Capítulo III
Circuitos Digitais Combinacionais

1 Introdução

Vimos no Capítulo II que uma desejada função lógica pode ser implementada mediante a combinação de portas lógicas. Esta combinação de portas lógicas objetivando atender um mapeamento $Y = f(A, B, \cdots)$ é denominada de **Circuito Digital**. Neste Capítulo estudaremos circuitos digitais que visam solucionar problemas específicos e comuns em Eletrônica Digital.

2 Decodificadores para *Display* de 7 Segmentos

● Um *display* de 7 segmentos mostra ao usuário de um sistema digital um algarismo decimal de 0 a 9, conforme mostram as Figuras 1, 2 e 3.

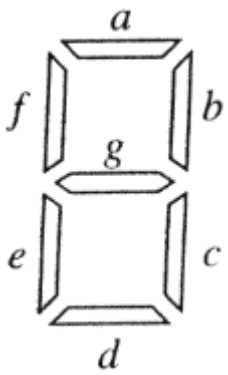


Figura 1: Formato de um *display* de 7 segmentos mostrando a localização dos segmentos *a, b, c, d, e, f, g*.

Figura 2: Algarismos decimais possíveis de serem formados mediante o acionamento combinado dos segmentos *a, b, c, d, e, f, g*.

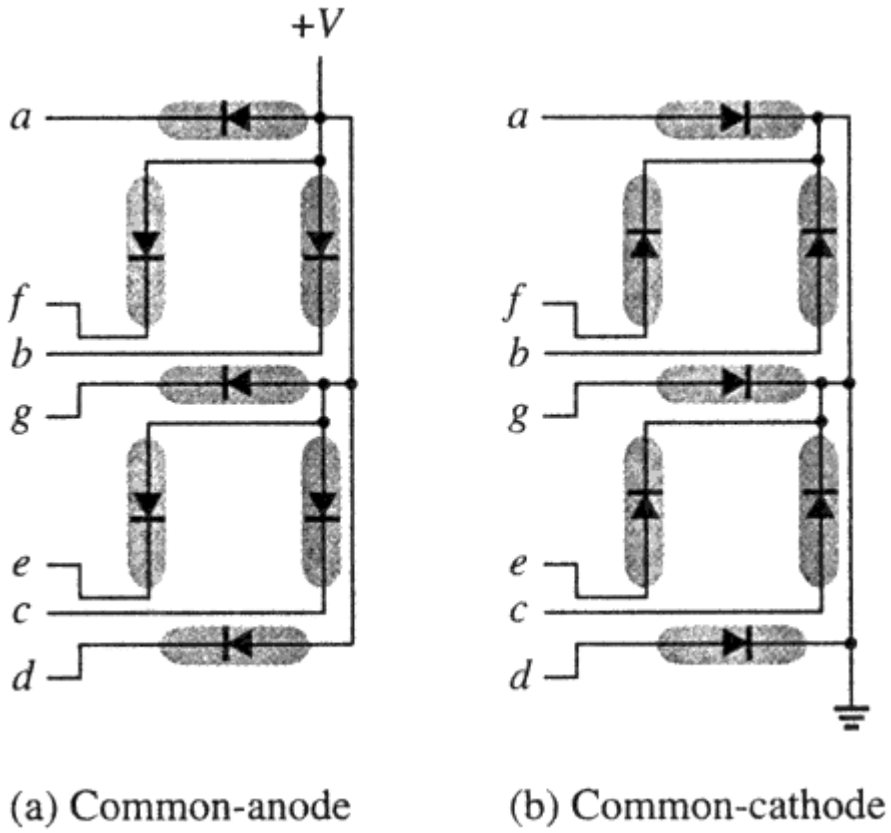


Figura 3: Acionamento dos segmentos *a, b, c, d, e, f, g*. Cada segmento é um LED (*Light Emitter Diode*), que emite luz quando o diodo é percorrido por uma corrente direta ($1\,\text{mA}$ a $50\,\text{mA}$). (a) *display* de anodo comum. (b) *display* de catodo comum.

● Um *display* de 7 segmentos alternativo é o denominado LCD (*Liquid Crystal Display*), largamente utilizado em relógios digitais de pulso. A Figura 4 mostra a técnica de acionamento dos segmentos *a, b, c, d, e, f, g* de um LCD.

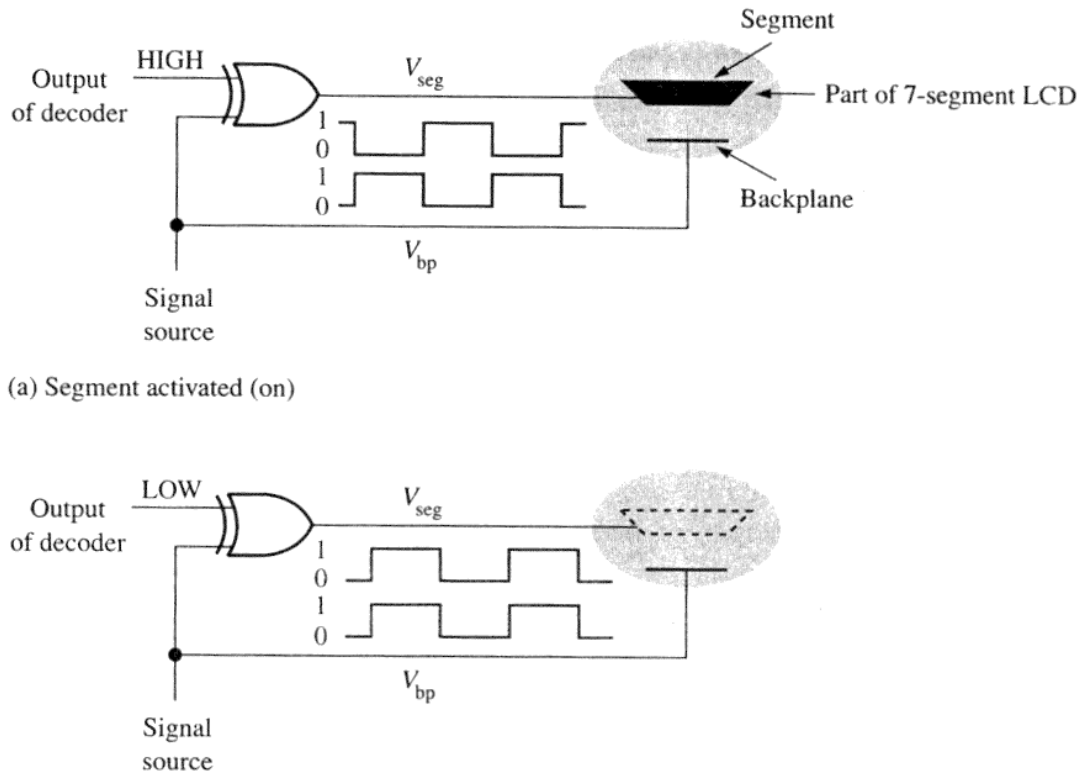


Figura 4: Acionamento dos segmentos *a, b, c, d, e, f, g* de um LCD. Um segmento é acionado por uma onda quadrada com freqüência entre 30 a 60Hz aplicada entre o segmento e uma superfície comum a todos os segmentos denominada *backplane*. Quando a onda quadrada (*signal source*) é aplicada ao segmento através do controle exercido pela porta XOR, o segmento assim ativado deixa de refletir a luz incidente, alterando sua tonalidade para um cinza escuro.

- A Tabela 1 mostra os algarismos decimais resultantes do acionamento combinado dos segmentos *a, b, c, d, e, f, g*:

Digit	Segments Activated
0	*a, b, c, d, e, f*
1	*b, c*
2	*a, b, d, e, g*
3	*a, b, c, d, g*
4	*b, c, f, g*
5	*a, c, d, f, g*
6	*a, c, d, e, f, g*
7	*a, b, c*
8	*a, b, c, d, e, f, g*
9	*a, b, c, d, f, g*

Tabela 1: Algarismos decimais resultantes do acionamento combinado dos segmentos *a, b, c, d, e, f, g*. Ver Figura 1.

● Um **Decodificador para *Display* de 7 Segmentos** é um circuito digital formado por portas lógicas que, ao receber uma palavra binária de 4 bits representativa do algarismo decimal a ser mostrado, aciona os segmentos correspondente no *display*, conforme mostram a Figuras 5 e a Tabela 2.

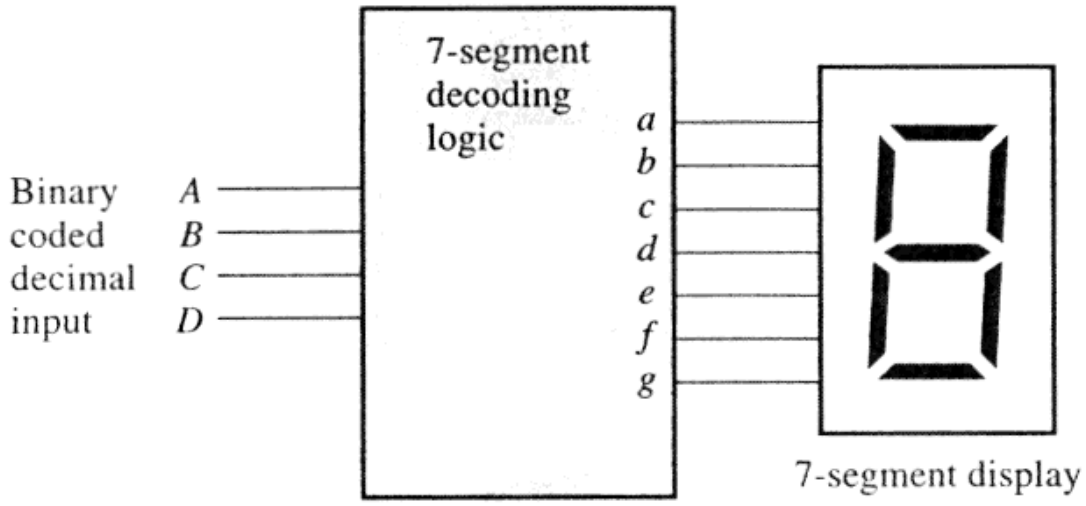


Figura 5: Interligação de um Decodificador para *Display* de 7 Segmentos com o *Display*.

Decimal Digit	Inputs D	C	B	A	Segment Outputs a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Output = 1 means segment is activated (on)
Output = 0 means segment is not activated (off)
Output = X means "don't care"

Tabela 2: Tabela Verdade de um Decodificador para *Display* de 7 Segmentos.

⇒ Observe que a coluna 1 da Tabela 2 representa o número decimal correspondente à palavra binária respectiva na coluna 2 da tabela através da relação: $\text{NúmeroDecimal} = D \cdot 2^3 + C \cdot 2^2 + B \cdot 2^1 + A \cdot 2^0$

- Um Decodificador para *Display* de 7 Segmentos é um Circuito Integrado que contém as combinações de portas lógicas necessárias e otimizadas para a implementação do conjunto de Expressões Booleanas definidas pela Tabela 2.

- Por exemplo, da Tabela 2 verificamos que a Expressão Booleana para o segmento a é:

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

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

- Cujo Mapa K é:

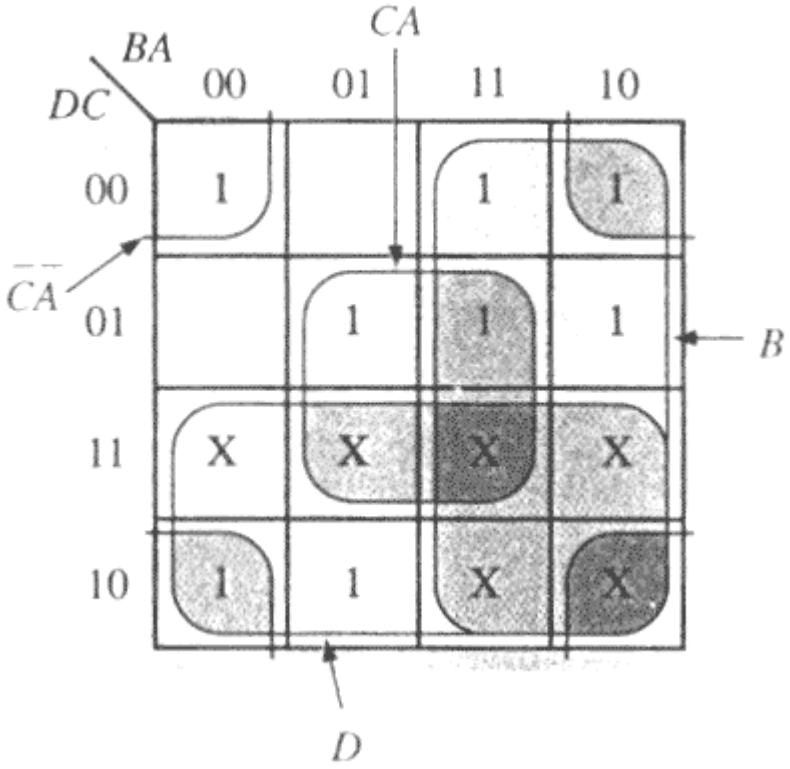


Figura 6: Mapa K para a lógica de acionamento do segmento a. A Expressão Booleana minimizada resulta em $a = D + B + C\,A + \overline{C}\,\overline{A}$.

