{D}$
6	0 1 1 0	$\bar{A}\,B\,C\,\bar{D}$
7	0 1 1 1	$A\,B\,C\,\bar{D}$
8	1 0 0 0	$\bar{A}\,\bar{B}\,\bar{C}\,D$
9	1 0 0 1	$A\,\bar{B}\,\bar{C}\,D$

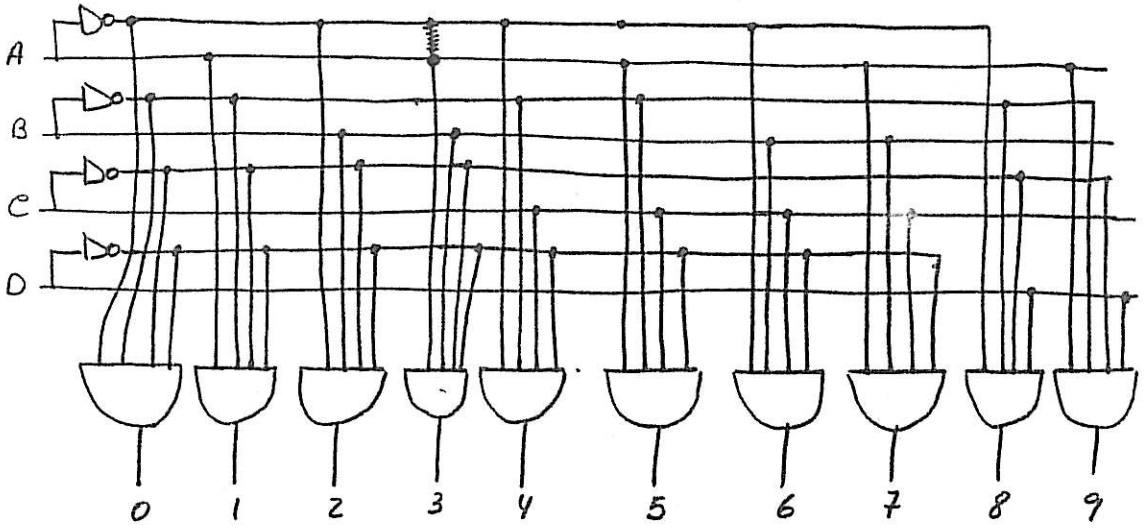
A descodificação das palavras correspondentes aos níveis mínimos é a indicada.

Recomenda-se a métodos gráficos de simplificação e utilizando os "don't care states" é possível chegar a simplificação envolvendo menos variáveis

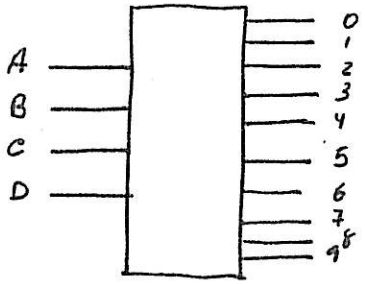


0	$\bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$
1	$A\,\bar{B}\,\bar{C}\,\bar{D}$
2	$\bar{A}\,B\,\bar{C}$
3	$A\,B\,\bar{C}$
4	$\bar{A}\,\bar{B}\,C$
5	$A\,\bar{B}\,C$
6	$\bar{A}\,B\,C$
7	$A\,B\,C$
8	$\bar{A}\,D$
9	$A\,D$

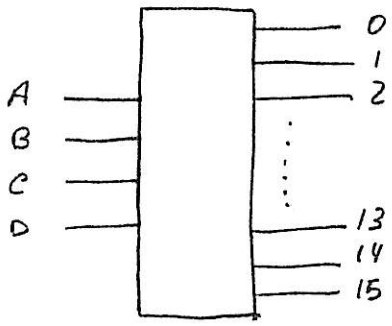
A implementação de um descodificador será realizada do seguinte modo:



e poderá ser representado por:



