

### Capítulo 3 - Álgebra de proposições - Projeto de Circuitos

#### Objetivos

- Apresentar os conceitos básicos da álgebra de proposições;
- Mostrar as principais propriedades da álgebra de proposições;
- Estudar expressões e circuitos lógicos.

#### Álgebra de proposições

Em Matemática, chama-se **proposição** ao enunciado de uma verdade que se quer demonstrar, ou como usaremos: uma sentença que pode ser falsa (0), ou verdadeira (1), mas nunca ambos ao mesmo tempo.

#### - Correspondência entre as principais relações e portas lógicas

A conjunção determina que se duas proposições (p e q) forem verdadeiras (1), a conjunção de ambas também o será; basta que uma delas seja falsa (0), para que a conjunção (s) também o seja. A porta **AND** ( E ) implementa essa relação, pode ter duas (p, q), ou mais entradas, e a saída (s) assumirá o valor 1 se, e somente se, todas as entradas forem iguais a 1; caso uma, ou mais entradas sejam iguais a 0, a saída terá valor 0.

A disjunção determina que se duas proposições (p e q) forem falsas (0), a disjunção de ambas (s) também o será; basta que uma delas seja verdadeira (1), para que a disjunção também o seja. A porta **OR** ( OU ) implementa essa relação, pode ter duas (p, q), ou mais entradas, e a saída (s) assumirá o valor 0 se, e somente se, todas as entradas forem iguais a 0; caso uma, ou mais entradas forem iguais a 1, a saída terá valor 1.

A negação determina que se uma proposição (p) for falsa (0), a negação (s) será verdadeira (1), ou vice-versa. A porta **NOT** (NÃO) implementa essa relação, é também chamada de **INVERTER** (INVERSOR) e só tem uma entrada (p), e a saída assumirá o valor 1, se a entrada for igual a 0; senão, a saída terá valor 0.

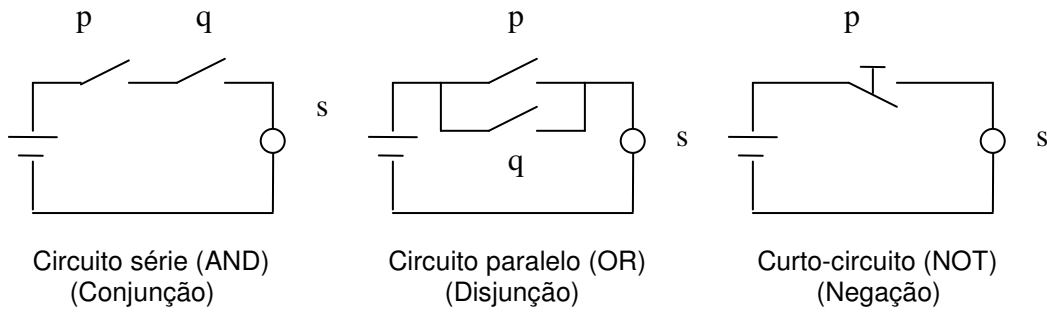
#### - Analogias com circuitos elétricos

O primeiro circuito a seguir (conjunção) determina que se duas chaves (p e q) forem fechadas (1), o resultado (s) será o de um circuito fechado com uma lâmpada acesa (1), por exemplo; basta que uma delas seja aberta (0), para que o circuito se abra, e a lâmpada apague (0). O circuito poderá ter duas (p, q), ou mais chaves, em série que a saída (s) terá o mesmo resultado (1) se, e somente se, todas as chaves forem fechadas (1); caso uma, ou mais chaves forem abertas (0), o resultado será um circuito aberto com a lâmpada apagada (0).

O segundo circuito a seguir (disjunção) determina que se duas chaves (p e q) forem abertas (0), o resultado (s) será o de um circuito aberto com uma lâmpada apagada (0), por exemplo; basta que uma delas seja fechada (1), para que o circuito se feche. O circuito poderá ter duas (p, q), ou mais chaves, em paralelo que a saída (s) terá o mesmo resultado (0), se, e somente se, todas as entradas forem abertas (0); caso uma, ou mais chaves forem fechadas (1), o resultado será um circuito fechado com a lâmpada acesa (1).