● E cujo circuito lógico resultante é:

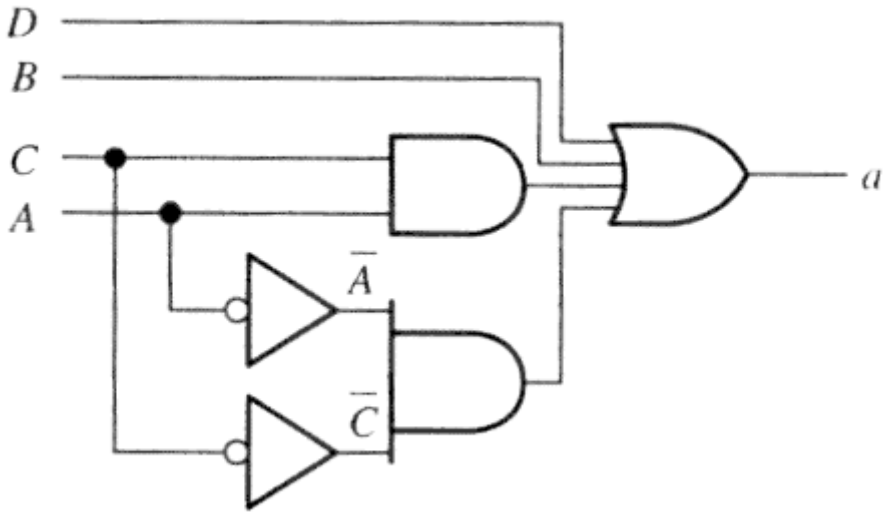


Figura 7: Circuito lógico para acionamento do segmento a. A Expressão Booleana implementada é $a = D + B + C\,A + \overline{C}\,\overline{A}$.

● **Exercício Proposto:** Determine o circuito lógico completo para o acionamento dos segmentos *a, b, c, d, e, f, g*. Caso, após a minimização individual das expressões booleanas para cada segmento, as funções lógicas resultantes para o acionamento de dois ou mais segmentos compartilharem termos comuns, faça a minimização adicional aproveitando o compartilhamento entre os termos.

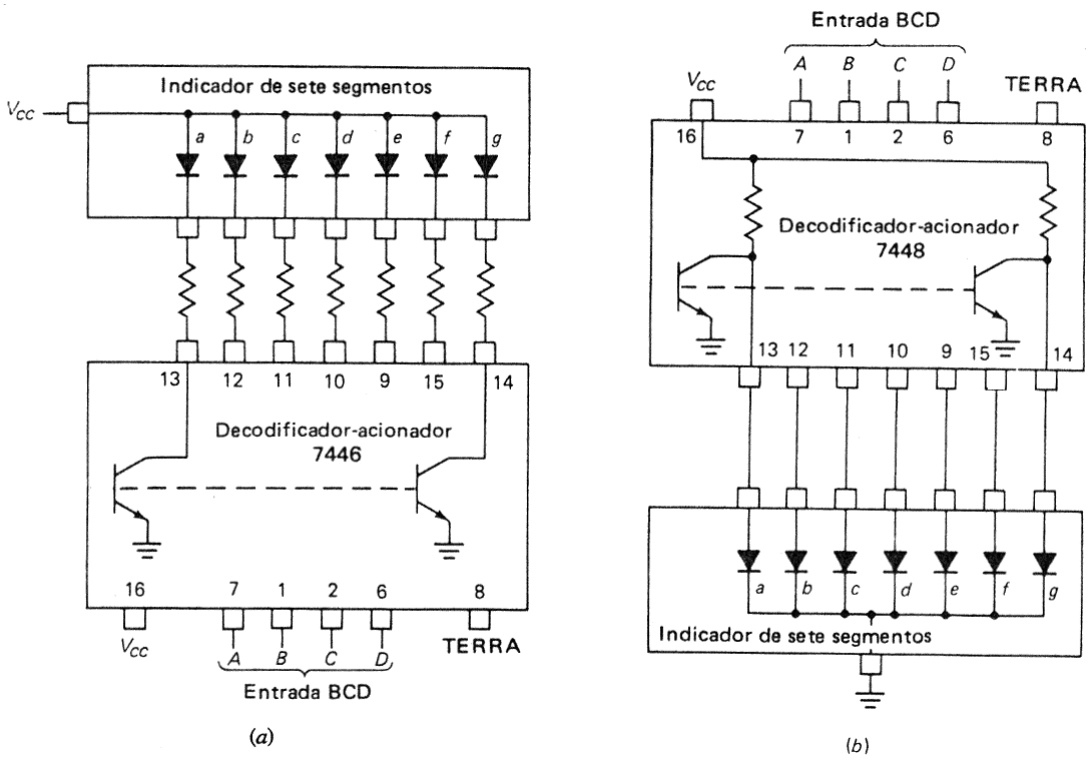


Figura 8: Circuitos Integrados TTL comercialmente disponíveis para a implementação da função de Decodificador para *Display* de 7 Segmentos. (a) 7446 – decodificador para *display* de anodo comum. (b) 7448 – decodificador para *display* de catodo comum.

3 Decodificadores BCD-para-Decimal

● BCD é a abreviação para Decimal Codificado em Binário (*Binary Coded Decimal*).

● O código BCD expressa cada dígito de um número decimal por uma palavra binária de 4 bits (*Nibble*) no formato $b_3\, b_2\, b_1\, b_0$ através da relação: $\text{NúmeroDecimal} = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$. A Tabela 3 mostra o resultado desta relação.

Nibble				Número Decimal
b_3	b_2	b_1	b_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Tabela 3: Tabela para conversão de um *Nibble* em um Número Decimal. A conversão obedece a relação $\text{NúmeroDecimal} = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$.

- As entradas $DCBA$ de um Decodificador para *Display* de 7 Segmentos (ver Seção 1) constituem um exemplo de informação binária codificada em BCD.

- Por exemplo, o número decimal 8963 codificado em BCD resulta em (ver Tabela 3):

8	9	6	3
↓	↓	↓	↓
1000	1001	0110	0011

- Por outro lado, o número binário 010101111000 codificado em BCD, quando convertido para decimal resulta em

0101	0111	1000
↓	↓	↓
5	7	8

- A Figura 9 mostra o diagrama interno de um Decodificador BCD-para-Decimal.

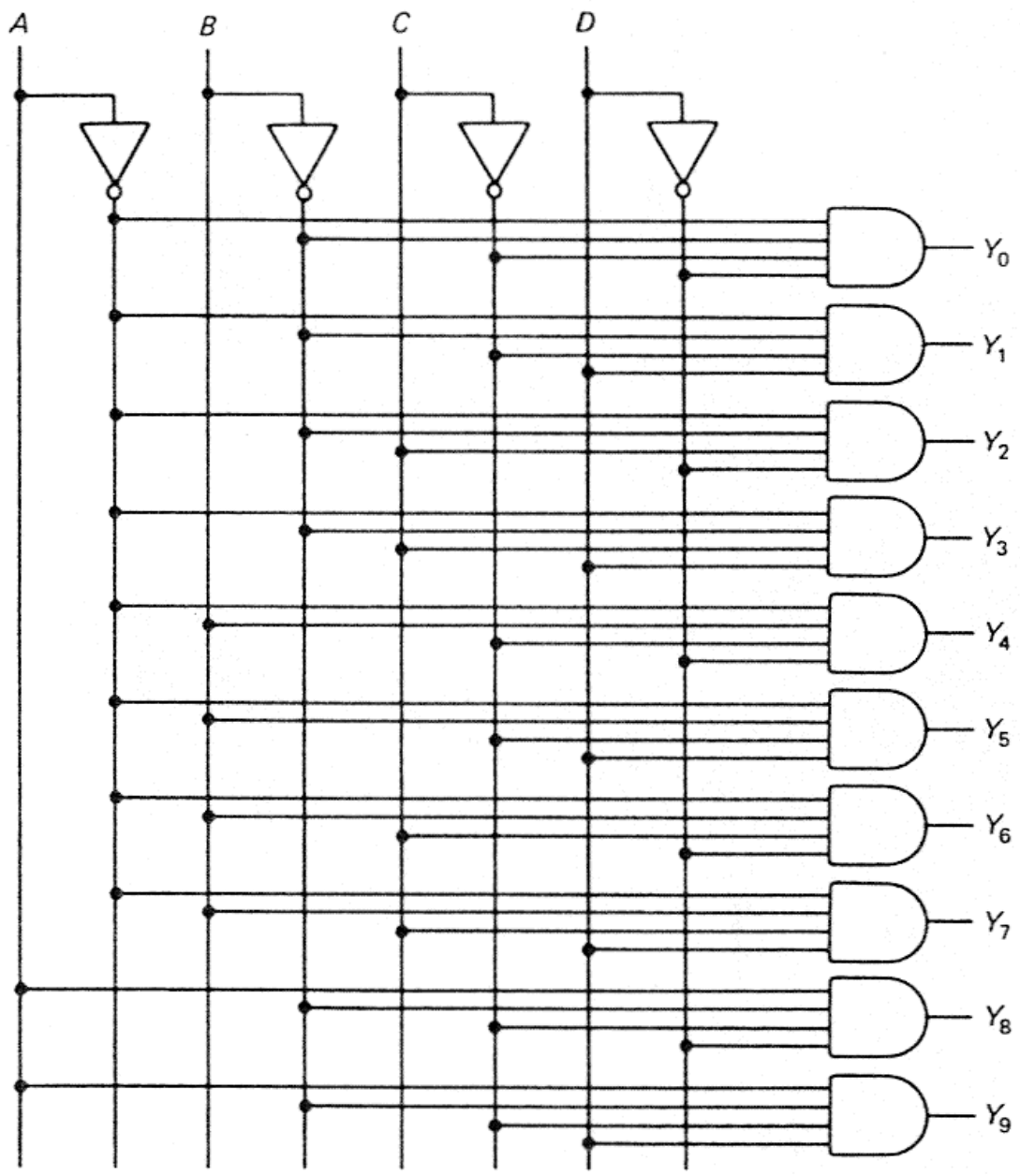


Figura 9: Diagrama interno de um Decodificador BCD-para-Decimal. Este decodificador é conhecido como **decodificador 1-de-10**, porque para cada *Nibble* $ABCD$ na entrada do decodificador, somente uma das 10 saídas está em nível lógico 1. Por exemplo, para $ABCD = 0011$ temos para as saídas: $Y_3 = 1$ e todas as demais saídas $Y_k = 0$, com $k \neq 3$. Note que o subscrito da saída cujo nível lógico é 1 corresponde ao valor decimal do *Nibble* codificado em BCD nas entradas $ABCD$.

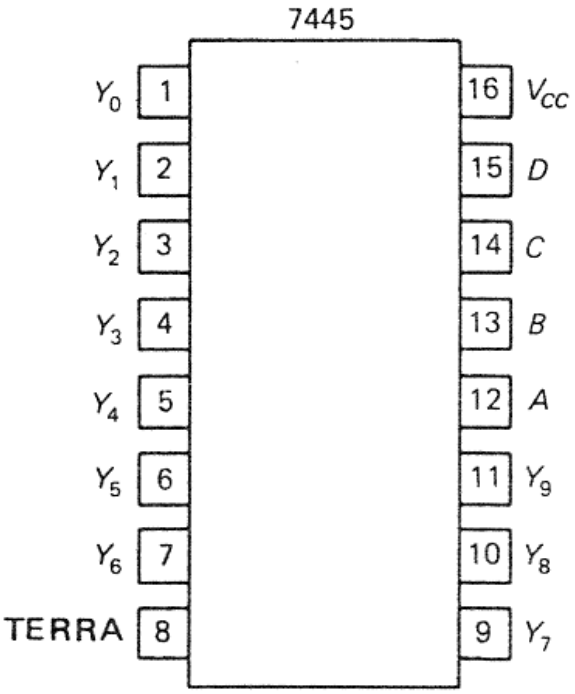


Figura 10: Diagrama de pinagem do circuito integrado TTL – 7445 comercialmente disponível para a implementação da função Decodificador BCD-para-Decimal.

