Capítulo 3

Circuitos combinatórios

Circuitos combinatórios são aqueles cujas saídas dependem apenas da combinação dos valores das entradas em determinado instante. Neste capítulo serão vistos os principais circuitos combinatórios utilizados em sistemas digitais: codificadores, decodificadores, multiplexadores, demultiplexadores e circuitos aritméticos.

3.1 Codificadores/decodificadores

Os sistemas digitais trabalham com informações representadas por níveis lógicos zeros (0) e uns (1), conhecidos como bits (*binary digits*, ou dígitos binários). Portanto, todas as informações correspondentes a sinais de som, vídeo e teclado (números e letras), por exemplo, devem ser convertidas em bits para que sejam processadas por um sistema digital. Devido ao número de códigos diferentes criados para a representação de grandezas digitais, fez-se necessário desenvolver circuitos eletrônicos capazes de converter um código em outro, conforme a aplicação.

Um codificador é um circuito lógico que converte um conjunto de sinais de entrada em determinado código, adequado ao processamento digital.

3.1.1 Codificador de M-N (M entradas e N saídas)

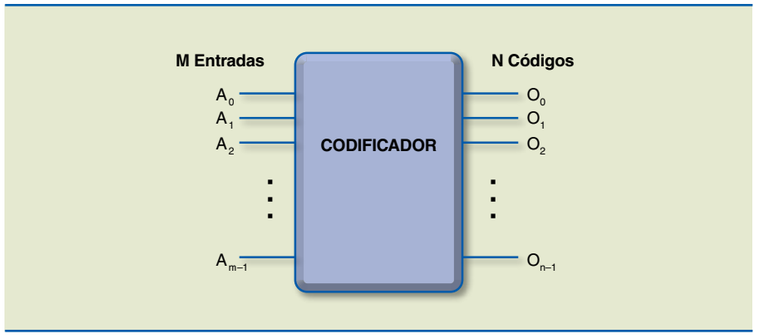


Figura 3.1
Codificador M entradas e N saídas.

3.1.2 Exemplo de codificador decimal-binário

Um codificador decimal para binário possui dez entradas e quatro saídas. A qualquer momento, somente uma linha de entrada tem um valor igual a 1.

Por exemplo, acionando a tecla 6 ($A_6 = 1$), teremos o binário de saída 0110, ou seja, $S_3 = 0$, $S_2 = 1$, $S_1 = 1$ e $S_0 = 0$ (figura 3.2).

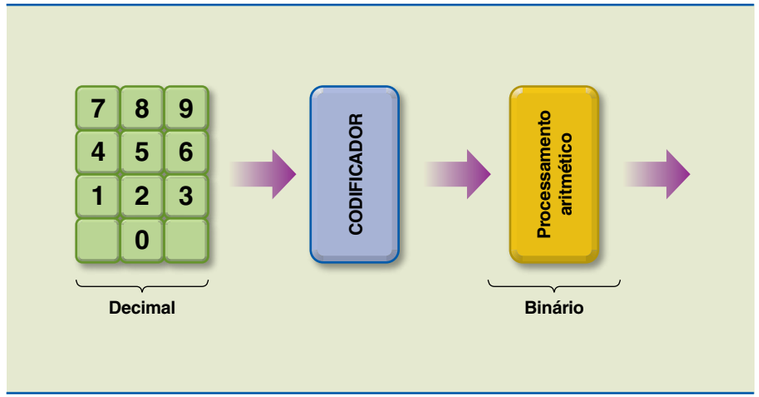


Figura 3.2
Codificador decimal-binário.

O diagrama em blocos do codificador pode ser representado conforme a figura 3.3.

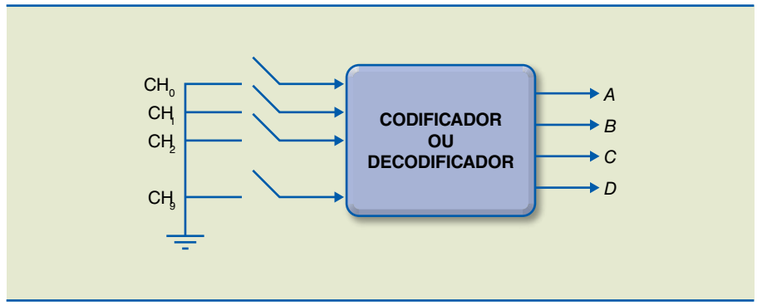


Figura 3.3
Diagrama em blocos do codificador.

Para esse codificador, temos a tabela verdade reproduzida a seguir:

CH_0	CH_1	CH_2	CH_3	CH_4	CH_5	CH_6	CH_7	CH_8	CH_9	A	B	C	D
0	1	1	1	1	1	1	1	1	1	*0*	*0*	*0*	*0*
1	0	1	1	1	1	1	1	1	1	*0*	*0*	*0*	*1*
1	1	0	1	1	1	1	1	1	1	*0*	*0*	*1*	*0*
1	1	1	0	1	1	1	1	1	1	*0*	*1*	*1*	*1*
1	1	1	1	0	1	1	1	1	1	*0*	*1*	*0*	*0*
1	1	1	1	1	0	1	1	1	1	*0*	*1*	*0*	*1*
1	1	1	1	1	1	0	1	1	1	*0*	*1*	*1*	*0*
1	1	1	1	1	1	1	0	1	1	*0*	*1*	*1*	*1*
1	1	1	1	1	1	1	1	0	1	*1*	*0*	*0*	*0*
1	1	1	1	1	1	1	1	1	0	*1*	*0*	*0*	*1*

Codificador com prioridade

Se observarmos com cuidado o circuito do codificador apresentado na figura 3.3, reconheceremos as seguintes limitações: se mais do que duas entradas forem ativadas simultaneamente, a saída será imprevisível ou então não aquela que esperávamos. Essa ambiguidade é resolvida estabelecendo uma prioridade de modo que apenas uma entrada seja codificada, não importando quantas estejam ativas em determinado instante.

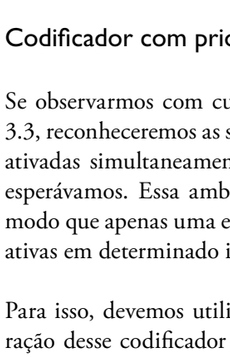
Para isso, devemos utilizar um codificador com função de prioridade. A operação desse codificador é tal que, se duas ou mais entradas forem ativadas ao mesmo tempo, a entrada que tem a prioridade mais elevada terá precedência.

Exemplo de circuito integrado 74147: codificador com prioridade decimal-BCD

A figura 3.4 identifica os pinos do CI 74147 e a tabela verdade correspondente.

Tabela verdade

Figura 3.4
Circuito integrado 74147: codificador com prioridade decimal-BCD.



Entradas									Saídas			
1	2	3	4	5	6	7	8	9	D	C	B	A
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	1	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

Observando a tabela verdade do circuito integrado da figura 3.4, concluímos que nove linhas de entrada ativas (ativas em nível baixo) representam os números decimais de 1 a 9. A saída do CI sugerido é o código BCD invertido, correspondente à entrada de maior prioridade. Caso todas as entradas estejam inativas (inativas em nível alto), então as saídas estarão todas em nível alto. As saídas ficam normalmente em nível alto quando nenhuma entrada está ativa (figura 3.5).