O terceiro circuito a seguir (negação) determina que se uma chave (p) for acionada (1), o resultado (s) será o de um circuito aberto com uma lâmpada apagada (0); caso contrário, o circuito permanecerá fechado, e a lâmpada se manterá acesa (1) .

## - Representações de circuitos



## - Representações de relações lógicas

Conjunção  
( $p$  e  $q$ )

$$p \wedge q$$

$$p \cdot q \text{ ou } p q$$

$p$	$q$	$s$
0	0	= 0
0	1	= 0
1	0	= 0
1	1	= 1

Disjunção  
( $p$  ou  $q$ )

$$p \vee q$$

$$p + q$$

$p$	$q$	$s$
0	0	= 0
0	1	= 1
1	0	= 1
1	1	= 1

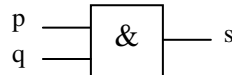
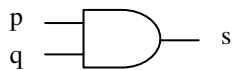
Negação  
(não  $p$ )

$$\neg p$$

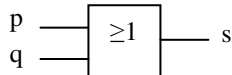
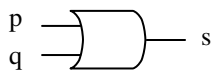
$$/p \text{ ou } \overline{p} \text{ ou } p'$$

$p$	$s$
0	= 1
1	= 0

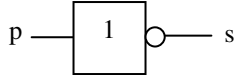
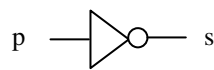
Porta AND ( E )



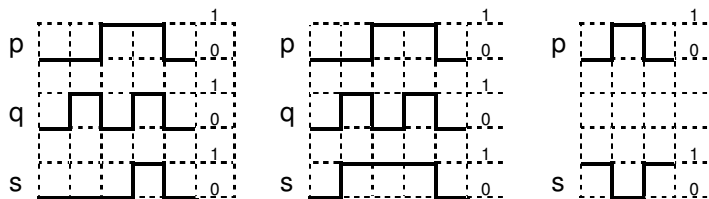
Porta OR ( OU )



Porta NOT (NÃO)



## Notações para portas lógicas



## Diagramas de tempo para as portas lógicas

### - Prioridade de conectivos

Estabelece-se que a ordem de avaliação de uma expressão, envolvendo conectivos lógicos, será da esquerda para a direita, respeitando-se as prioridades dos conectivos na ordem mostrada abaixo, sendo a primeira a mais alta quando aplicada imediatamente a um valor.

NÃO  
E  
OU

Pode-se mudar a ordem de avaliação por meio de parênteses.

Exemplo :

Considere a expressão :  $x + y' \cdot z$

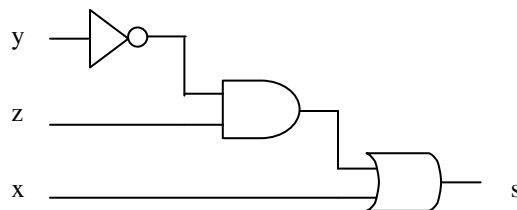
A sua avaliação será feita na seguinte ordem de prioridade:

- negação de (y) :  $y'$
- conjunção com (z) :  $y' \cdot z$
- disjunção com (x) :  $(y' \cdot z) + x$

A expressão também poderá ser representada na forma tabular (**tabela-verdade**):

x y z	$y'$	$(y' \cdot z)$	$(y' \cdot z) + x$
0 0 0	1	0	0
0 0 1	1	1	1
0 1 0	0	0	0
0 1 1	0	0	0
1 0 0	1	0	1
1 0 1	1	1	1
1 1 0	0	0	1
1 1 1	0	0	1

A representação por um circuito lógico poderá ser a mostrada abaixo.



Se fosse desejado que a operação de disjunção ocorresse antes da conjunção, seria necessário o uso de parênteses, e o circuito seria diferente.

$$(x + y') \cdot z$$

Se fosse desejado negar toda a expressão, também se usaria parênteses, e esta ação seria a última a ser avaliada. O circuito resultante seria o mesmo acima com mais um inversor.

$$(x + y' \cdot z)'$$

Exercício - Construir tabela e circuito para as proposições:

- a)  $p + \bar{q}$
- b)  $\bar{p} \cdot \bar{q}$
- c)  $(p \cdot \bar{q}) + r$
- d)  $\overline{(p \cdot q)}$
- e)  $\overline{(p + q)}$
- f)  $(p + \bar{r})(q + \bar{r})$