	Entradas				Saídas									
Nº	*A*	*B*	*C*	*D*	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
Não-válida	H	L	H	L	H	H	H	H	H	H	H	H	H	H
	H	L	H	H	H	H	H	H	H	H	H	H	H	H
	H	H	L	L	H	H	H	H	H	H	H	H	H	H
	H	H	L	H	H	H	H	H	H	H	H	H	H	H
	H	H	H	L	H	H	H	H	H	H	H	H	H	H
	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Tabela 4: Tabela Verdade para o TTL – 7445. Note que a saída ativa é precedida de uma porta NOT.

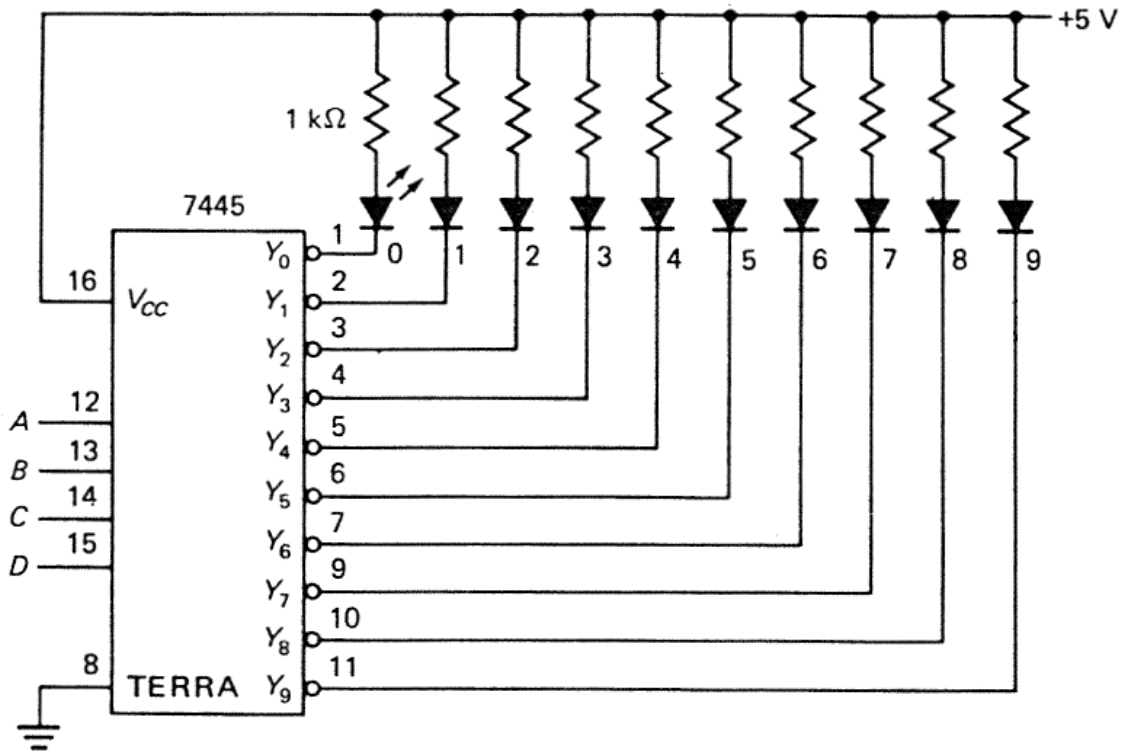


Figura 11: Exemplo de utilização do TTL – 7445 como Decodificador BCD-para-Decimal. O valor decimal do $ABCD$ na entrada do decodificador é indicado pelo LED que está aceso conectado à saída correspondente.

3.1 Codificador Decimal-para-BCD

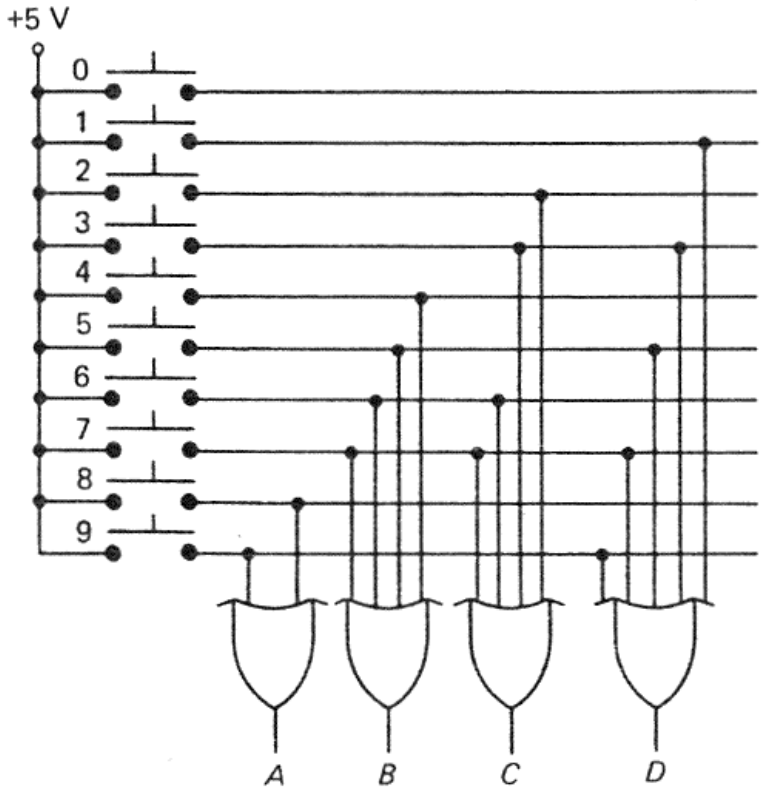


Figura 12: Diagrama interno de um Codificador Decimal-para-BCD. As chaves são do tipo *pushbutton* (como no teclado de um computador). Por exemplo, quando o *pushbutton* 3 é pressionado as portas OR cujas saídas são C e D têm entradas cujo nível lógico é 1, resultando $ABCD = 0011$.

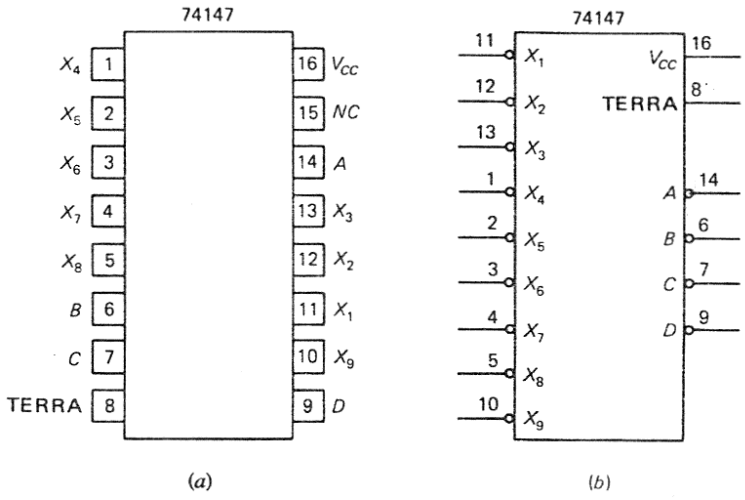


Figura 13: Diagrama (a) de pinagem e (b) diagrama lógico do circuito integrado TTL – 74147 comercialmente disponível para a implementação da função Codificador Decimal-para-BCD.

Entradas									Saídas			
X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	A	B	C	D
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

Tabela 5: Tabela Verdade para o TTL – 74147.

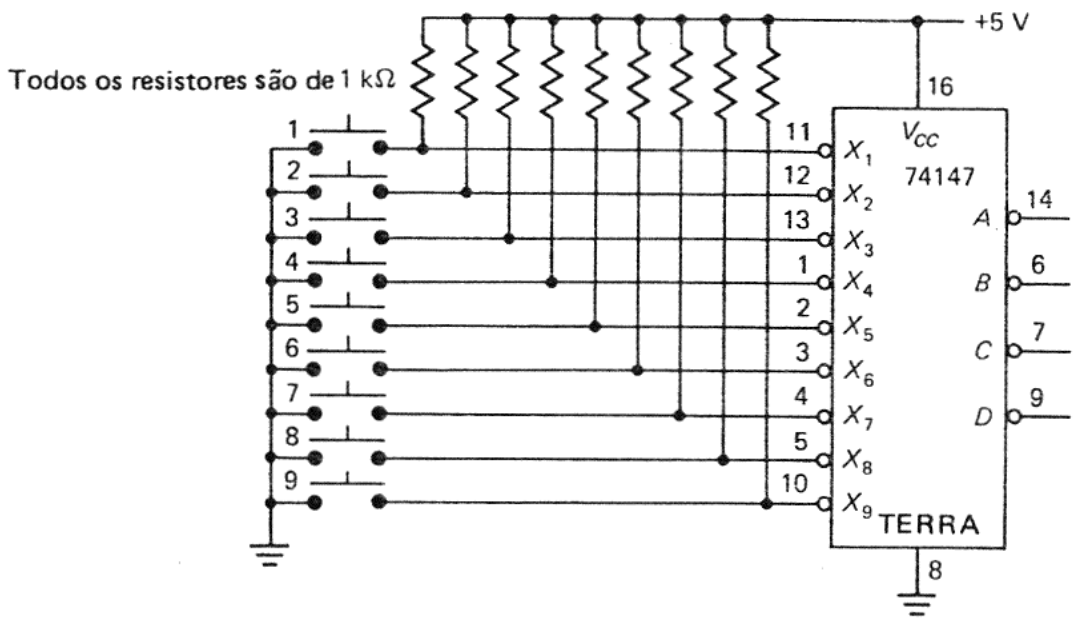


Figura 14: Exemplo de utilização do TTL – 74147 como Codificador Decimal-para-BCD. Quando nenhuma chave é pressionada todas as entradas *X* têm valor lógico 1 e todas as saídas têm valor lógico 1. Quando a chave X_9 é pressionada ($X_9 = 0$) temos na saída $ABCD = 0110$, que é equivalente a 9 se complementarmos os bits. Quando a chave X_8 é pressionada ($X_8 = 0$) temos na saída $ABCD = 0111$, que é equivalente a 8 se complementarmos os bits.

- Note da Tabela 5 que a entrada *X* ativa (ativa → valor lógico 0) de ordem mais alta tem prioridade sobre as demais. Se todas as entradas *X* têm valor lógico 0, a de ordem mais alta (X_9) é codificada com prioridade resultando $ABCD = 0110$, cujo complemento identifica a entrada ativa de ordem mais alta. Se X_9=1 e X_8=0 então a entrada *X* ativa de ordem mais alta é X_8 e é codificada com prioridade resultando $ABCD = 0111$, cujo complemento identifica a entrada ativa de ordem mais alta. Devido a este comportamento o TTL – 74147 é também conhecido como **Codificador de Prioridade**.

4 Decodificador Gray-para-Binário

- O Código Gray é um código digital com a propriedade de que duas palavras-código consecutivas diferem apenas de um bit.

- O Código Gray é um código que se enquadra na classe de **Códigos Refletidos**, enquadramento devido ao algoritmo de construção do mesmo. Por exemplo, a Tabela 6 mostra a construção por quantificação-reflexão do Código Gray para 4 bits:

Quantificação	Reflexão	Quantificação	Reflexão	Quantificação	Reflexão	Quantificação
0	0	00	00	000	000	0000
1	1	01	01	001	001	0001
	1	11	11	011	011	0011
	0	10	10	010	010	0010
			10	110	110	0110
			11	111	111	0111
			01	101	101	0101
			00	100	100	0100
					100	1100
					101	1101
					111	1111
					110	1110
					010	1010
					011	1011
					001	1001
					000	1000

Tabela 6: Algoritmo de construção do Código Gray de 4 bits.