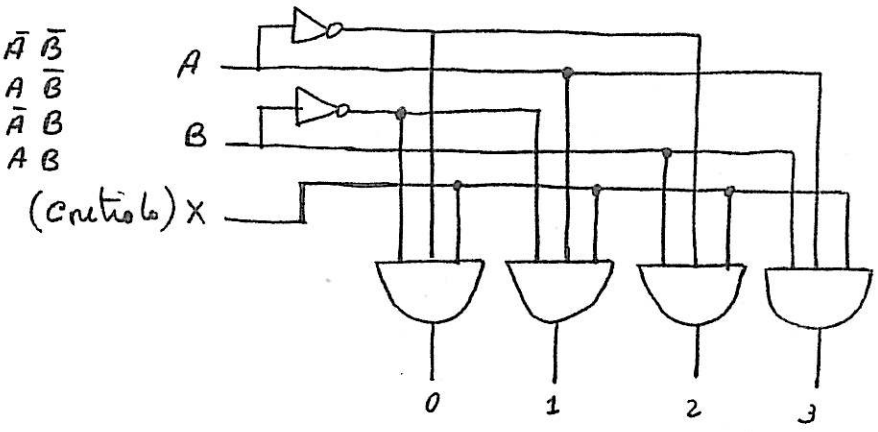
Se o código for o binário pura, para quatro variáveis, então o número de palavras possíveis será 16 (0 a 15) e a sua representação será:



É possível a introdução de uma variável de controlo que inibe a acção das variáveis de entrada.

Exemplifique-se para o código binário de duas variáveis

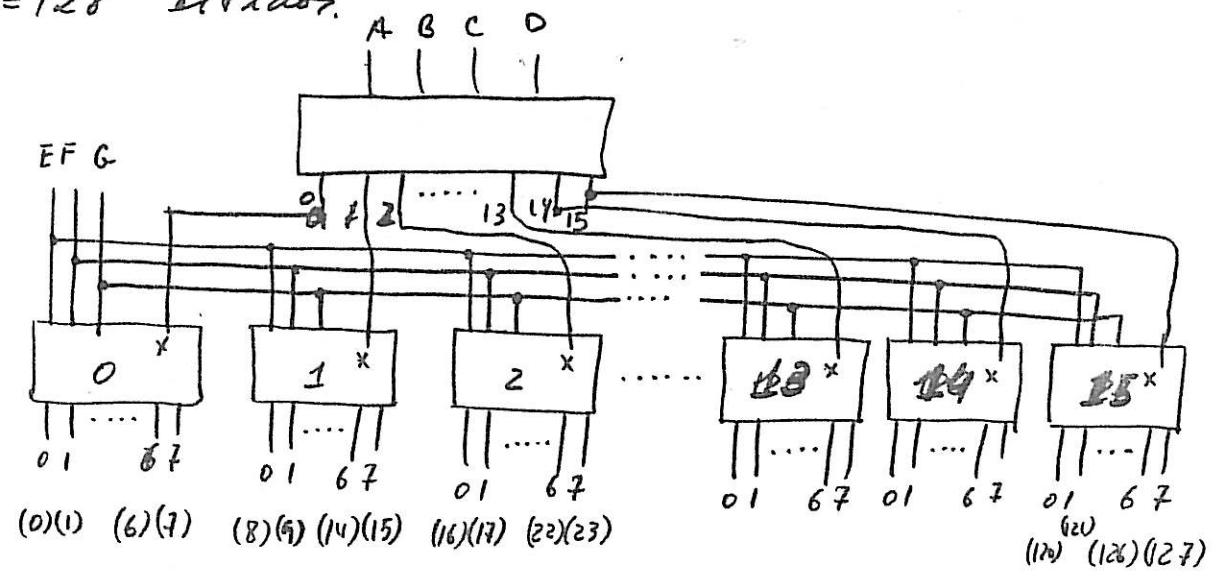
	A	B	
0	0	0	$\bar{A}\,\bar{B}$
1	0	1	$A\,\bar{B}$
2	1	0	$\bar{A}\,B$
3	1	1	$A\,B$



Desde que a variável denominada controlo tenha o valor "zero" todas as saídas do descodificador se encontram também em zero

Considere-se então um descodificador de oito estados (3 variáveis) e uma variável de inibição.