- Principais propriedades

Idempotência $p + p = p$ $p \cdot p = p$	Comutativa $p + q = q + p$ $p \cdot q = q \cdot p$	Associativa $(p+q)+r = p+(q+r)$ $(p \cdot q) \cdot r = p \cdot (q \cdot r)$
Distributiva $p+(q \cdot r)=(p+q) \cdot (p+r)$ $p \cdot (q+r)=(p \cdot q)+(p \cdot r)$	Absorção $p + (\bar{p} \cdot q) = (p+q)$ $\bar{p} + (p \cdot q) = (\bar{p} + q)$ $p + (p \cdot q) = (p+q)$	Identidade $p+0 = p$ $p \cdot 0 = 0$ $p+1 = 1$ $p \cdot 1 = p$
Complementar $p + \bar{p} = 1$ (tautologia) $p \cdot \bar{p} = 0$ (contradição)	De Morgan $\overline{(p + q)} = \bar{p} \cdot \bar{q}$ $\overline{(p \cdot q)} = \bar{p} + \bar{q}$	

Observação:

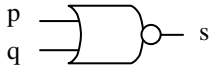
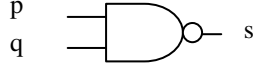
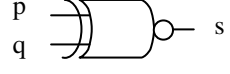
As regras de absorção são aplicadas, em geral, para se efetuar simplificações; normalmente, ao se levar em conta a precedência de operadores, a conjunção tem prioridade sobre a disjunção e as regras de absorção não se aplicam.

Exercício - Simplificar pelas propriedades da álgebra:

- a)  $(p \cdot q)'$
- b)  $(p' + q)'$
- c)  $((p \cdot q)' + (q \cdot p'))'$
- d)  $(p \cdot q) + (p' \cdot (p + q'))$
- e)  $(p + q)' \cdot (p \cdot q)$
- f)  $(p \cdot q) \cdot (p' + q')' + r$

- Outras relações lógicas importantes

É comum usar as negações das portas principais e definir outras relações lógicas:

Porta NOR		Porta NAND		Porta XNOR (NEXOR)	
					
$p \ q$	$(p + q)'$	$p \ q$	$(p \cdot y)'$	$p \ q$	$(p \text{ xor } y)'$
0 0	1	0 0	1	0 0	1
0 1	0	0 1	1	0 1	0
1 0	0	1 0	1	1 0	0
1 1	0	1 1	0	1 1	1

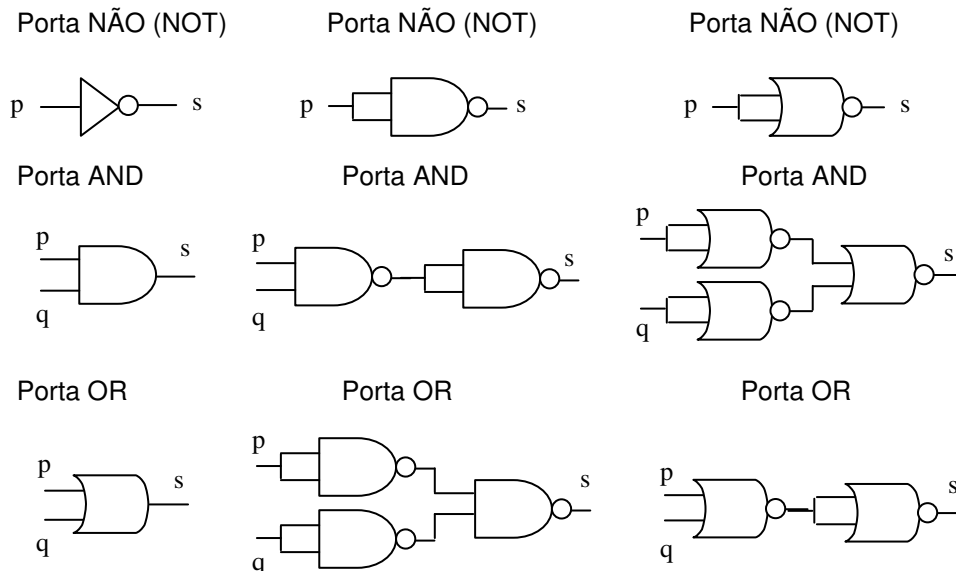
Além delas, também se pode definir:

$p \ q$	$p \rightarrow q$	$= (p' + q)$	$(p \text{ implica } q)$	$p \Leftrightarrow q$	$= (p \rightarrow q) \cdot (q \rightarrow p)$	$(p \text{ equivale a } q)$
0 0	1	(se p então q) ou (q, se p) ou	1	1	(se p então q, e se q então p) ou	
0 1	1	(q, dado p) ou	0	0	(q é condição necessária	
1 0	0	(p é condição suficiente para q) ou	0	0	e suficiente para p)	
1 1	1	(q é condição necessária para p)	1	1		

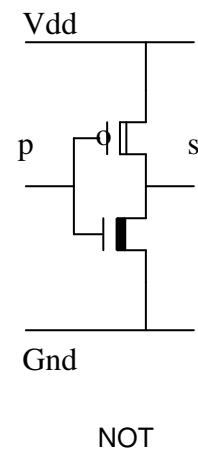
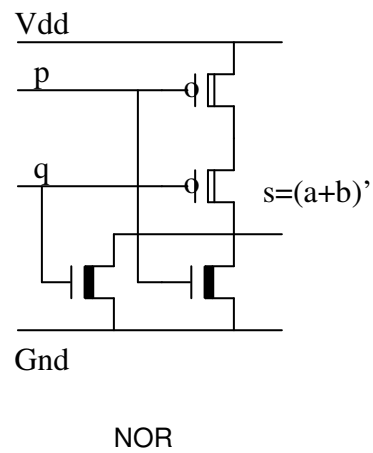
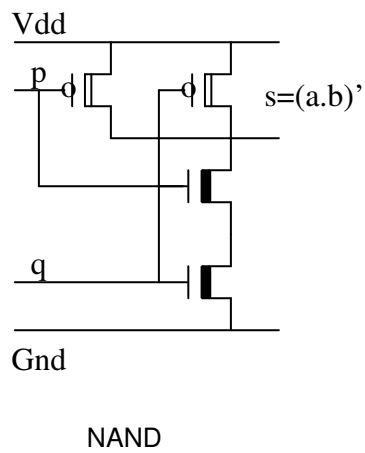
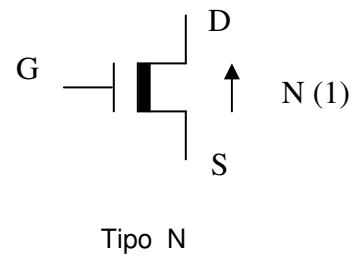
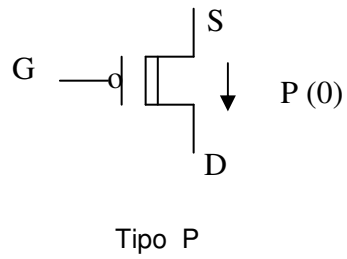
A prioridade dessas últimas relações normalmente é inferior às das anteriores.

- Universalidade das portas NAND e NOR

A universalidade das portas NAND e NOR permite que todas as funções lógicas básicas possam ser substituídas por composições equivalentes, como se mostrará a seguir.

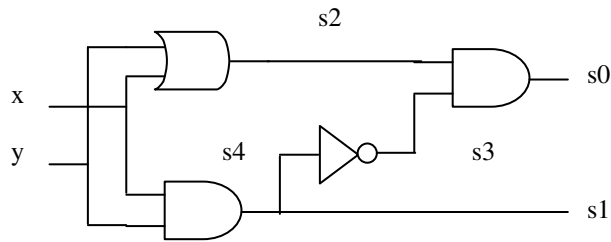


## - Analogias com transistores (CMOS)



## Aplicações aritméticas de circuitos lógicos

Dado o circuito lógico:



As relações abaixo descrevem os sinais de saída em função dos sinais de entrada:

$$s0 = s2 \cdot s3 = s2 \cdot s4' = (x + y) \cdot (x \cdot y)' \quad (1)$$

$$s1 = s4 = x \cdot y \quad (2)$$

$$s2 = x + y$$

$$s3 = s4'$$

$$s4 = x \cdot y$$

A partir das relações (1) e (2) pode-se construir a tabela-verdade:

x y	x + y	x • y	(x•y)'	(x+y) • (x•y)'
0 0	0	0	1	0
0 1	1	0	1	1
1 0	1	0	1	1
1 1	1	1	0	0

O resultado pode ser apresentado de outra forma:

x + y =	s1 s0
0 + 0 =	0 0
0 + 1 =	0 1
1 + 0 =	0 1
1 + 1 =	1 0

ou seja, o circuito é o responsável por uma soma binária de dois bits, e é chamado de circuito de **meia-soma**. Para operar três ou mais bits, vários destes, organizados em cascata, formariam um circuito de **soma-completa**.