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

Tabela 7: Tabela de conversão Decimal-Binário-Gray de 4 bits.

- Representando o *nibble* do Código Gray da Tabela 7 por $G_3G_2G_1G_0$ e o *nibble* do Código Binário por $B_3B_2B_1B_0$ temos:

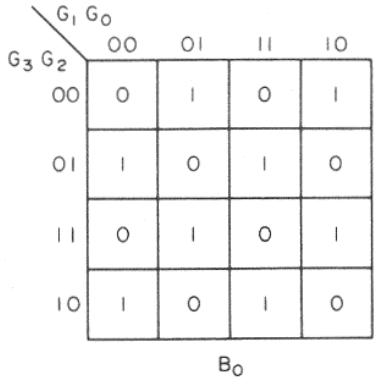
	Gray				Binário			
	G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

Tabela 8: Identificação dos *nibbles* dos códigos Gray e Binário de 4 bits.

- A Figura 15 mostra os mapas K para as funções lógicas que expressam B_0, B_1, B_2 e B_3 em função de $G_3G_2G_1G_0$, a partir da Tabela 8:

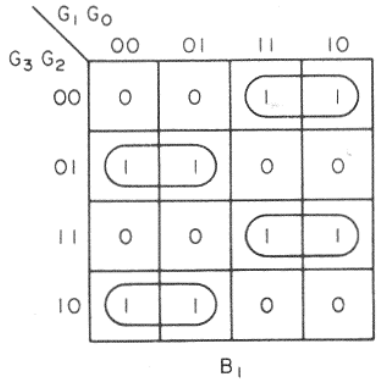
Mapa de Karnaugh para B_0

$$B_0 = \bar{G}_2\bar{G}_3(G_0\bar{G}_1 + \bar{G}_0G_1) + G_2\bar{G}_3(G_0G_1 + \bar{G}_0\bar{G}_1) + G_2G_3(G_0\bar{G}_1 + \bar{G}_0G_1) + \bar{G}_2G_3(G_0G_1 + \bar{G}_0\bar{G}_1)$$
$$= (G_0\bar{G}_1 + \bar{G}_0G_1)(G_2G_3 + \bar{G}_2\bar{G}_3) + (G_0G_1 + \bar{G}_0\bar{G}_1)(G_2\bar{G}_3 + \bar{G}_2G_3)$$
$$= (G_0 \oplus G_1)\overline{(G_2 \oplus G_3)} + \overline{(G_0 \oplus G_1)}(G_2 \oplus G_3)$$
$$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$$



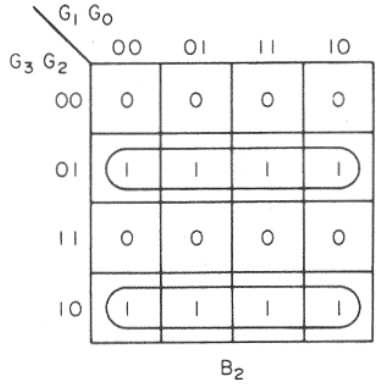
Mapa de Karnaugh para B_1

$$B_1 = G_1\bar{G}_2\bar{G}_3 + \bar{G}_1G_2\bar{G}_3 + G_1G_2G_3 + \bar{G}_1\bar{G}_2G_3$$
$$= G_1(G_2G_3 + \bar{G}_2\bar{G}_3) + \bar{G}_1(G_2\bar{G}_3 + \bar{G}_2G_3)$$
$$= G_1\overline{(G_2 \oplus G_3)} + \bar{G}_1(G_2 \oplus G_3)$$
$$B_1 = G_1 \oplus G_2 \oplus G_3$$



Mapa de Karnaugh para B_2

$$B_2 = G_2\bar{G}_3 + \bar{G}_2G_3$$
$$B_2 = G_2 \oplus G_3$$



Mapa de Karnaugh para B_3

$$B_3 = G_3$$

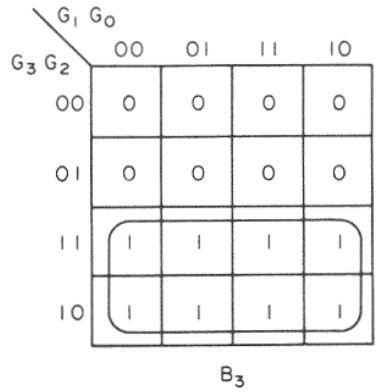


Figura 15: Mapas K para as funções lógicas que expressam B_0, B_1, B_2 e B_3 em função de $G_3G_2G_1G_0$.

- Da Figura 15 temos que as funções lógicas minimizadas que expressam B_0, B_1, B_2 e B_3 em função de $G_3G_2G_1G_0$ são:

$$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$$

$$B_1 = G_1 \oplus G_2 \oplus G_3$$

$$B_2 = G_2 \oplus G_3$$

$$B_3 = G_3$$

- Generalizando para um Código Gray de N bits, podemos escrever que $B_n = G_n \oplus G_{(n+1)} \oplus G_{(n+1)} \oplus \ldots \oplus G_{N-1}$, o que sugere o circuito lógico mostrado na Figura 16:

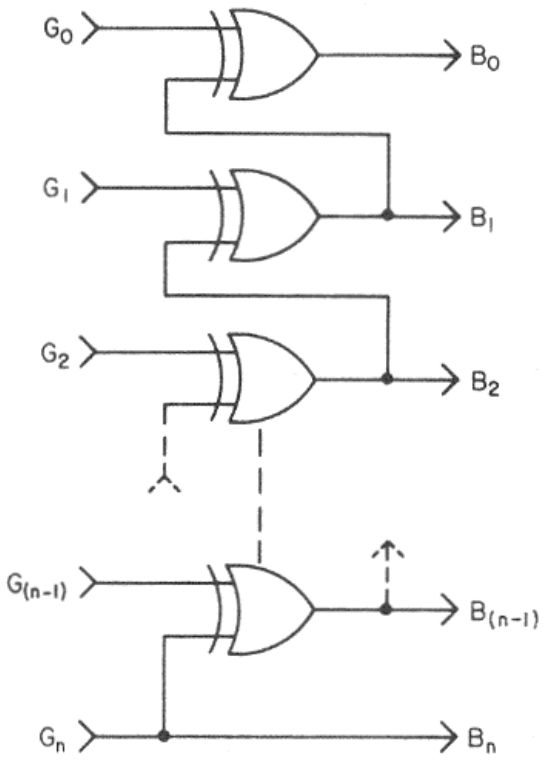


Figura 16: Conversor Gray-para-Binário.

4.1 Decodificador Binário-para-Gray

● A Figura 17 mostra os mapas K para as funções lógicas que expressam G_0, G_1, G_2 e G_3 em função de $B_3B_2B_1B_0$, tendo como ponto de partida a Tabela 8:

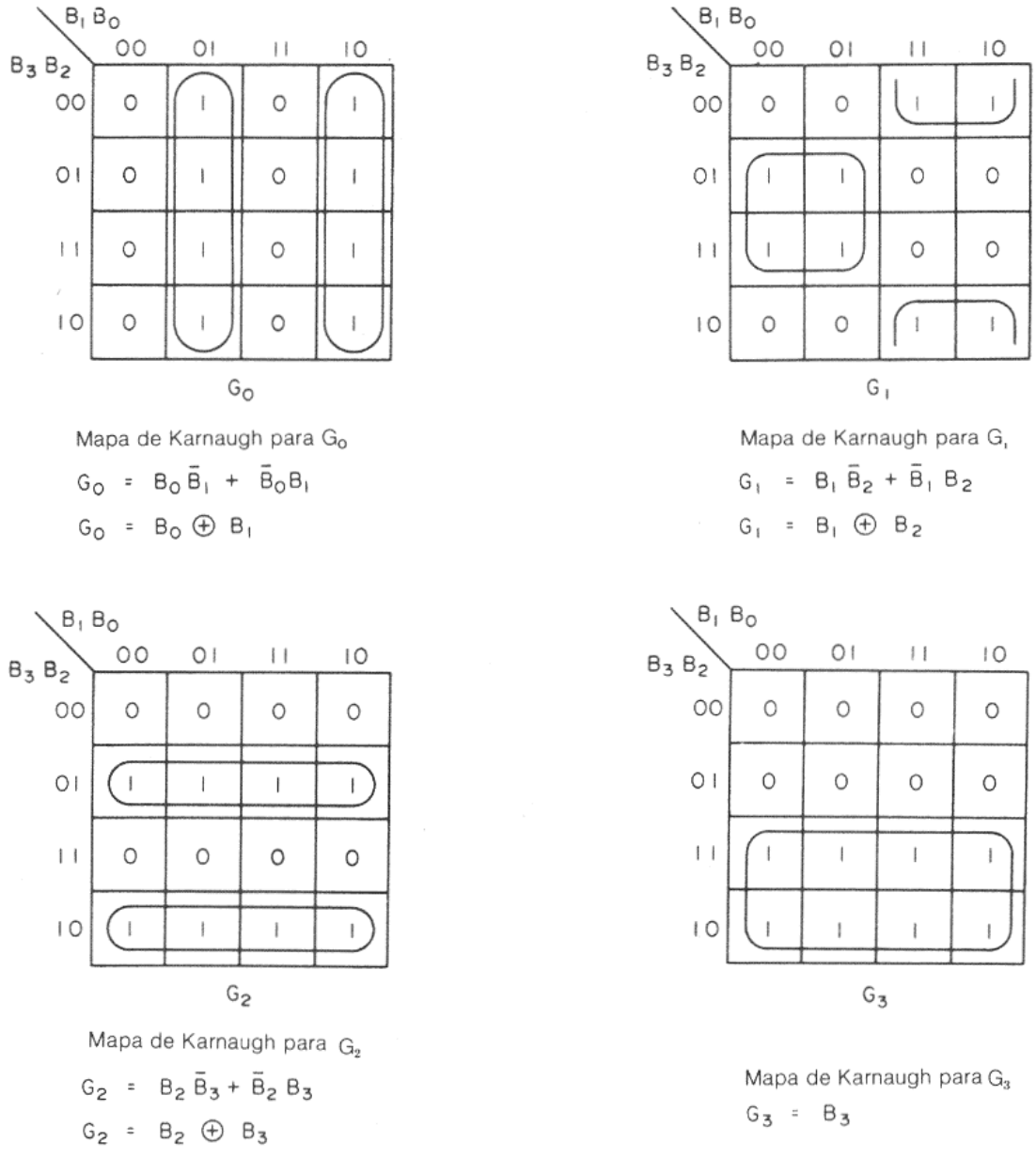


Figura 17: Mapas K para as funções lógicas que expressam G_0, G_1, G_2 e G_3 em função de $B_3B_2B_1B_0$.

● Da Figura 17 temos que as funções lógicas minimizadas que expressam G_0, G_1, G_2 e G_3 em função de $B_3B_2B_1B_0$ são:

$$G_0 = B_0 \oplus B_1$$

$$G_1 = B_1 \oplus B_2$$

$$G_2 = B_2 \oplus B_3$$

$$G_3 = B_3$$

- Generalizando para um Código Gray de N bits, podemos escrever que $G_n = B_n \oplus B_{(n+1)}$, sendo $n+1 \leq N-1$. Isto sugere o circuito lógico mostrado na Figura 18:

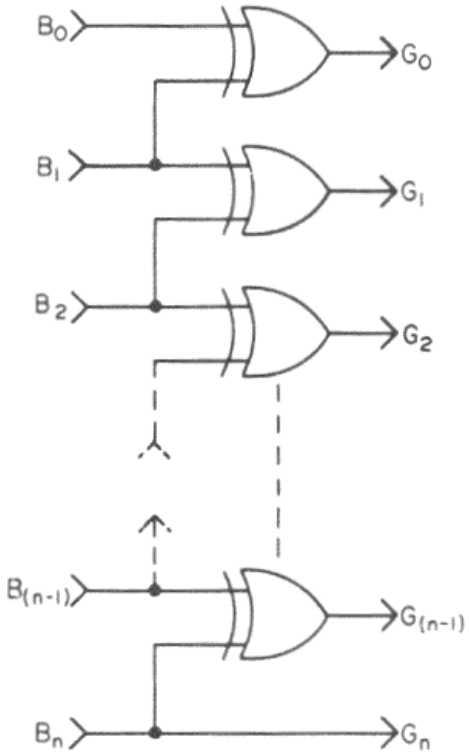


Figura 18: Conversor Binário-para-Gray.