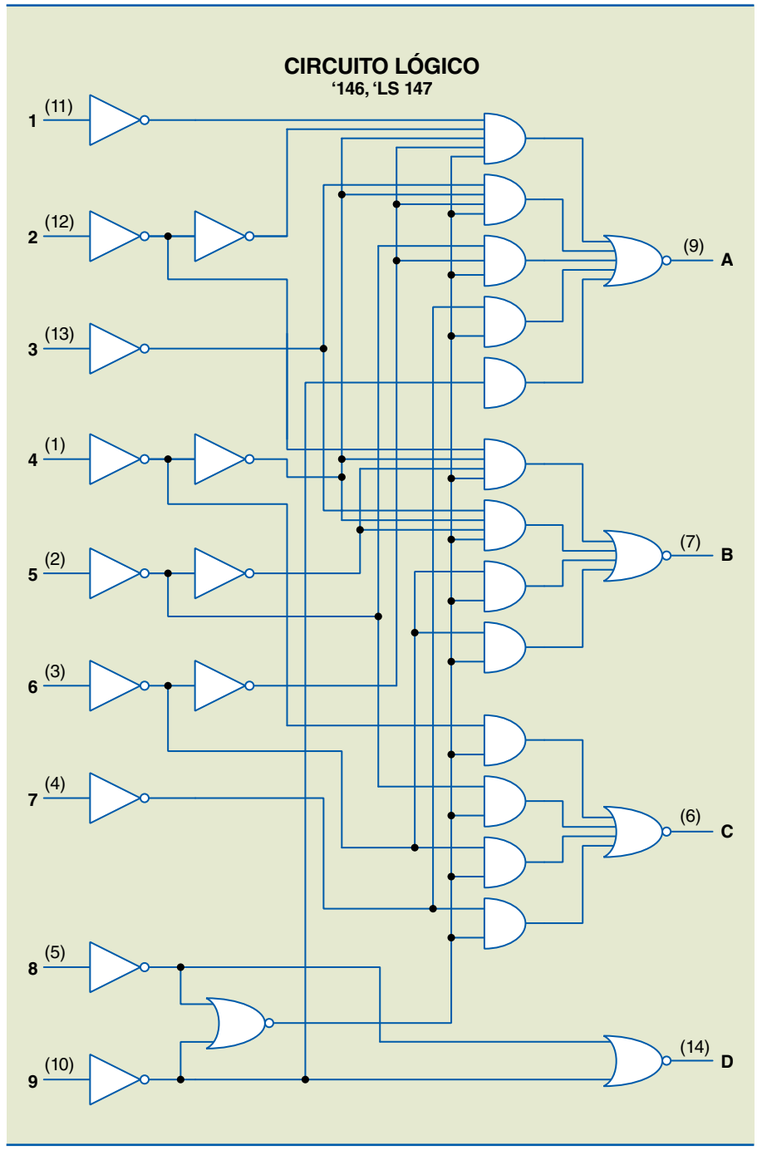


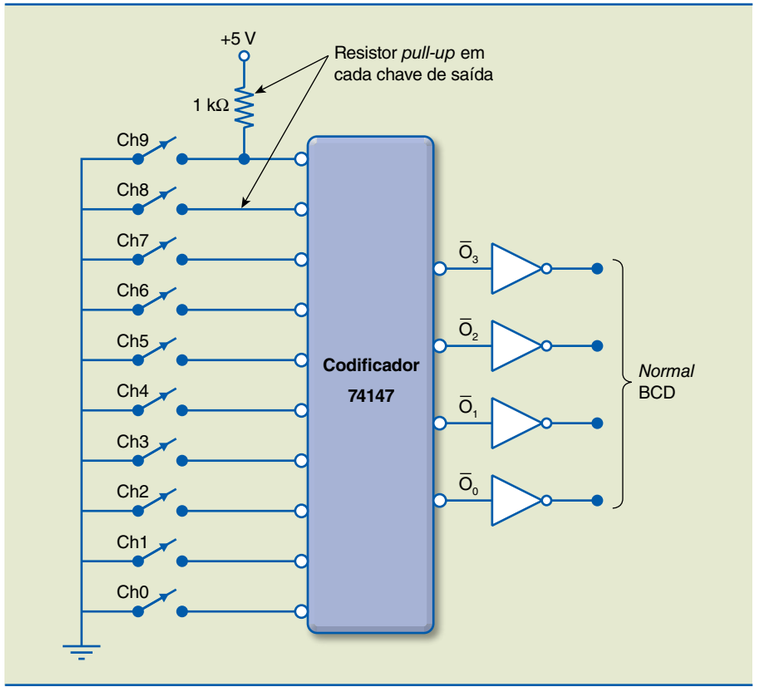
Figura 3.5
Circuito lógico: configuração das portas lógicas do circuito integrado da figura 3.4.

Exemplo de aplicação do CI 74147 em um teclado

Se as chaves estiverem abertas, todas as entradas estarão em nível alto e as saídas em 0000. Se qualquer chave estiver fechada, a entrada correspondente estará em nível baixo e as saídas assumirão o valor do código BCD do número da chave.

O CI 74147 é um exemplo de circuito com prioridade. Dessa maneira, a saída ativa será relativa à chave de maior prioridade entre aquelas que estiverem fechadas em determinado momento (figura 3.6).

Figura 3.6
Exemplo de aplicação do CI 74147.

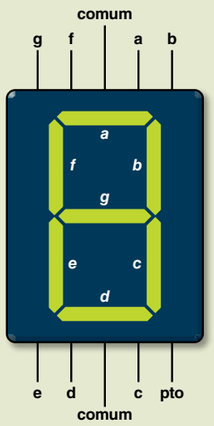


O decodificador também é um circuito combinacional, normalmente usado para habilitar uma, e somente uma, dentre *m* saídas por vez, quando aplicada uma combinação binária específica em suas *n* entradas.

Exemplo de decodificador HEX/BCD – sete segmentos

O *display* de sete segmentos apresenta sete LEDs dispostos de modo que se observe uma estrutura em forma de oito, conforme mostra a figura 3.7.

Figura 3.7
Display de sete segmentos.



Quando queremos, por exemplo, acender o número "0", polarizamos diretamente os *LEDs* (segmentos) que formam o dígito "0" no *display*, ou seja, os segmentos *a*, *b*, *c*, *d*, *e*, *f*, para ser possível visualizar o dígito, conforme ilustrado na figura 3.8.

Figura 3.8
Representação do ***LED*** indicando o número zero.

Para acionar adequadamente o *display* de sete segmentos a fim de visualizarmos o código hexadecimal, é necessário um decodificador com as características apresentadas na figura 3.9 e na tabela verdade correspondente.

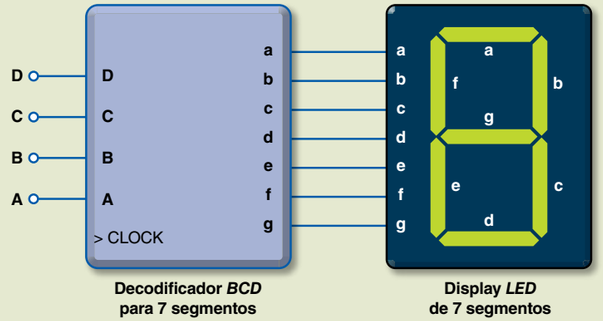
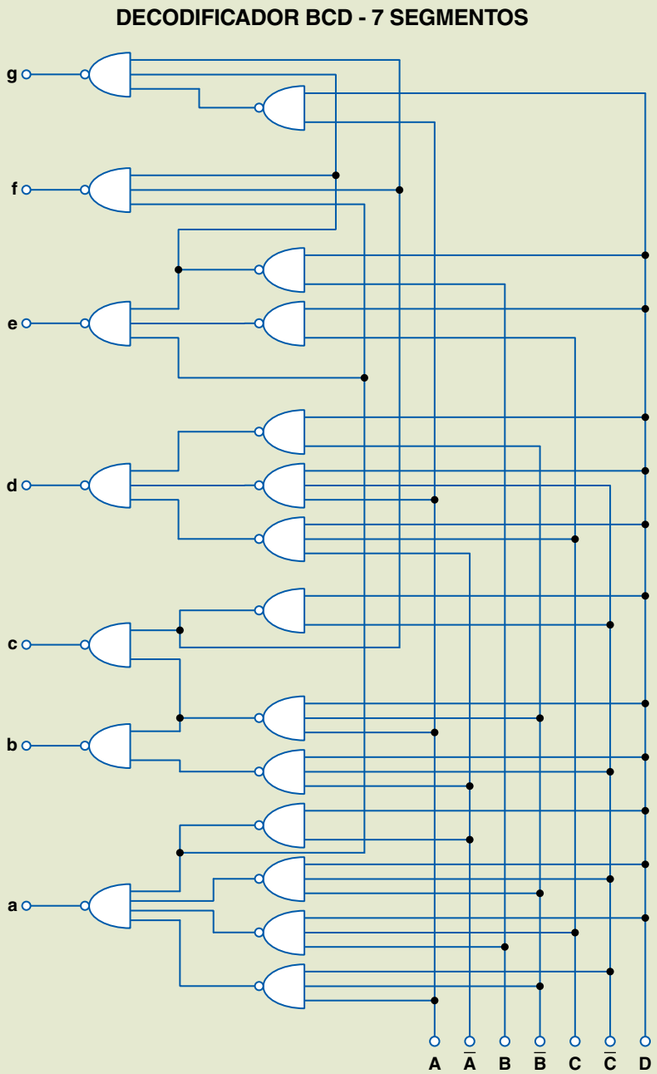


Figura 3.9
Representação do ***display*** e tabela verdade para cada um dos segmentos.

Binary Inputs				Decoder Outputs							Segment Display Outputs
D	C	B	A	a	b	c	d	e	f	g	
Ø	Ø	Ø	Ø	1	1	1	1	1	1	Ø	0
Ø	Ø	Ø	1	Ø	1	1	Ø	Ø	Ø	Ø	1
Ø	Ø	1	Ø	1	1	Ø	1	1	Ø	1	2
Ø	Ø	1	1	1	1	1	1	Ø	Ø	1	3
Ø	1	Ø	Ø	Ø	1	1	Ø	Ø	1	Ø	4
Ø	1	Ø	1	1	Ø	1	1	Ø	1	1	5
Ø	1	1	Ø	1	Ø	1	1	1	1	1	6
Ø	1	1	1	1	1	1	Ø	Ø	Ø	Ø	7
1	Ø	Ø	Ø	1	1	1	1	1	1	1	8
1	Ø	Ø	1	1	1	1	1	Ø	1	1	9

Resolvendo os diagramas de Karnaugh correspondentes aos sete segmentos, obtemos o circuito lógico conforme mostra a figura 3.10.