A equação que definirá o circuito poderá ser simplificada pelas propriedades da álgebra:

$$(x + y) \cdot (x \cdot y)'$$

De Morgan

$$(x + y) \cdot (x' + y')$$

Distributiva

$$x \cdot x' + y \cdot x' + x \cdot y' + y \cdot y'$$

Complementar

$$0 + y \cdot x' + x \cdot y' + 0$$

Associativa

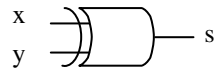
$$(0 + y \cdot x') + (x \cdot y' + 0)$$

Identidade

$$y \cdot x' + x \cdot y'$$

A relação encontrada descreve uma outra porta lógica - **OU exclusivo** - cuja representação encontra-se abaixo:

Porta OU-exclusivo (XOR)



x	y	$x \oplus y$	mintermos	MAXTERMOS	N
0	0	0	0	$(X'+Y')$	0
0	1	1	$(x' \cdot y)$	1	1
1	0	1	$(x \cdot y')$	1	2
1	1	0	0	$(X + Y)$	3

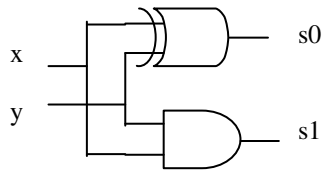
Essa relação poderá ser definida por uma soma dos produtos (SoP):

$$0 + (x' \cdot y) + (x \cdot y') + 0 = (x' \cdot y) + (x \cdot y') = \sum (1, 2)$$

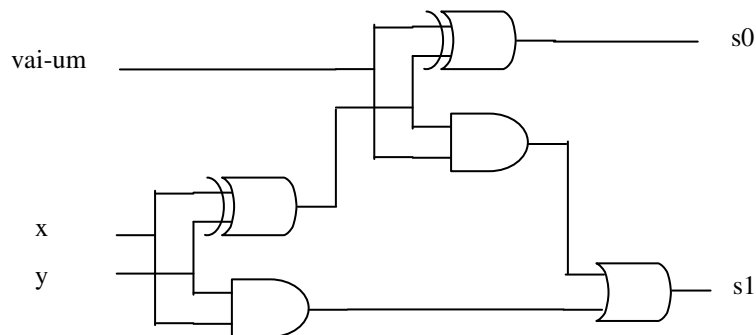
ou definida por um produto das somas (PoS):

$$(X'+Y') \cdot 1 \cdot 1 \cdot (X+Y) = (X' \cdot X) + (X' \cdot Y) + (Y' \cdot X) + (Y' \cdot Y) = (X' \cdot Y) + (Y' \cdot X) = \prod (0, 3)$$

O circuito de **meia-soma** poderá ser feito com a porta **XOR** :



O circuito de **soma-completa** também poderá ser feito com essa porta:





As relações expressas pelo circuito de **soma-completa** podem ser encontradas abaixo:

vai-um + x + y	s1	s0	mintermos	MAXTERMS	N
0 0 0	0	0	$(v' \cdot x' \cdot y')$	$(V + X + Y)$	0
0 0 1	0	1	$(v' \cdot x' \cdot y)$	$(V + X + Y')$	1
0 1 0	0	1	$(v' \cdot x \cdot y')$	$(V + X' + Y)$	2
0 1 1	1	0	$(v' \cdot x \cdot y)$	$(V + X' + Y')$	3
1 0 0	0	1	$(v \cdot x' \cdot y')$	$(V' + X + Y)$	4
1 0 1	1	0	$(v \cdot x' \cdot y)$	$(V' + X + Y')$	5
1 1 0	1	0	$(v \cdot x \cdot y')$	$(V' + X' + Y)$	6
1 1 1	1	1	$(v \cdot x \cdot y)$	$(V' + X' + Y')$	7