5 Multiplexadores

- Um Multiplexador é um circuito digital com *n* entradas e uma única saída, e serve para selecionar qual sinal, dentre os *n* sinais de entrada, deve ser roteado até a saída.

- Uma palavra binária de controle seleciona qual das *n* entradas é conectada à saída conforme mostram as Figuras 19 e 20.

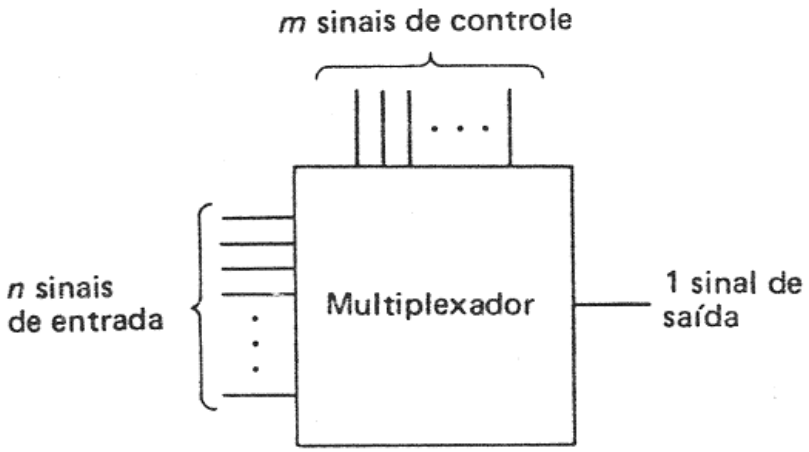


Figura 19: Diagrama geral de um multiplexador. Note que o número *m* de bits da palavra binária de controle deve ser tal que $2^m=n$.

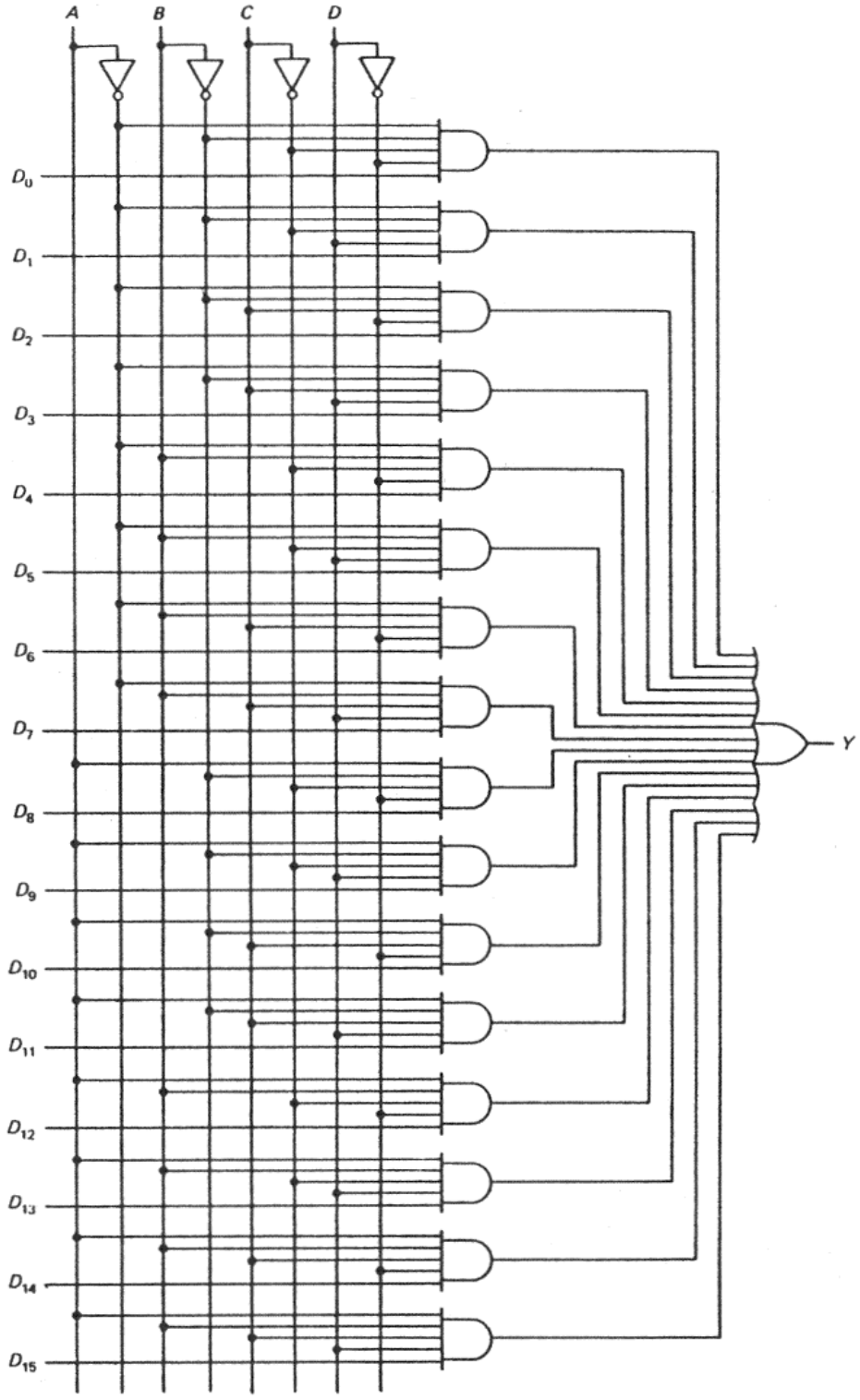


Figura 20: Diagrama interno um multiplexador 16-para-1. O número de bits da palavra binária de controle é *m*=4 (*nibble*) e, portanto, o multiplexador pode selecionar uma das *n* =2^m=16 entradas. Por exemplo, se $ABCD = 0111$ então o sinal digital que está sendo aplicado na entrada D_7 é roteado até a saída Y.

● A Figura 21 e a Tabela 9 descrevem um multiplexador 16-para-1 disponível comercialmente na forma de circuito integrado da família TTL.

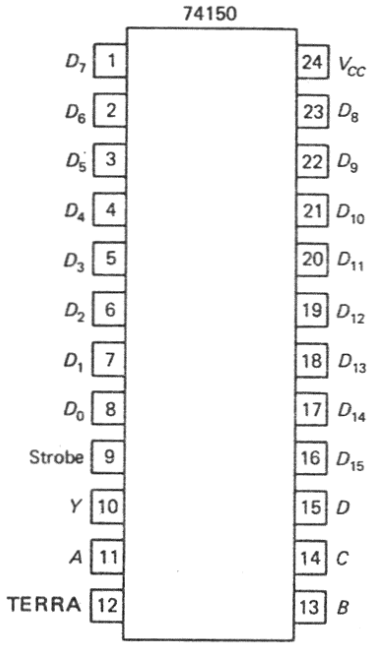


Figura 21: Diagrama de pinos do TTL 74150 – circuito integrado comercialmente disponível para a implementação da função MUX 16-para-1.

Strobe	*A B C D*	*Y*
L	L L L L	$\overline{D}_0$
L	L L L H	$\overline{D}_1$
L	L L H L	$\overline{D}_2$
L	L L H H	$\overline{D}_3$
L	L H L L	$\overline{D}_4$
L	L H L H	$\overline{D}_5$
L	L H H L	$\overline{D}_6$
L	L H H H	$\overline{D}_7$
L	H L L L	$\overline{D}_8$
L	H L L H	$\overline{D}_9$
L	H L H L	$\overline{D}_{10}$
L	H L H H	$\overline{D}_{11}$
L	H H L L	$\overline{D}_{12}$
L	H H L H	$\overline{D}_{13}$
L	H H H L	$\overline{D}_{14}$
L	H H H H	$\overline{D}_{15}$
H	X X X X	H

Tabela 9: Tabela-Verdade para o TTL 74150. Note que o sinal de entrada roteado à saída Y é submetido a uma porta NOT. Note também que o sinal de *strobe* (pino 9 na Figura 21) é um sinal ativo-baixo que serve para ativar/desativar o multiplexador.

5.1 Implementação de Funções Lógicas utilizando um MUX

● Vamos supor que queremos implementar a função lógica mostrada na Tabela 10 a seguir.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Tabela 10: Tabela-Verdade de uma função lógica hipotética a ser implementada.

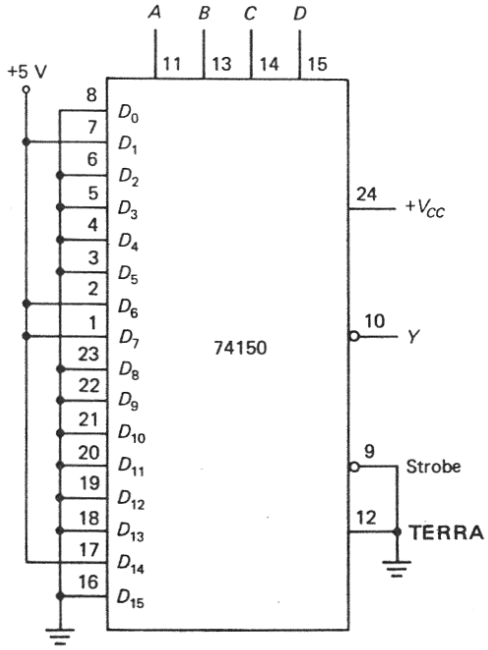


Figura 22: Implementação da função lógica descrita na Tabela 10 utilizando o TTL 74150. O procedimento geral para a implementação de qualquer função lógica de 4 variáveis é o seguinte: Sempre que o *nibble* $ABCD$ resultar em uma saída $Y = 1$ na Tabela-Verdade (ver Tabela 10), conecta-se à terra o pino de dado D_k, sendo k o valor decimal correspondente ao *nibble* $ABCD$. O pino de dado de índice k cujo *nibble* correspondente na Tabela-Verdade refere-se à saídas $Y = 0$ é conectado à $+5V$.

- Por exemplo, na Figura 22, se $ABCD = 0000$ então a entrada $D_0 = 0$ é conectada à saída através de uma porta NOT, de modo que $Y = 1$. Se $ABCD = 0111$ então a entrada $D_7 = 1$ é conectada à saída através de uma porta NOT, de modo que $Y = 1$. Todas as demais linhas da Tabela-Verdade (Tabela 10) podem ser obtidas através de procedimento semelhante.

5.2 Multiplexadores de Nibble

- Em muitas situações práticas precisamos selecionar um entre dois *nibbles* de entrada, conforme mostram as Figuras 22 e 23.

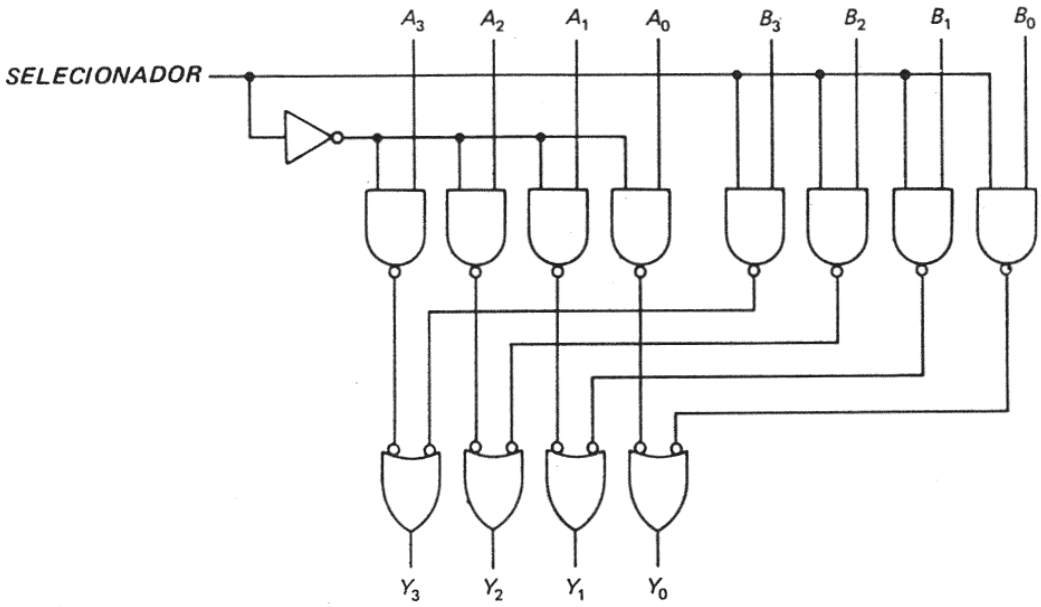


Figura 23: Diagrama interno de um multiplexador de *nibble*. Quando $SELECIONADOR = 0$ obtemos $Y_3Y_2Y_1Y_0 = A_3A_2A_1A_0$ e quando $SELECIONADOR = 1$ obtemos $Y_3Y_2Y_1Y_0 = B_3B_2B_1B_0$.