Figura 3.10
Representação do circuito lógico do decodificador de sete segmentos.



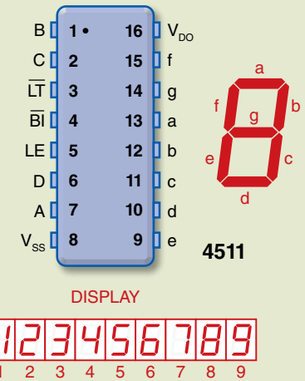
Exemplo de decodificador BCD – sete segmentos

A maior parte das aplicações com *displays* requer que trabalhemos com o código decimal (BCD). Uma possibilidade é utilizar o CI 4511, que é um decodificador BCD – 7 segmentos. A figura 3.11 mostra a representação dos pinos desse circuito e a tabela verdade detalhada.

Figura 3.11
Representação dos pinos do CI 4511 e tabela verdade para cada um dos segmentos.

Tabela verdade

Entradas BCD				Saídas							*Display* tipo cátodo comum
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9



Entrada D = MSB e entrada A = LSB

Para os códigos binários correspondentes aos dígitos maiores do que 9 (1010 até 1111), todas as saídas são colocadas em nível "0" e, consequentemente, todos os segmentos do *display* ficam apagados (função *blank*).

Sinais de controle

Para visualizarmos os códigos, conectamos as entradas LT (*lamp test*) e BI (*ripple blanking input*) em nível lógico "1" e a entrada LE (*latch enable*) em nível lógico "0". Para testarmos os segmentos do *display*, conectamos a entrada LT em nível lógico "0" (todos os segmentos do *display* deverão acender, independentemente do código presente nas entradas D, C, B e A).

A entrada LE pode ser utilizada (quando em nível lógico "1") para armazenar o código presente nas entradas BCD. O *display* permanecerá inalterado até que se aplique nível lógico "0" na entrada LE para um novo código presente nas entradas BCD.

Conexões externas

O diagrama da figura 3.12 ilustra a utilização do CI com *display* de sete segmentos cátodo comum.

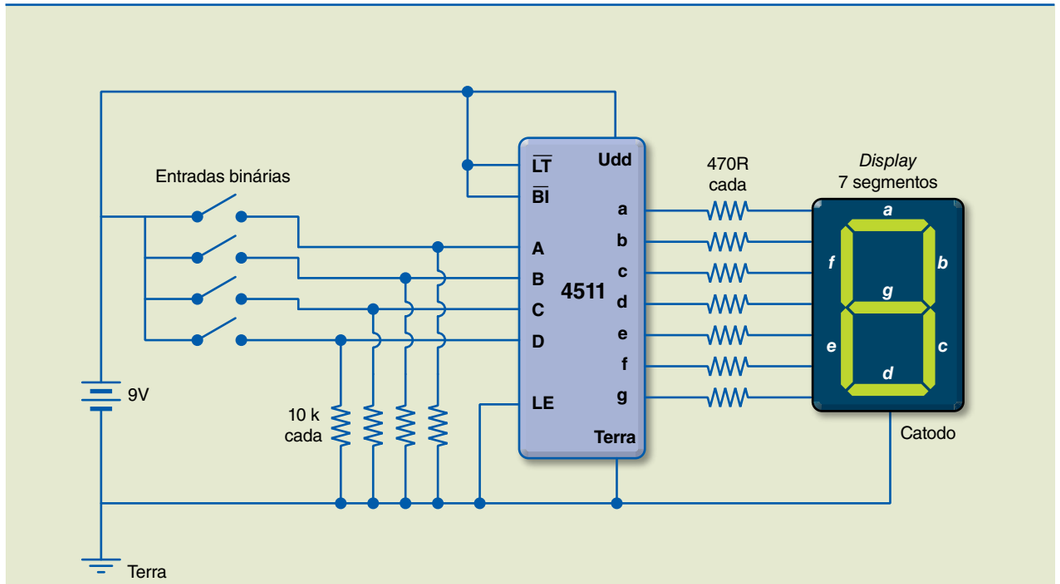


Figura 3.12
CI com ***display*** de sete segmentos cátodo comum.

3.2 Multiplexadores/demultiplexadores

Consideremos a seguinte situação: queremos transferir dados lógicos ("0", "1") de quatro entradas para oito saídas, com a possibilidade de qualquer entrada se comunicar com qualquer saída, tendo para isso uma única via de transferência de dados (figura 3.13).

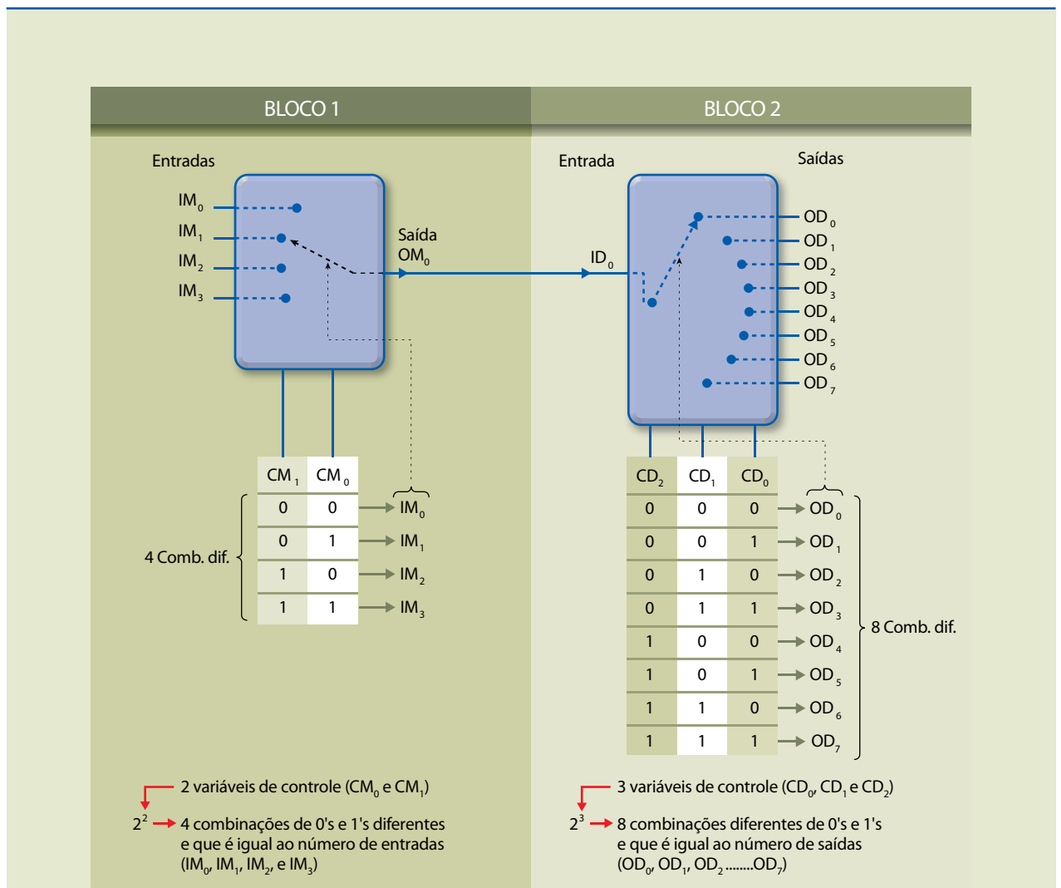


Figura 3.13
Transferência de dados entre os blocos 1 e 2.

Na figura 3.13, o bloco 1 apresenta a ideia básica de um multiplexador (MUX), ou seja, de várias entradas, uma é selecionada e direcionada para a saída. A seleção é representada na figura por uma chave; no circuito real, a seleção é feita por meio das variáveis de controle (seleção). Nesse exemplo, o multiplexador tem quatro entradas (IM_0, IM_1, IM_2, IM_3) e, portanto, precisamos de duas variáveis de controle, pois é possível com elas obter quatro combinações de "0" e "1" diferentes.

O bloco 2 apresenta a ideia básica de um demultiplexador (DEMUX), ou seja, a entrada única de dados é direcionada para uma das várias saídas, para a saída selecionada.