A partir de um descodificador de 16 estados e 16 de 8 é possível descodificar $16 \times 8 = 128$ estados.



. Se A=B=C=D=0 então a saída "0" do descodificador de 16 entradas ficará activa e inibirá todos os descodificadores de 8 entradas excepto o indicado por "0" (o primeiro). Então qualquer das saídas do 0 a 7 poderá estar activa dependendo de E,F,G. É assim possível seleccionar 128 saídas.

É evidente que se obtinha um resultado semelhante se se descodificasse directamente 7 bits

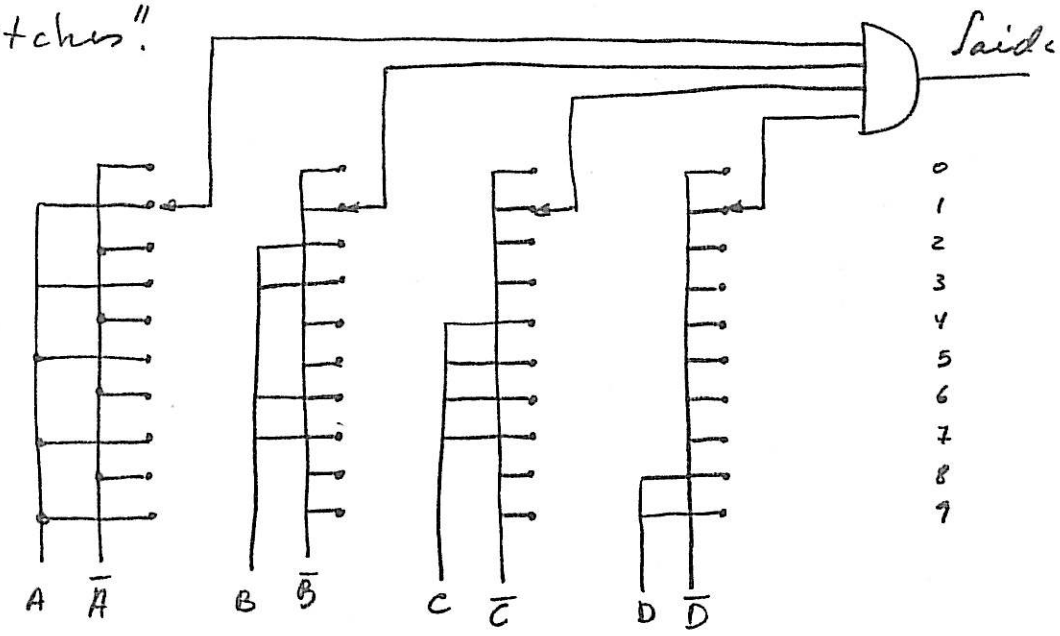
A	B	C	D	E	F	G	
0	0	0	0	0	0	0	(0)
0	0	0	0	0	0	1	(1)
0	0	0	0	0	1	0	(2)
⋮							
1	1	1	1	1	0	1	(125)
1	1	1	1	1	1	0	(126)
1	1	1	1	1	1	1	(127)

A vantagem da configuração apresentada neste está se apoiar na tecnologia existente (Descodificador de 8 e 16).

A implementação directa a partir de gates obrigaria ao uso de 128 "gates" de 7 entradas ou seja 128 pacotes. O mesmo circuito implementa-se, como se vive, a partir de 16+1 = 17 circuitos.

DESCODIFICADORES MECÂNICOS

Referir-nos-emos apenas aos "Thumbwheel Switches".



Quando o código de entrada corresponde ao número mecânicamente relacionado a saída assume o valor lógico "1".

Ex.: A relecção está feita para o número 1. Assim quando:

$$A = 1 \Longrightarrow \bar{A} = 0$$
$$B = 0 \Longrightarrow \bar{B} = 1$$
$$C = 0 \Longrightarrow \bar{C} = 1$$
$$D = 0 \Longrightarrow \bar{D} = 1$$

como à entrada da gate se encontra A, $\bar{B}$, $\bar{C}$ e $\bar{D}$ a saída toma o valor "1."