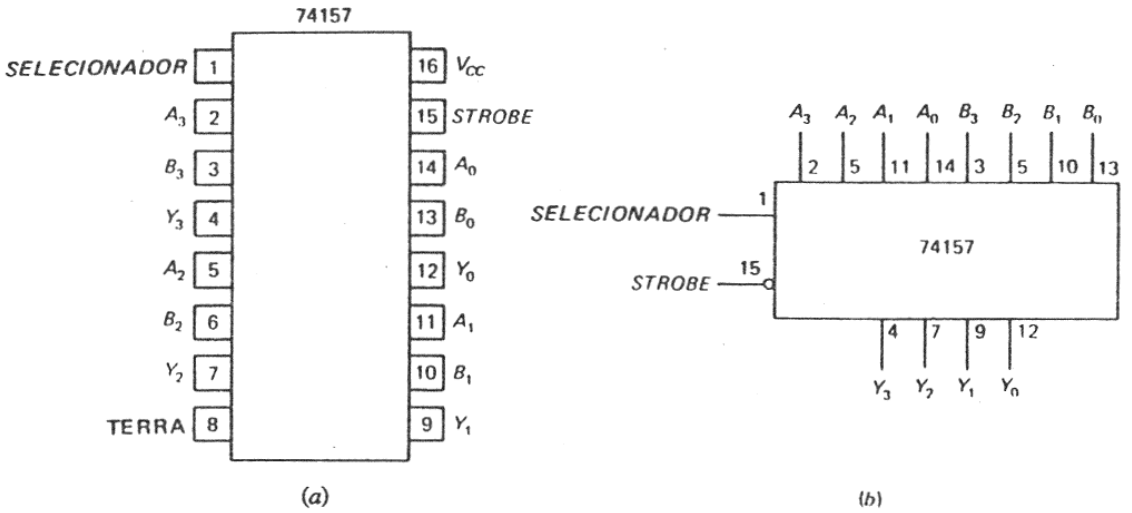


Figura 24: Diagrama de pinagem do circuito integrado TTL – 74157 comercialmente disponível para a implementação da função Multiplexador de *Nibble*. O diagrama interno é o mostrado na Figura 23. O pino de *strobe* (pino 15) desativa o multiplexador quando encontra-se em nível lógico 1 e ativa o multiplexador quando encontra-se em nível lógico 0.

6 Demultiplexadores

- Um Demultiplexador é um circuito digital com uma única entrada e *n* saídas, e serve para selecionar à qual saída, dentre as *n* saídas, deve ser roteado o sinal de entrada.

- Uma palavra binária de controle seleciona à qual das *n* saídas é conectada o sinal de entrada conforme mostram as Figuras 25 e 26.

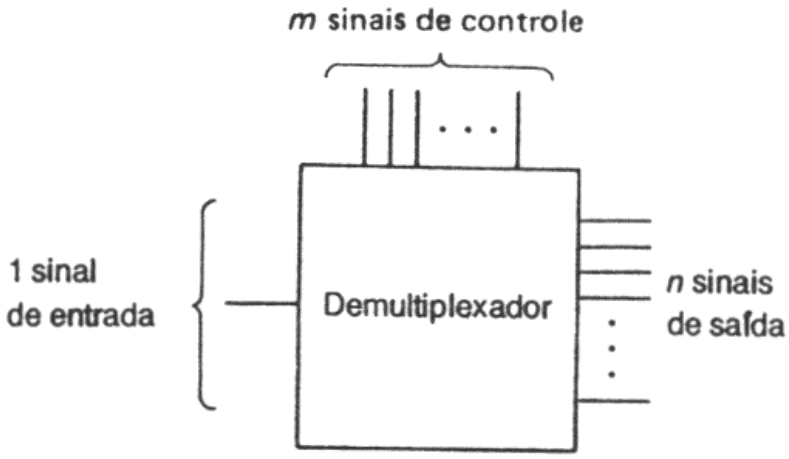


Figura 25: Diagrama geral de um demultiplexador. Note que o número *m* de bits da palavra binária de controle deve ser tal que $2^m=n$.

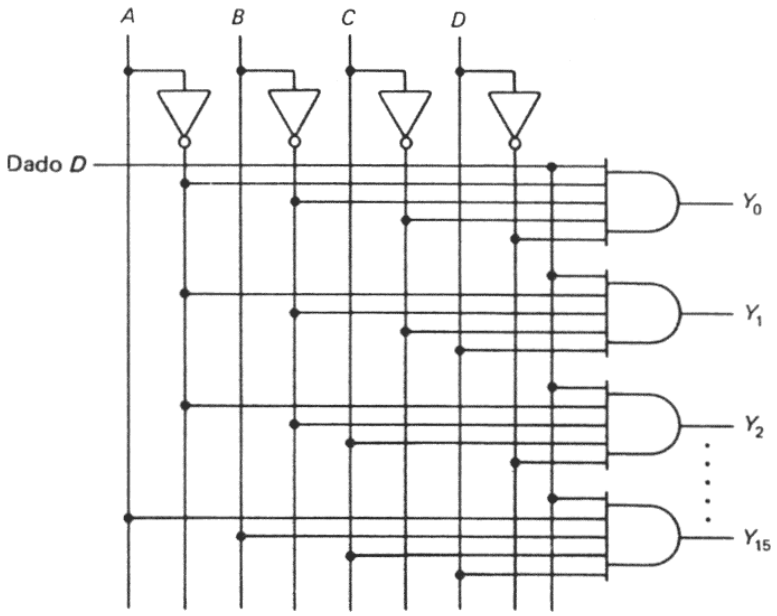


Figura 26: Diagrama interno um demultiplexador 1-para-16. O número de bits da palavra binária de controle é *m*=4 (*nibble*) e, portanto, o demultiplexador pode selecionar uma das *n* =2^m=16 saídas. Por exemplo, se $ABCD = 0111$ então o sinal digital que está sendo aplicado na entrada D é roteado até a saída Y_7.

● As Figuras 27 e 28 e a Tabela 11 descrevem um demultiplexador 1-para-16 disponível comercialmente na forma de circuito integrado da família TTL.

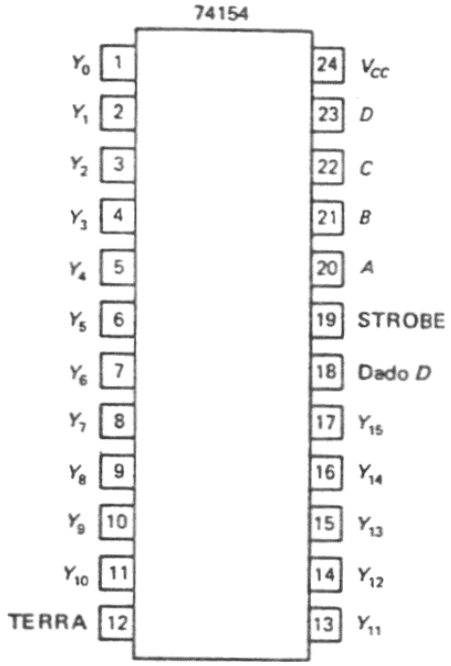


Figura 27: Diagrama de pinos do TTL 74154 – circuito integrado comercialmente disponível para a implementação da função DEMUX 1-para-16.

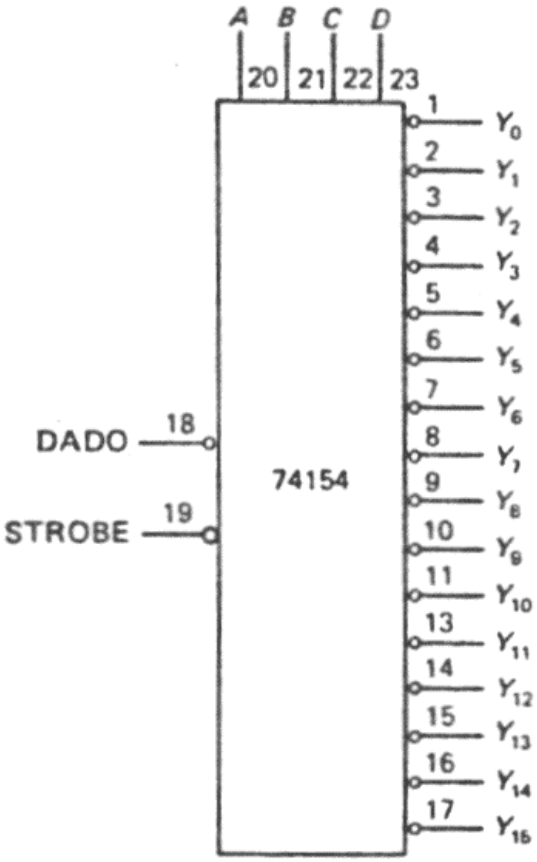


Figura 28: Diagrama lógico do TTL 74154.

STROBE	DADO	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Tabela 11: Tabela-Verdade para o TTL 74154.

7 Decodificadores

- Em seções anteriores deste capítulo já estudamos diversos decodificadores específicos (BCD-para-Decimal, Binário-para-Gray, etc.). Vimos que eles servem como "tradutores" entre diversos os formatos de representação numérica da informação a ser processada.

- Nesta seção estudaremos os decodificadores sob um ponto de vista genérico.

- Um **decodificador** é similar a um **demultiplexador** , a única diferença é que a entrada de dado do demultiplexador não existe no decodificador, conforme podemos concluir comparando a Figura 26 com a Figura 29:

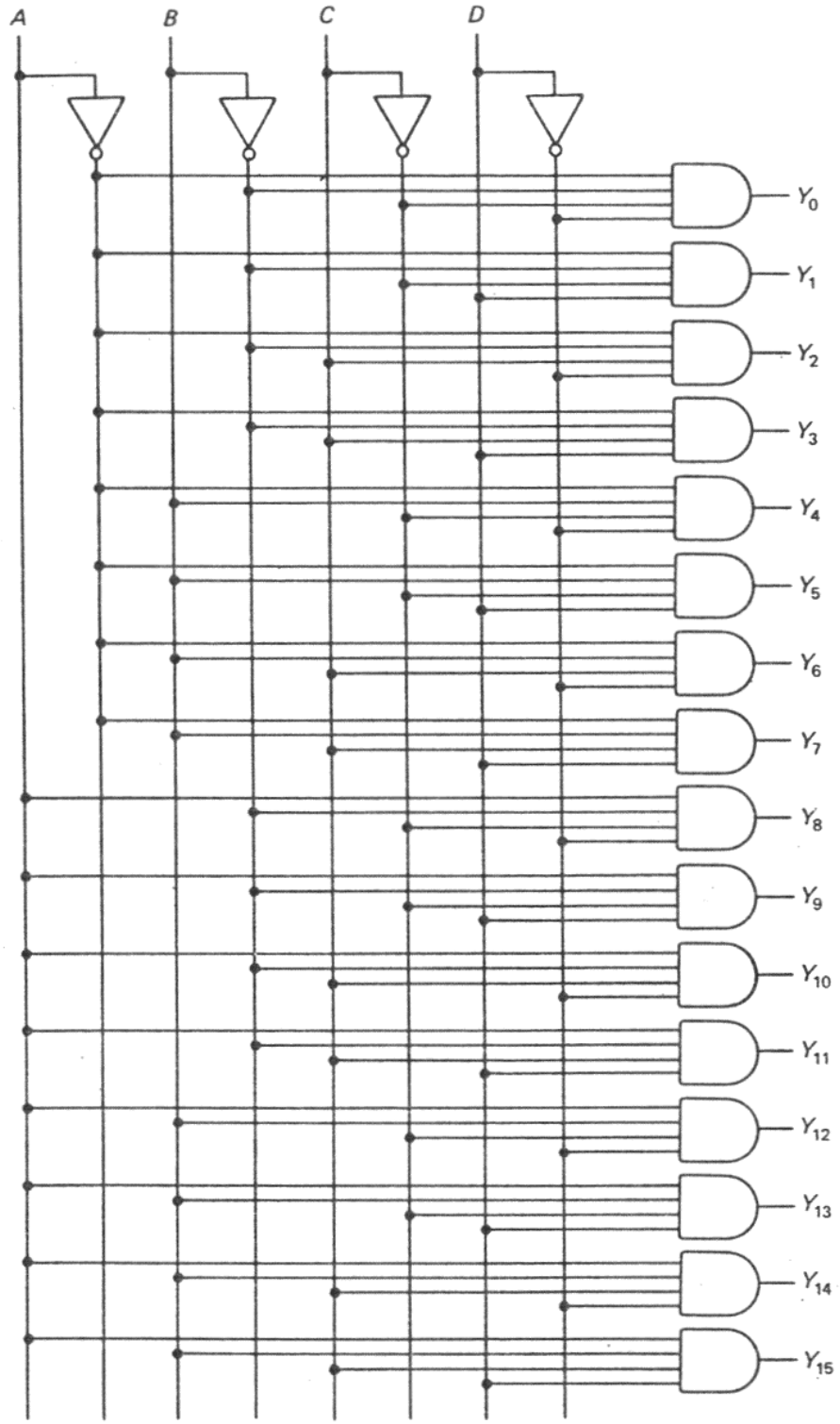


Figura 29: Diagrama interno um decodificador 1-de-16. A designação 1-de-16 decorre do fato de somente uma das 16 saídas assumir o nível lógico 1 em função do valor da palavra binária de controle. Por exemplo, se $ABCD = 0111$ então $Y_7 = 1$.