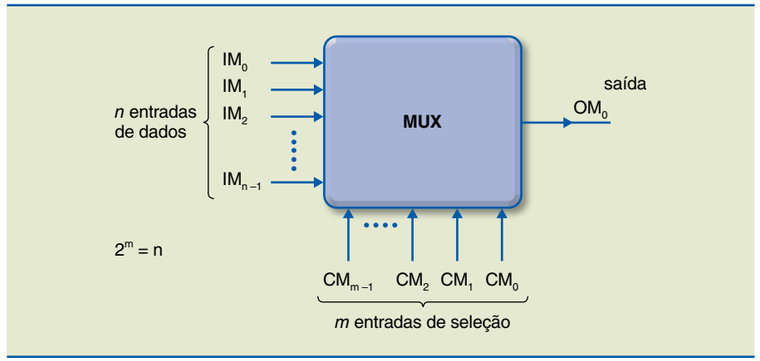
A tabela 3.5 registra, em cada linha, o "caminho" de determinada entrada até certa saída por meio das variáveis de controle de entrada no MUX e das variáveis de controle de saída no DEMUX.

Tabela 3.1
Tabela verdade

ENTRADA						
DADOS	CONTROLE					
	MUX		DEMUX			SAÍDA
	CM_1	CM_0	CD_2	CD_1	CD_0	
IM_2	1	0	0	1	1	OD_3
IM_0	0	0	1	1	0	OD_6
IM_2	1	0	1	1	0	OD_6
IM_3	1	1	0	0	1	OD_1
IM_1	0	1	1	1	1	OD_7
IM_0	0	0	1	0	0	OD_4
IM_0	0	0	0	1	0	OD_2
IM_1	0	1	1	0	1	OD_5T

A figura 3.14 representa um multiplexador de *n* entradas de dados, *m* entradas de controle (seleção) e uma saída.

Figura 3.14
Multiplexador.



Vamos implementar um MUX de oito entradas. Para isso, necessitamos de três variáveis de controle, pois $2^3 = 8$, que corresponde ao número de entradas (tabela verdade).

Variáveis de controle			Saída	Produtos das variáveis de controle
CM_2	CM_1	CM_0	OM_0	
0	0	0	IM_0	$CM_2.CM_1.CM_0$
0	0	1	IM_1	$CM_2.CM_1.CM_0$
0	1	0	IM_2	$CM_2.CM_1.CM_0$
0	1	1	IM_3	$CM_2.CM_1.CM_0$
1	0	0	IM_4	$CM_2.CM_1.CM_0$
1	0	1	IM_5	$CM_2.CM_1.CM_0$
1	1	0	IM_6	$CM_2.CM_1.CM_0$
1	1	1	IM_7	$CM_2.CM_1.CM_0$

Tabela 3.2
Tabela verdade

Observe na tabela verdade que a coluna "Saída" corresponde às entradas selecionadas pelas variáveis de controle, como deve ocorrer em um MUX, ou seja, $OM_0 = IM$ selecionada.

Sabemos que, se todas as entradas de uma porta E forem "1", exceto uma, que poderá ser "1" ou "0", a saída da porta será "1" ou "0". Então, temos, por exemplo, a figura 3.15.

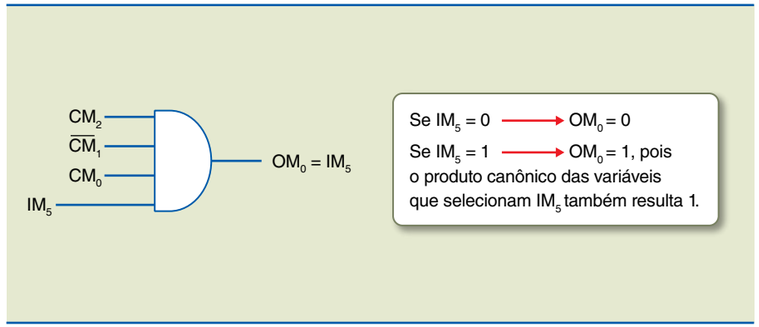
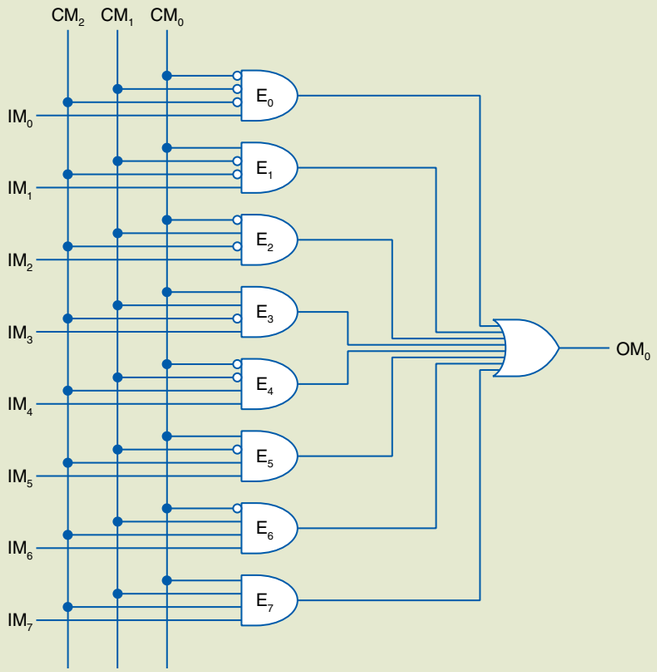


Figura 3.15
Porta E com três entradas.

Assim, podemos implementar o MUX de oito entradas e três variáveis de controle como apresentado na figura 3.16.

Figura 3.16
MUX de oito entradas e três variáveis.



A "bolinha" (○) indica que a entrada foi complementada, substituindo, na representação, a porta inversora.

Podemos implementar o MUX por meio da tabela verdade. Para isso, devemos considerar que a tabela verdade terá como entrada oito variáveis de dados e três variáveis de controle – assim, em princípio, uma tabela verdade com 2^{11} = 2 048 combinações.

Somente as linhas em que a variável de dados selecionada é "1", a saída é 1 e essa condição independe das demais variáveis de dados. Para isso, temos de levar em consideração oito linhas das 2 048, e, portanto, a função booleana de saída é a soma do produto dessas oito linhas:

$$OM_0 = IM_0 \cdot \overline{CM_2} \cdot \overline{CM_1} \cdot \overline{CM_0} + IM_1 \cdot \overline{CM_2} \cdot \overline{CM_1} \cdot CM_0 + IM_2 \cdot \overline{CM_2} \cdot CM_1 \cdot \overline{CM_0} + IM_3 \cdot \overline{CM_2} \cdot CM_1 \cdot \underline{CM_0} + IM_4 \cdot CM_2 \cdot \overline{CM_1} \cdot \overline{CM_0} + IM_5 \cdot CM_2 \cdot \overline{CM_1} \cdot CM_0 + IM_6 \cdot CM_2 \cdot CM_1 \cdot \overline{CM_0} + IM_7 \cdot CM_2 \cdot CM_1 \cdot CM_0$$

Essa função booleana é executada pelo circuito da figura 3.16 (oito portas E e uma porta OU).

É possível implementar funções lógicas diretamente em um multiplexador. Os exemplos a seguir ilustram essa técnica.

Exemplos

1. Seja a função $y = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$.

Escolhemos um multiplexador com três entradas de controle (seleção), pois a função possui três variáveis independentes (figura 3.17). Fazemos uma tabela verdade, relacionando as variáveis de controle e as de dados.

Tabela verdade
$y = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$

A	B	C	y	Dados
0	0	0		$I_0 = 0$
0	0	1	$\overline{A} \cdot \overline{B} \cdot C$	$I_1 = 1$
0	1	0		$I_2 = 0$
0	1	1		$I_3 = 0$
1	0	0	$A \cdot \overline{B} \cdot \overline{C}$	$I_4 = 1$
1	0	1	$A \cdot \overline{B} \cdot C$	$I_5 = 1$
1	1	0		$I_6 = 0$
1	1	1		$I_7 = 0$

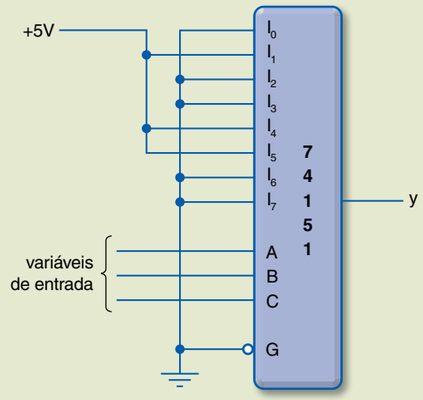


Figura 3.17
Multiplexador com três entradas de controle e tabela verdade correspondente.

As variáveis de dados I_1, I_4, e I_5 são levadas para nível "1", pois correspondem às entradas do MUX que são selecionadas pelas variáveis de controle e que aparecerão na saída conforme estabelecido pela função. As demais variáveis são levadas para o nível "0". As variáveis de controle são as dependentes da função booleana. A variável independente é representada pela saída do multiplexador.

Para implementarmos o circuito da figura 3.17, podemos usar o CI TTL 74151 – multiplexador digital de oito canais (figura 3.18).

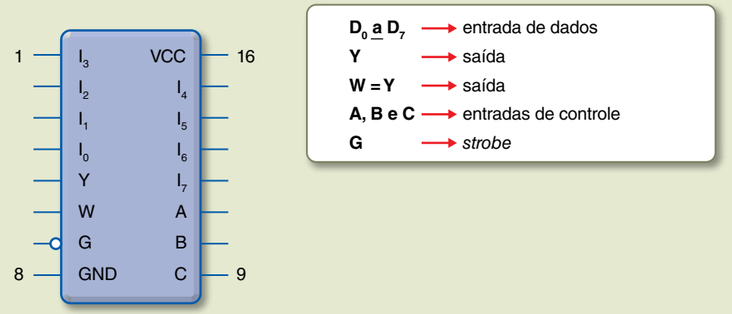


Figura 3.18
Pinagem do CI TTL 74151 – multiplexador digital de oito canais (16 pinos).

Analisando a figura 3.18, temos:

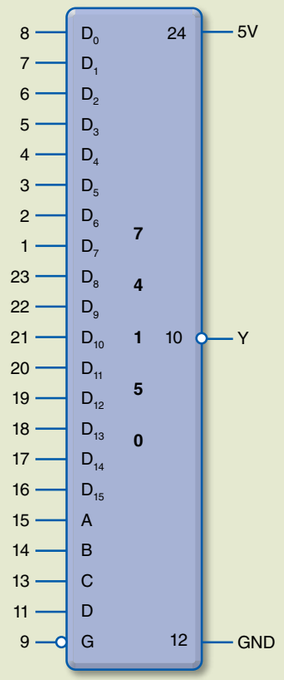
- Y apresenta o valor da variável selecionada;
- W = Y; G é ativo em nível baixo (indicado com a "bolinha" na representação da figura), o que significa que em G = 0 o MUX está liberado para funcionamento normal; para G = 1, Y = 0 independentemente dos valores das entradas A, B e C.

Agora, vamos analisar o CI TTL 74150 (figura 3.19) – multiplexador digital de 16 canais (24 pinos) – e a tabela verdade correspondente.

Figura 3.19
Pinagem do CI TTL 74150 e tabela verdade correspondente.

Tabela verdade

strobe	seleção				saída
G	A	B	C	D	$\overline{Y}$
0	0	0	0	0	D_0
0	0	0	0	1	D_1
0	0	0	1	0	D_2
0	0	0	1	1	D_3
0	0	1	0	0	D_4
0	0	1	0	1	D_5
0	0	1	1	0	D_6
0	0	1	1	1	D_7
0	1	0	0	0	D_8
0	1	0	0	1	D_9
0	1	0	1	0	D_{10}
0	1	0	1	1	D_{11}
0	1	1	0	0	D_{12}
0	1	1	0	1	D_{13}
0	1	1	1	0	D_{14}
0	1	1	1	1	D_{15}
1	X	X	X	X	0



Analisando a figura 3.19, temos:

- a saída Y é complemento da entrada selecionada (ver representação – tem "bolinha");
- o *strobe* é ativo em 0 (ver representação – tem "bolinha");
- G = 1 → Y = 0, independentemente de A, B, C e D.

2. Seja, na figura 3.20, a função $y = A \cdot B \cdot C + A \cdot \overline{B \cdot C} + A \cdot B \cdot \overline{C}$. A tabela verdade representa a função utilizada.

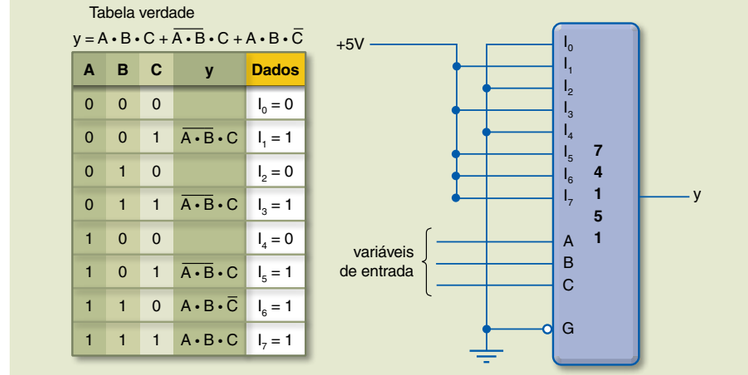


Figura 3.20
Pinagem do CI 74151 referentes à função utilizada e a tabela verdade correspondente.

Pela associação de multiplexadores, é possível aumentar o número de entradas do circuito original, conforme mostra a figura 3.21, e montar um multiplexador de 16 canais utilizando multiplexadores de oito canais cada. Para isso, vamos utilizar o CI 74151, que já conhecemos.

Figura 3.21
Associação de multiplexadores utilizando CI 74151.

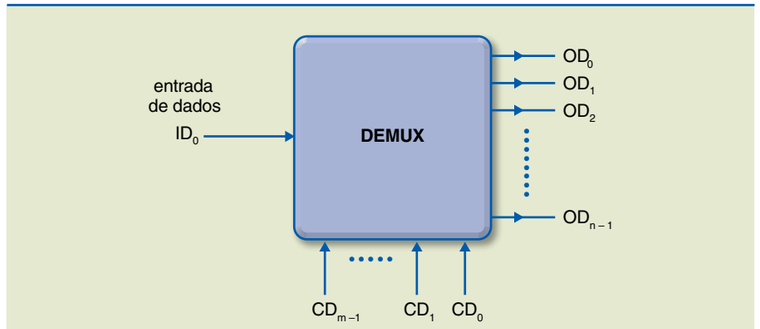
Analisando a figura 3.21, podemos notar que D é o bit MSB (bit mais significativo) dos bits de seleção. Assim, temos como exemplos dois endereços:

D A B C
0 1 0 1 → IF_5 D = 0 seleciona as entradas IF_0 a IF_7
1 0 1 1 → IF_{11} D = 1 seleciona as entradas IF_8 a IF_{15}

O demultiplexador realiza a função inversa do multiplexador, ou seja, a informação recebida em uma única entrada de dados é enviada para uma saída selecionada por variáveis de controle (seleção).

O demultiplexador representado na figura 3.22 tem m entradas de controle e n saídas.

Figura 3.22
Representação do DEMUX.



Vamos implementar um DEMUX de oito saídas. Para isso, necessitamos de três variáveis de controle, pois $2^3 = 8$, que corresponde ao número de saídas. Como são oito saídas, há oito tabelas verdades, que podem ser montadas em uma só com as mesmas entradas e as respectivas saídas.

Entradas				Saídas							
CD_2	CD_1	CD_0	ID_0	OD_7	OD_6	OD_5	OD_4	OD_3	OD_2	OD_1	OD_0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	I	I	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...

Tabela 3.3
Tabela verdade para um DEMUX de oito saídas (representação parcial)

Analisando cada saída, sem a necessidade de montar a tabela verdade completa, concluímos que ela somente será "1" se a entrada de dados for "1", uma vez que o produto canônico correspondente a essa saída será "1". Qualquer outra condição levará a saída para "0". A figura 3.23 apresenta um exemplo.

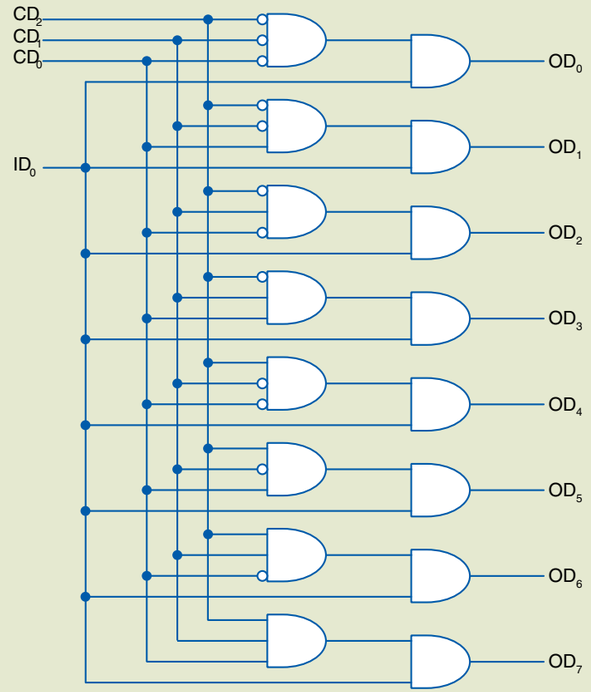
$$OD_3 = \underbrace{ID_0 \cdot CD_0 \cdot CD_1 \cdot \overline{CD}_2}$$

produto canônico das variáveis de controle para seleção de OD_3

Figura 3.23
Exemplo para análise da condição estabelecida no DEMUX de oito saídas.