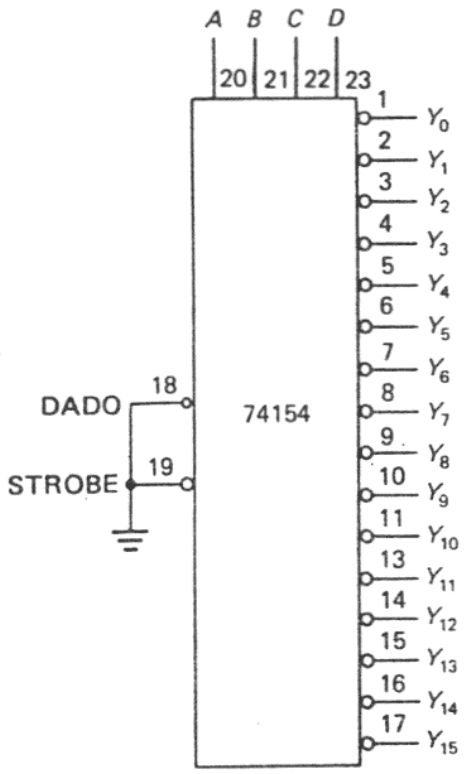


Figura 30: Utilização do TTL 74154 como decodificador 1-de-16. Já estudamos este circuito integrado como DEMUX 1-para-16 (ver Figura 27). Note que para converter o 74154 de demultiplexador para decodificador basta aterrar a entrada de dados (pino 18).

8 Comparadores

- Um **comparador** indica a igualdade entre duas palavras binárias A e B, isto é, indica se A=B.

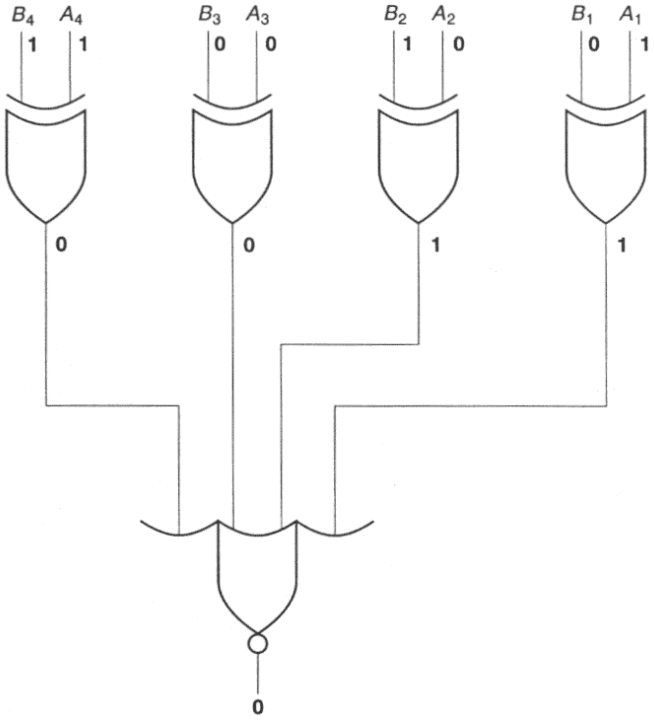


Figura 31: Diagrama interno de um comparador de 4 bits. É mostrado a comparação entre os *nibbles* 1010 e 1001, resultando o valor lógico 0 na saída, o que significa que os *nibbles* não são iguais.

- Um **comparador de magnitude** indica se A=B, A<B ou A>B.

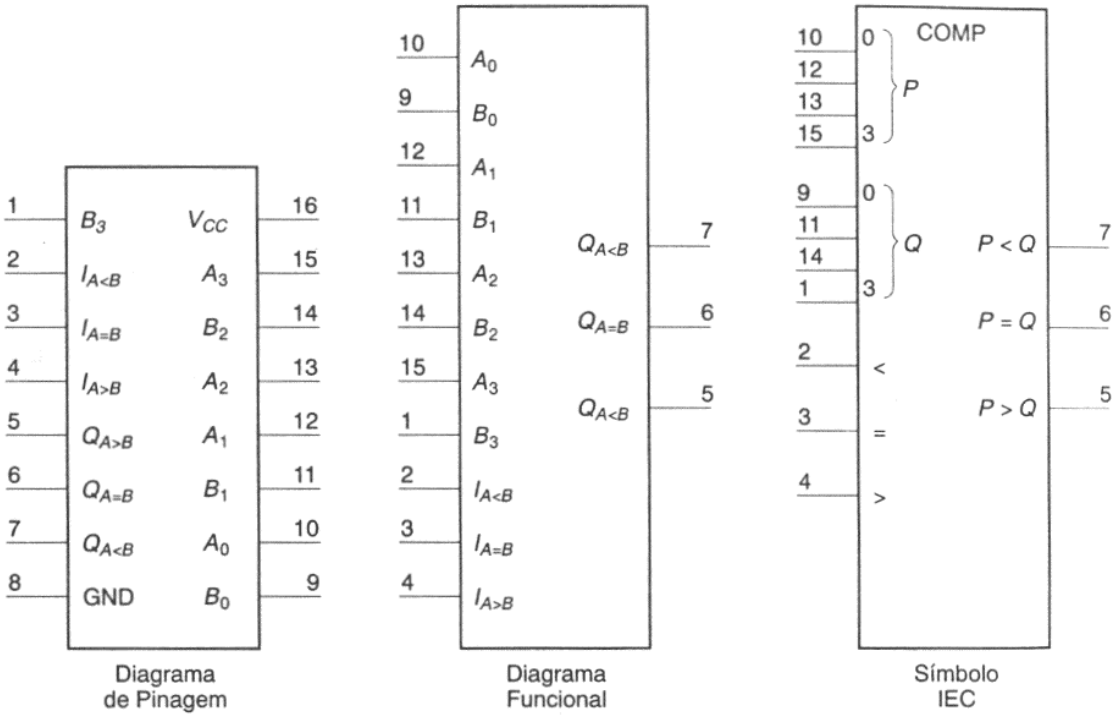


Figura 32: Circuito Integrado (CI) TTL 7485, comercialmente disponível para a implementação da função comparador de magnitude de 4 bits. Os pinos 2, 3 e 4 são entradas para conexão em cascata de 2 CIs e são utilizados quando se deseja comparar palavras binárias com mais de 4 bits.

- O TTL 7485 compara duas palavras binárias de 4 bits A e B colocando uma das saídas $Q_{A<B}$, $Q_{A=B}$, $Q_{A>B}$ em nível lógico 1 respectivamente quando $A<B$, $A=B$ e $A>B$.

Entradas de comparação				**Entradas de conexão em cascata**			**Saídas**		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A > B$	$A < B$	$A = B$	$A > B$	$A < B$	$A = B$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H

H = Nível ALTO (HIGH) L = Nível BAIXO (LOW) *X* = Irrelevante

Tabela 12: Tabela-Verdade para o TTL 7485.

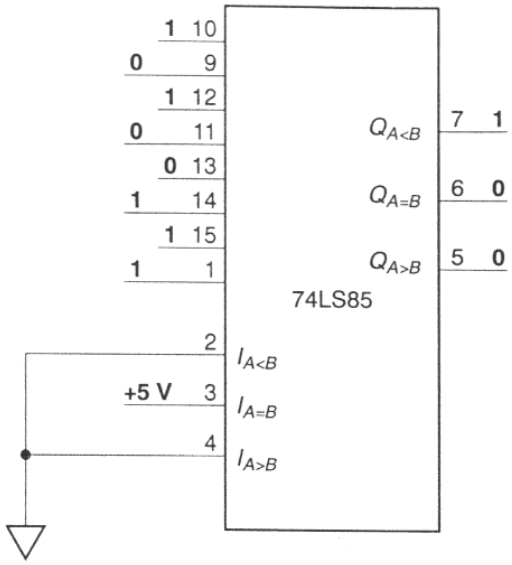


Figura 33: Exemplo de utilização do CI TTL 7485 como comparador de magnitude de 4 bits. As entradas são $A = 1011 = 11_{(\text{dec})}$ e $B = 1100 = 12_{(\text{dec})}$, de modo que $A < B$, e, portanto, a saída $Q_{A<B} = 1$.

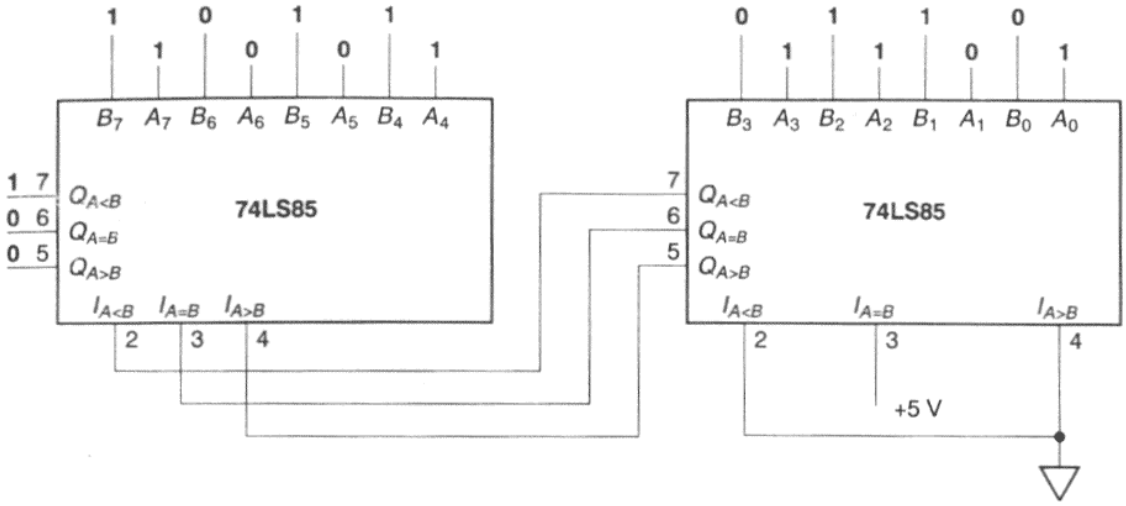


Figura 34: Conexão em cascata entre 2 CIs TTL 7485 de modo a implementar um comparador de magnitude para palavras binárias de 8 bits. As entradas são $A = 10011101 = 157_{(\text{dec})}$ e $B = 10110110 = 182_{(\text{dec})}$, de modo que $A < B$, e, portanto, a saída $Q_{A<B} = 1$.