Com a expressão booleana de cada saída obtida de maneira semelhante, podemos implementar o circuito do DEMUX com portas lógicas (figura 3.24).

Figura 3.24



Entradas						Saídas															
G1	G2	D	C	B	A	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}
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Tabela 3.4
Tabela verdade (CI 74154)

Observando a tabela verdade da figura 3.24, podemos notar que as duas entradas *strobe* G1 e G2 são ativas em nível baixo, e, para seu funcionamento normal, elas devem estar em nível baixo. Se G1 e G2 não estiverem em nível baixo, todas as saídas vão para nível alto. Observe que, em funcionamento normal, somente a saída selecionada está em nível baixo; as demais encontram-se em nível alto.

Vamos usar o CI 74154 (figura 3.25) para executar a função.

$$y = A\cdot\overline{B}\cdot C\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D} + A\cdot B\cdot C\cdot D$$

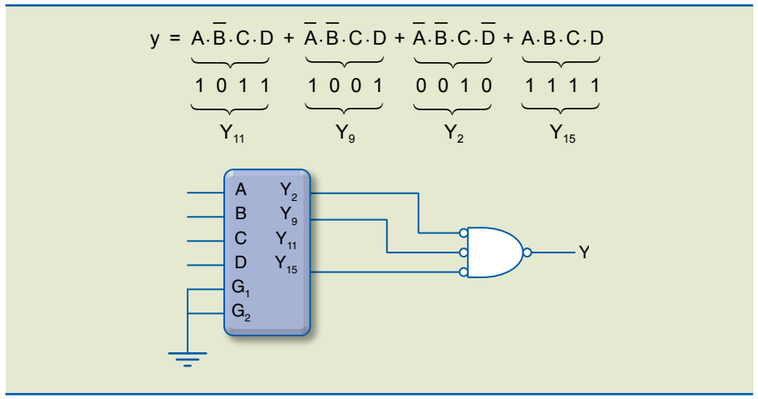


Figura 3.25

Analisando a figura 3.25, podemos perceber que, como Y é saída de uma porta NE, se uma das entradas for "0", Y será igual a "1". Isso só acontece se uma das saídas corresponder a um dos termos da função booleana de Y, selecionada pelas variáveis de controle (A, B, C, D).

Da mesma forma como foi feito com os multiplexadores, é possível a combinação de demultiplexadores para aumentar a capacidade do circuito, conforme exemplo da figura 3.26. Utilizando o 74154, vamos montar um demultiplexador de 32 saídas.

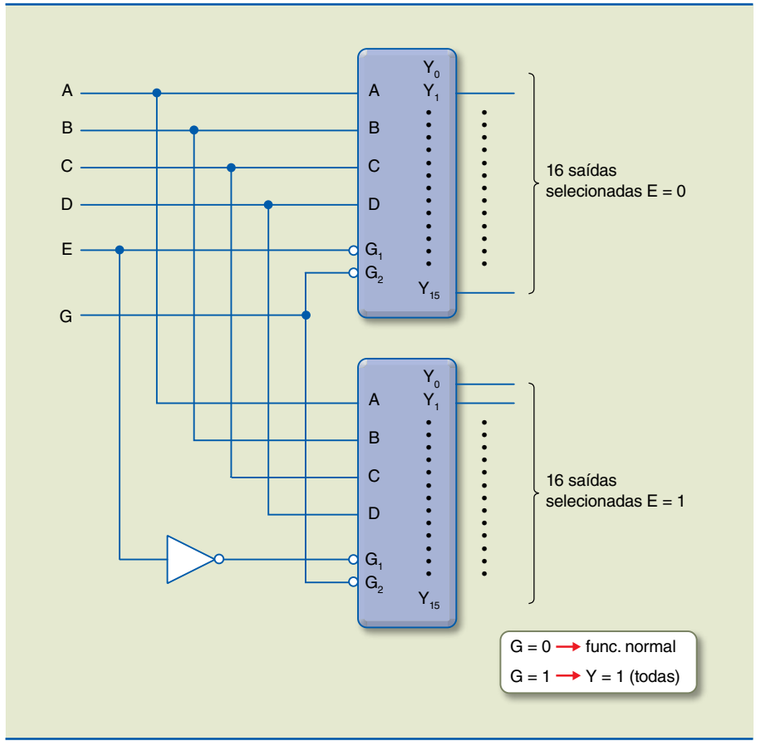
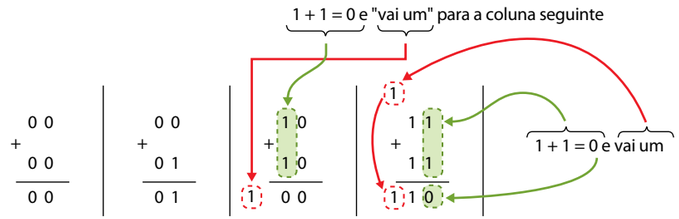


Figura 3.26
DEMUX de 32 saídas.

3.3 Circuitos aritméticos

O microprocessador, componente fundamental de um computador, tem em sua arquitetura interna uma ULA (unidade lógica aritmética), na qual são realizadas as operações lógicas e aritméticas. Associando portas lógicas de maneira conveniente, podemos obter circuitos que realizam operações aritméticas. Devemos lembrar que **portas lógicas** têm como entrada **estados lógicos** que foram associados aos símbolos “0” e “1”, e **circuitos aritméticos** têm como entrada **números**.

A adição, a subtração e a multiplicação de números binários e decimais são efetuadas de modo semelhante, lembrando que o “vai um” em binário ocorre quando a soma dos dígitos é 2 e não 10 como em decimal. Por exemplo:



Agora, vamos calcular:

B_1 = (0101 0011 + 0110 1001) e B_2 = (0101 1101 + 1000 1110):

$$\begin{array}{rcl} 1 \quad 1 & \leftarrow \text{ os “vai um” } \rightarrow & 1 \quad 1 \\ 0101 \quad 0011 & & 0101 \quad 1101 \\ + & & + \\ \underline{0110 \quad 1001} & & \underline{1000 \quad 1110} \\ 1011 \quad 1100 & & 1110 \quad 1011 \end{array}$$

$$B_1 = 1011\ 1100 \qquad B_2 = 1110\ 1011$$

Os microprocessadores não possuem circuitos de subtração, porém essa operação pode ser realizada por meio da adição usando números na forma complemento 2. Para isso, vamos primeiro considerar ,por exemplo, o cálculo de $X = 85 - 37$, ou seja, uma subtração em decimal.

Somando 100 e subtraindo 100 do segundo membro da equação, o valor de X não se altera. Portanto, temos $X = 85 - 37 + (100 - 100) = 85 + (100 - 37) - 100 = 85 + 63 - 100$. O valor $(100 - 37)$ poderia ser obtido complementando os algarismos de 37 para o valor 9 e somando 1, resultando $(62 + 1) = 63$. Assim, temos $X = (85 + 63) - 100 = 148 - 100$. Nesse caso, subtrair 100 equivale a desprezar o último algarismo do 148, resultando $X = 48$, que é o resultado procurado.

Vamos analisar outro exemplo com números decimais, aplicando, agora, a regra usada no exemplo anterior generalizada: $X = 743 - 571$. Somamos ao minuendo o complemento 9 do subtraendo mais 1 e desprezamos o último dígito à esquerda, resultando $X = 743 + (428 + 1) = 1172$. Assim, temos $X = 172$ e chegamos ao resultado correto fazendo um cálculo diferente do usual.

Vamos retornar ao primeiro exemplo:

$X = 85 - 37$ (estamos **subtraindo** do número positivo 85 o número positivo 37) é equivalente a

$X = 85 + (-37)$ (estamos **somando** ao número positivo 85 o número negativo –37)

Observe que, ao desenvolvermos o cálculo no exemplo, tínhamos chegado a $X = 85 + 63$ e desprezamos o último algarismo da esquerda. Comparando $X = 85 + (-37)$ com $X = 85 + 63$ (desprezando o último algarismo), o número 63 poderia ser interpretado como o negativo de 37, pois o resultado foi igual. Com o mesmo raciocínio, poderíamos interpretar no segundo exemplo o número 429 como o negativo de 571.

No processo usado para obtermos o resultado, a complementação do subtraendo foi feita para 9, ou seja, para o valor da base tirando 1 (sistema decimal 10 – 1 = 9).

Procedimento similar é usado na base 2 para transformar uma operação de subtração em uma adição. No caso de binários, que são base 2, a complementação do subtraendo seria para 1, e complementaríamos o processo somando 1, obtendo, assim, a representação complemento 2 do binário a ser subtraído.

Complementar dígitos binários para 1 não é difícil, uma vez que se trata de circuito numérico correspondente a porta lógica inversora com **entradas numéricas** (0 ou 1). Somar com circuitos digitais também é simples. Portanto, a ideia exemplificada em decimais é usada em sistemas binários. O objetivo é transformar operações de subtração em adição, que é mais fácil de implementar com circuitos digitais.

Em binário, quando é necessário trabalhar com números negativos, o primeiro bit da esquerda é reservado para definição do sinal. Assim, quando trabalhamos com binário com sinal, precisamos saber o número de dígitos com que os números serão apresentados. Os binários negativos têm o primeiro bit da esquerda igual a “1”, e os binários positivos, igual a “0”. Usando esse critério, ou seja, ter bem definida a posição do bit de sinal, podemos representar os binários negativos pelo **complemento 2** de seu valor positivo.

Trabalhando com números de oito bits, temos, por exemplo:

1001 1101 corresponde a um número negativo em representação binária.
0010 0110 corresponde ao decimal 38 positivo.

Vamos representar com oito dígitos, em binário, os números decimais 37, –6, –19 e –97 em representação complemento 2.

Binário (+37) 0 0 1 0 0 1 0 1

Binário (-6)

Binário (+6)	0000 0110
Achamos complemento 1	1111 1001
Somamos 1	0000 0001 +
Binário (-6) →	1111 1010

Binário (-19)

Binário (+19)	0001 0011
Achamos complemento 1	1110 1100
Somamos 1	0000 0001 +
Binário (-19) →	1110 1101

Binário (-97)

Binário (+97)	0110 0001
Achamos complemento 1	1001 1110
Somamos 1	1 +
Binário (-97) →	1001 1111

Qual o valor decimal que corresponde ao binário com sinal B_1 = 1001 1101 representado em complemento 2?

B_1 representa um binário negativo, pois o primeiro dígito à esquerda é 1; portanto, o correspondente decimal do complemento 2 dele com sinal negativo é o valor procurado.

Binário B_1	1001 1101
Complemento 1 de B_1	0110 0010 +
Somando 1	1
Representação complemento 2 de B_1	0110 0011
Convertendo em decimal	$-B_1 = (64 + 32 + 1)_{10} = (-99)_{10}$
Portanto	$B_1 = (-99)_{10}$

A representação complemento 2 de um binário significa o valor negativo do binário, independentemente de ele ser positivo ou negativo.

A tabela 3.5 apresenta alguns números binários com sinal representados com oito dígitos e seu respectivo valor decimal.

Tabela 3.5

decimal	binário	decimal	binário
+ 99	0110 0011	– 6	1111 1010
+ 46	0010 1110	– 19	1110 1101
+ 21	0001 0101	– 21	1110 1011
+ 19	0001 0011	– 99	1001 1101

Observe os cálculos representados em números decimais usando os correspondentes binários e confira os resultados tendo como referência os valores decimais.

Exemplos

1. 21 – 19

Solução:

$$21 - 19 = 21 + (-19)$$

$$(0001\ 0101) - (0001\ 0011) = (0001\ 0101) + (1110\ 1101)$$

Transformamos a subtração em uma adição substituindo –19 pelo complemento 2 de +19, que corresponde ao valor negativo.

1111 1 1	← os “vai um”
0001 0101	
1110 1101 +	
10000 0010	

0 0 0 0 0 0 1 0 = $(2)_{10}$ O nono bit, último à esquerda, é desconsiderado, pois, conforme estabelecido no início deste estudo, estamos trabalhando com binários com sinal representados por oito dígitos. Conferindo a conta em decimal, o resultado bate.

2. 46 – 21

Solução:

$$46 - 21 = 46 + (-21)$$

$$(0010\ 1110) - (0001\ 0101) = (0010\ 1110) + (1110\ 1011)$$

Transformamos a subtração em uma adição substituindo –21 pelo complemento 2 de +21.

```
 1 1   1 1 1     ←  os "vai um"
 0 0 1 0  1 1 1 0
                +
 1 1 1 0  1 0 1 1
-------------------
1 0 0 0 1  1 0 0 1
```

$0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 = (25)_{10}$ O nono bit, último à esquerda, é desconsiderado, pois, conforme estabelecido no início deste estudo, estamos trabalhando com binários com sinal representados por oito dígitos. Conferindo a conta em decimal, o resultado bate.

3. 21 – 99

Solução:

$$21 - 99 = 21 + (-99)$$

$$(0001\ 0101) - (1001\ 1101) = (0001\ 0101) + (1001\ 1101)$$

Transformamos a subtração em uma adição substituindo –99 pelo complemento 2 de +99.

```
   1 1 1   1  ←   os "vai um"
 0 0 0 1  0 1 0 1
                +
 1 0 0 1  1 1 0 1
-------------------
 1 0 1 1  0 0 1 0
```

Como esperávamos, a operação resultou um número negativo. Para encontrarmos esse número em decimal, devemos calcular seu complemento 2, obtendo o binário positivo correspondente, e depois converter em decimal e trocar o sinal.

$$1011\ 0010 \rightarrow \text{complemento 1} \rightarrow 0100\ 1101 + 1 = 0100\ 1110 = (78)_{10}$$

Portanto, o resultado da operação em decimal é –78.

4. Determine o correspondente decimal do binário (1110 1011), em que o primeiro bit da esquerda é o bit de sinal.

Solução:

Determinamos o complemento 2 de (1110 1011), obtendo o valor positivo, convertemos em decimal e trocamos o sinal.

$$1110\ 1011 \rightarrow \text{complemento 1} \rightarrow 0001\ 0100 + 1 = 0001\ 0101 = (+21)_{10}$$

Portanto, o binário (1110 1011) corresponde ao decimal –21.

3.3.1 Meio somador

Vamos observar as tabelas do OU EXCLUSIVO, do E, e da soma aritmética de dois binários de um dígito cada um (um bit).

Porta lógica E e OR EXCLUSIVO

entrada		A ⊕ B	A • B
A	B	S	
0	0	0	**0**
0	1	1	**0**
1	0	1	**0**
1	1	1	**1**

SOMA ARITMÉTICA

entrada		(A + B)		
A	B	C	S	(decimal)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	2

Tabela 3.6
Porta lógica E e OU EXCLUSIVO e SOMA ARITMÉTICA

As colunas assinaladas mostram que podemos usar o circuito da porta OU EXCLUSIVO para obter os dígitos menos significativos da soma aritmética (A + B), tendo como entradas variáveis numéricas binárias A e B. Podemos também usar o circuito da porta lógica E para obter o dígito seguinte, dígito "vai um", tendo como entradas variáveis numéricas binárias A e B (figura 3.27).

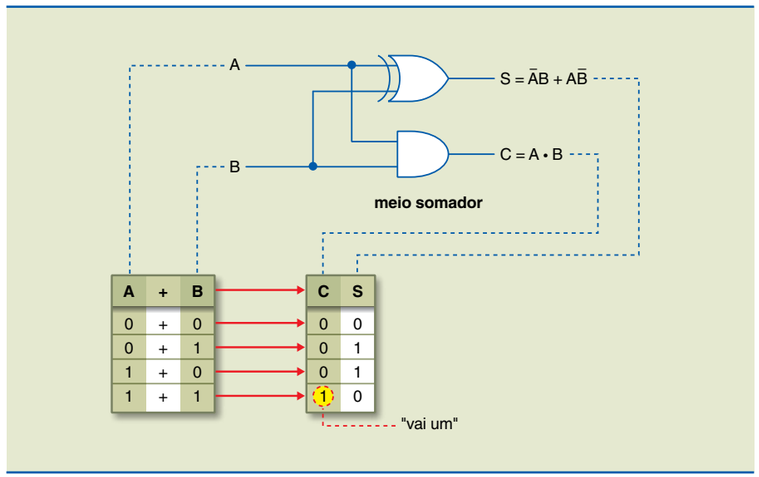
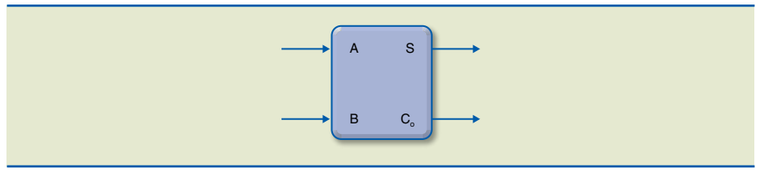


Figura 3.27
Circuito meio somador.

O meio somador é também conhecido como *half adder* (inglês), e o dígito de transporte C, como *carry* (inglês).

Representando o meio somador em um único bloco, temos a figura 3.28.

Figura 3.28
Circuito meio somador.



Não é possível somar mais de dois algarismos com o meio somador. Para atendermos a essa condição, devemos utilizar o somador completo.