- Note na Figura 34 que se os *nibbles* mais significativos (CI à esquerda) em A e B são iguais, então o resultado é determinado pelas entradas para conexão em cascata deste CI, as quais recebem o resultado da comparação entre os *nibbles* menos significativos (CI à direita) em A e B.

9 *Parity Check*

- Vimos na Seção 5 do Capítulo II que em muitas situações práticas de controle digital de processos industriais as palavras binárias constituem **Instruções de Comando** que devem ser enviadas por longas distâncias através de um **Canal de Transmissão** (cabo coaxial, fibra ótica, etc...) antes de chegarem ao destino onde a instrução desencadeará uma ação específica no processo controlado.

- Foi também discutido que **sempre** que palavras binárias são enviadas através de um Canal de Transmissão estas ficam sujeitas a algum tipo de **Interferência** (ruído aleatório, interferência de outras fontes de energia, interferência intersimbólica, etc...), gerando **Erros de Transmissão** que devem ser corrigidos ou pelo menos detectados.

- Quando o objetivo é somente detectar um erro de transmissão, sem precisar corrigir o erro, a operação ***parity check*** é uma possível solução ao problema de detecção de erros de transmissão.

- A operação ***parity check*** consiste em acrescentar um bit adicional a uma palavra binária a ser transmitida através do canal de transmissão, bit adicional que define a **paridade** da palavra binária transmitida. O circuito receptor no ponto remoto do processo controlado testa a paridade da palavra binária recebida verificando que houve erro quando a paridade não é mantida.

- Por exemplo, a palavra binária de 8 bits dada por 11001111 tem **paridade par** porque contém um número par de bits com valor lógico 1. Por outro lado a palavra binária de 16 bits definida por 1010110010001100 tem **paridade ímpar** porque contém um número ímpar de bits com valor lógico 1.

- O teste de paridade (***parity check***) é usualmente realizado através de uma porta XOR, conforme mostra a Figura 35:

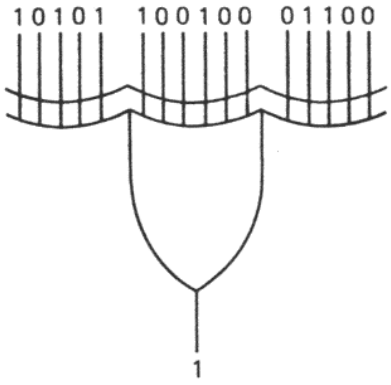


Figura 35: Teste de paridade da palavra binária 1010110010001100. A saída da porta XOR resulta em nível lógico 1, significando que esta palavra possui paridade ímpar. Se a palavra aplicada na entrada da porta XOR tivesse paridade par, então a saída da porta resultaria em nível lógico 0.

9.1 Gerador de Paridade

- Para que a operação ***parity check*** possa ser efetuada no receptor é necessário acrescentar um bit adicional à palavra binária a ser transmitida através do canal de transmissão, bit adicional que define a **paridade** da palavra binária transmitida. A Figura 36 mostra um circuito **gerador de paridade** utilizado no transmissor da palavra binária.

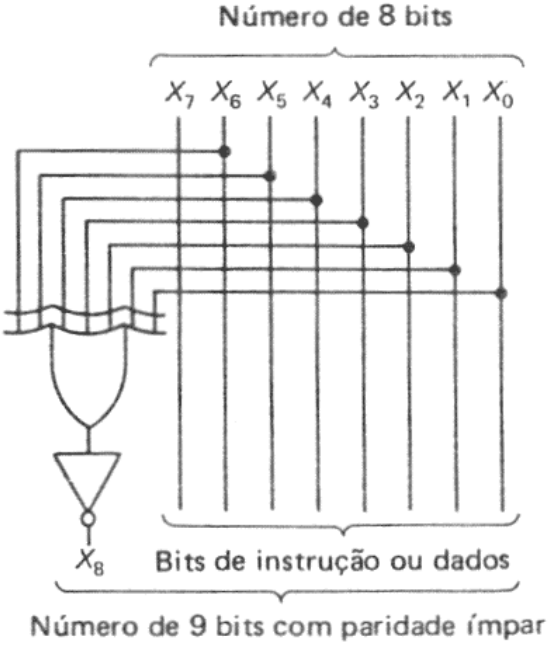


Figura 36: Circuito gerador de paridade. Se a entrada $X_7X_6X_5X_4X_3X_2X_1X_0$ tem paridade par então $X_8 = 1$, de modo que a palavra transmitida $X_8X_7X_6X_5X_4X_3X_2X_1X_0$ terá paridade ímpar. Por outro lado, se a entrada $X_7X_6X_5X_4X_3X_2X_1X_0$ tem paridade ímpar então $X_8 = 0$, de modo que a palavra transmitida $X_8X_7X_6X_5X_4X_3X_2X_1X_0$ **sempre** terá paridade ímpar. Se quisermos detectar erros com base em palavras transmitidas de paridade par, basta eliminarmos a porta NOT.

10 Memórias ROM (*Read Only Memory*)

- Uma ROM é uma memória de **apenas leitura**. Ela é usualmente implementada em um CI, podendo armazenar milhares de palavras binárias que representam instruções e/ou dados para um microcontrolador/microprocessador.

- Algumas ROMs de menor capacidade são também utilizadas para implementar tabelas-verdade. Em outras palavras, podemos usar uma ROM ao invés de portas lógicas para gerar qualquer função booleana. A Figura 37 mostra o diagrama de uma ROM constituída por diodos, com endereço de acesso definido pela posição da chave rotativa.

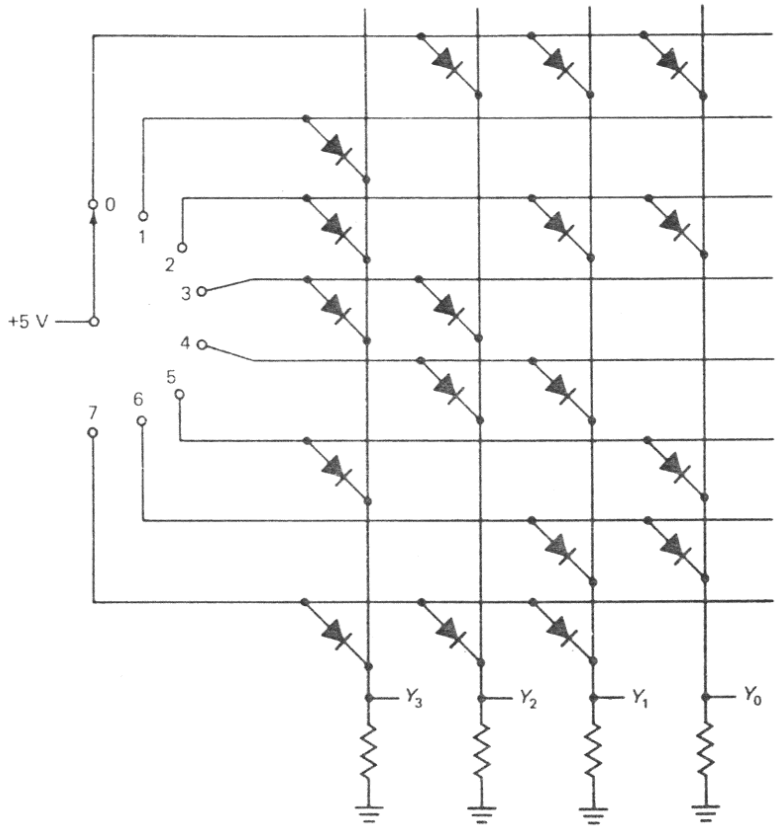


Figura 37: ROM a diodo, armazenando nos endereços 0 a 7 as palavras binárias de 4 bits (*nibbles*) mostradas na Tabela 13. Cada endereço corresponde a uma posição da chave rotativa. Por exemplo, quando a chave rotativa encontra-se na posição 3 (endereço 3) obtém-se $Y_3Y_2Y_1Y_0 = 1100$.

Endereço	Nibble
0	0111
1	1000
2	1011
3	1100
4	0110
5	1001
6	0011
7	1110

Tabela 13: Tabela-Verdade da ROM da Figura 37.

● Na realidade a chave rotativa aqui empregada é apenas um recurso didático para ilustrar a operação de uma ROM. Na prática, nenhuma chave rotativa é utilizada para seleção de endereço, mas sim um decodificador, conforme mostra a Figura 38.

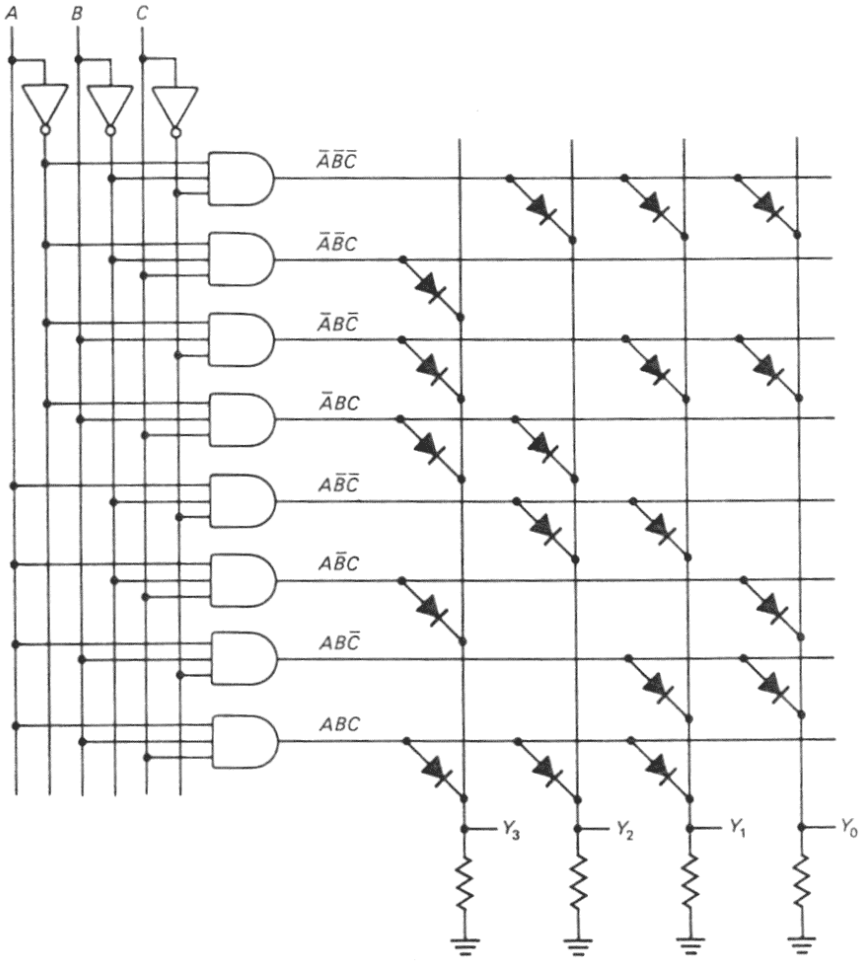


Figura 38: ROM a diodo da Figura 37, com a chave rotativa de 8 posições substituída por um decodificador 1-de-8. Por exemplo, quando $ABC = 011$ obtém-se $Y_3Y_2Y_1Y_0 = 1100$ (ver Tabela 14).

A	B	C	Y_3	Y_2	Y_1	Y_0
0	0	0	0	1	1	1
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	0	0	1	1
1	1	1	1	1	1	0

Tabela 14: Tabela-Verdade da ROM da Figura 38.

- Uma PROM (*Programmable ROM*) é uma ROM programável. Uma PROM vem de fabrica com todos os diodos implementados (a PROM gera todas as palavras binárias da tabela verdade com nível lógico 1 em seus bits). O usuário, através de um dispositivo **programador de PROM**, queima os microfusíveis em série com cada diodo que deva representa um bit de valor lógico 0 na palavra endereçada.

- Uma ROM é especificada basicamente pelo número de endereços e o número de bits na palavra binária armazenada em cada endereço. Por exemplo, uma ROM 2048×8 armazena 2048 palavras binárias de 8 bits (8 bits = 1 byte), e, portanto, armazena um total de 16384 bits.

- Existem vários tipos de ROM além da PROM, como, por exemplo, a UV-EPROM (*Ultra Violet erasable PROM*) e a EEPROM (*Eletrically Erasable PROM*). Estes tipos adicionais serão abordados em capítulo posterior específico ao estudo de memórias.