3.3.2 Somador completo

Consideremos a soma de dois números binários (1000 1011) + (0101 0111), representada no exemplo:

```
   1 1 1 1 1   →  C
1 0 0 0 1 0 1 1 →  A
0 1 0 1 0 1 1 1 →  B
---------------
1 1 1 0 0 0 1 0 →  S
                   ⌐
                   └→ Meio Somador (Ver Figura 3.27)
```

Os bits da primeira coluna à direita e o "vai um" podem ser obtidos com o meio somador. A partir da segunda coluna, o meio somador não é suficiente, pois há a possibilidade de haver três bits envolvidos na soma caso ocorra "vai um" da coluna anterior. Portanto, precisamos de um circuito aritmético com três entradas e duas saídas.

O circuito com a tabela verdade representada a seguir resolve o problema.

Entradas			Saídas	
A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Na tabela:

- A e B são dígitos dos binários somados;
- C_i é o *carry in*, "vai um" da coluna anterior – entrada no somador;
- C_o é o *carry out*, "vai um" – saída no somador;
- C_i é saída do somador anterior;
- C_o é entrada do somador seguinte.

Analisando a tabela, temos:

$$S = \overline{A}\,\overline{B}\,C_i + \overline{A}\,B\,\overline{C_i} + A\,\overline{B}\,\overline{C_i} + A\,B\,C_i$$

$$C_0 = \overline{A}\,B\,C_i + A\,\overline{B}\,C_i + A\,B\,\overline{C_i} + A\,B\,C_i$$

Passando para o mapa de Karnaugh (figura 3.29).

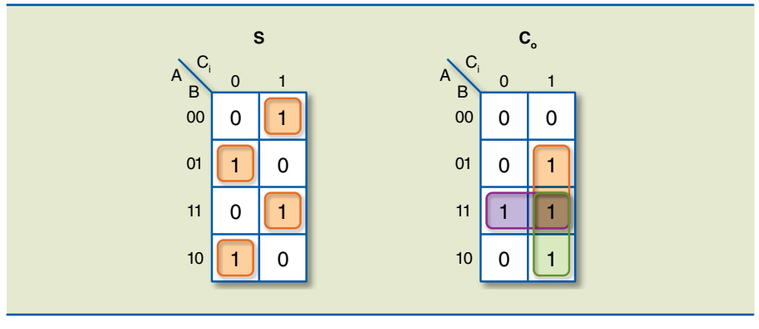


Figura 3.29
Mapa de Karnaugh para S e C_o.

Analisando o mapa de Karnaugh da figura 3.29, podemos notar que esse caso não admite simplificação, pois temos o OU EXCLUSIVO das três entradas:

$$S = A \oplus B \oplus C_i$$

$$C_o = A \cdot B + B \cdot C_i + A \cdot C_i$$

Simplificado pelo mapa de Karnaugh da figura 3.30, temos o AND da combinação duas a duas das entradas.

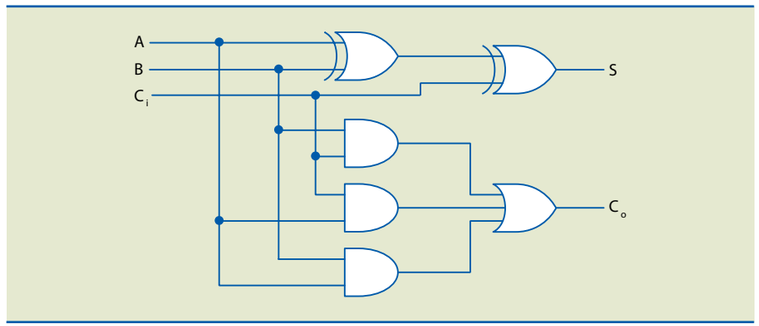


Figura 3.30
Somador completo (SC).

Podemos representar a saída $S = A + B + C_i$ por uma única porta OU EXCLUSIVO de três entradas, em nada alterando o circuito em si, apenas sua representação (figuras 3.31 e 3.32).

Figura 3.31
Somador completo (SC).

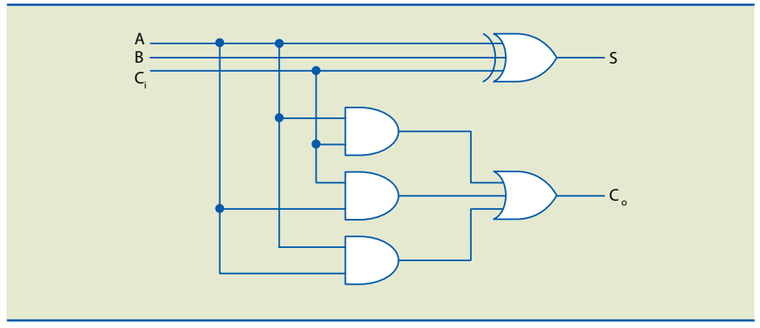
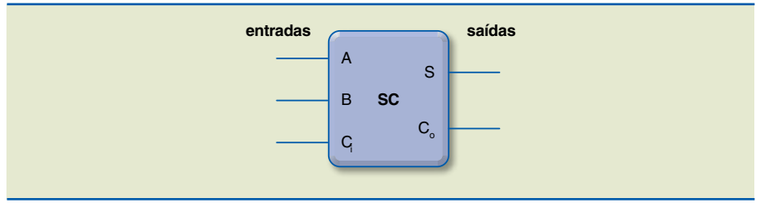


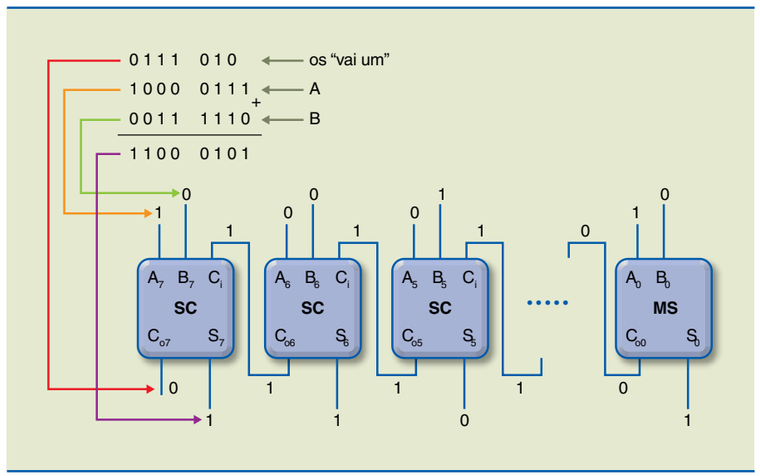
Figura 3.32
Representação simplificada do somador completo (SC).



Para somarmos dois binários, cada um formado por vários bits, podemos usar para a primeira coluna um meio somador, pois para essa coluna não existe *carry in* ("vai um anterior"). Para as demais colunas, empregamos somadores completos.

A figura 3.33 apresenta como exemplo a soma dos binários A + B, sendo A = (1000 0111) e B = (0011 1110).

Figura 3.33
Associação de meio somador com somador completo.



Podemos substituir o meio somador por um somador completo tendo $C_i = 0$. Dessa maneira, o funcional do circuito continua o mesmo, pois o MS equivale ao CS se $C_i = 0$.

3.3.3 Subtrator

Vamos relembrar na tabela seguir a tabela verdade da porta OU EXCLUSIVO.

OU EXCLUSIVO		
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

B = 0 S = A

B = 1 S = A

Analisando a tabela, podemos constatar que, se uma entrada é mantida em "0", a saída corresponde a outra entrada e, se uma entrada é mantida em "1", a saída corresponde ao complemento da outra entrada (porta INVERSORA).

Consideremos o circuito da figura 3.34, em que o MS foi substituído por um SC. Os bits do binário B são mantidos ou complementados, dependendo da variável de controle V.

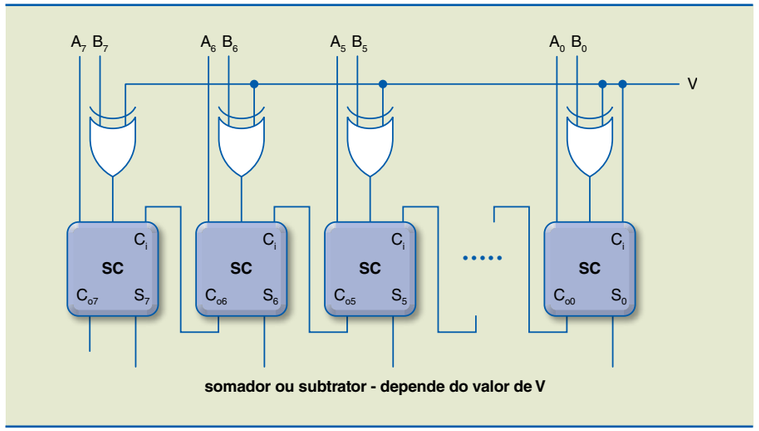


Figura 3.34
Somador ou subtrator – depende do valor de V.

Quando $V = 0$, o circuito é um somador com o mesmo funcional do circuito da figura 3.33, pois a entrada dos blocos do circuito é a mesma. Se $V = 1$, as en-