A relação ( **s0** ) poderá ser definida por uma soma dos produtos (SoP):

$$(v' \cdot x' \cdot y) + (v' \cdot x \cdot y') + (v \cdot x' \cdot y') + (v \cdot x \cdot y) = \sum(1, 2, 4, 7)$$

ou definida por um produto das somas (PoS):

$$(V + X + Y) \cdot (V + X' + Y') \cdot (V' + X + Y') \cdot (V' + X' + Y) = \prod(0, 3, 5, 6)$$

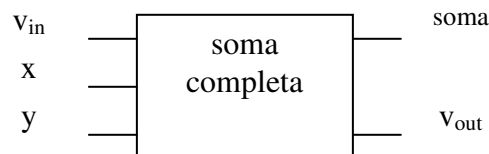
A relação ( **s1** ) também poderá ser definida por uma soma dos produtos (SoP):

$$(v' \cdot x' \cdot y) + (v' \cdot x \cdot y') + (v \cdot x' \cdot y') + (v \cdot x \cdot y) = \sum(3, 5, 6, 7)$$

ou definida por um produto das somas (PoS):

$$(V + X + Y) \cdot (V + X' + Y') \cdot (V' + X + Y') \cdot (V' + X' + Y) = \prod(0, 1, 2, 4)$$

O diagrama de blocos descrito a seguir resume esse circuito:



A diferença entre dois **bits** também pode ser projetada de modo semelhante.

x - y	d	v	mintermos	MAXTERMOS	N
0 0	0	0	$(x' \cdot y')$	$(X + Y)$	0
0 1	1	1	$(x' \cdot y)$	$(X + Y')$	1
1 0	1	0	$(x \cdot y')$	$(X' + Y)$	2
1 1	0	0	$(x \cdot y)$	$(X' + Y')$	3

A diferença ( d ) poderá ser definida por uma soma dos produtos (SoP):

$$(x' \cdot y) + (x \cdot y') = \sum (1, 2)$$

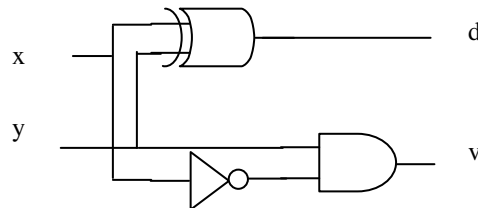
ou definida por um produto das somas (PoS):

$$(X+Y) \cdot (X'+Y') = \prod (0, 3)$$

A necessidade de empréstimo (“vem-um”) poderá ser expressa pela relação:

$$(x' \cdot y) = v$$

O circuito de **meia-diferença** é mostrado a seguir:



O circuito de **diferença-completa** (x-y-empréstimo) terá a definição abaixo:

x - y - vem-um	s0	s1	mintermos	MAXTERMOS	N
0 0 0	0	0	$(x' \cdot y' \cdot v')$	$(X + Y + V)$	0
0 0 1	1	1	$(x' \cdot y' \cdot v)$	$(X + Y + V')$	1
0 1 0	1	1	$(x' \cdot y \cdot v')$	$(X + Y' + V)$	2
0 1 1	0	1	$(x' \cdot y \cdot v)$	$(X + Y' + V')$	3
1 0 0	1	0	$(x \cdot y' \cdot v')$	$(X' + Y + V)$	4
1 0 1	0	0	$(x \cdot y' \cdot v)$	$(X' + Y + V')$	5
1 1 0	0	0	$(x \cdot y \cdot v')$	$(X' + Y' + V)$	6
1 1 1	1	1	$(x \cdot y \cdot v)$	$(X' + Y' + V')$	7

A relação ( **s0** ) também poderá ser definida pela soma dos produtos (SoP):

$$(x' \cdot y' \cdot v) + (x' \cdot y \cdot v') + (x \cdot y' \cdot v') + (x \cdot y \cdot v) = \sum m(1, 2, 4, 7)$$

ou ainda definida pelo produto das somas (PoS):

$$(X + Y + V) \cdot (X + Y' + V') \cdot (X' + Y + V') \cdot (X' + Y' + V) = \prod M(0, 3, 5, 6)$$

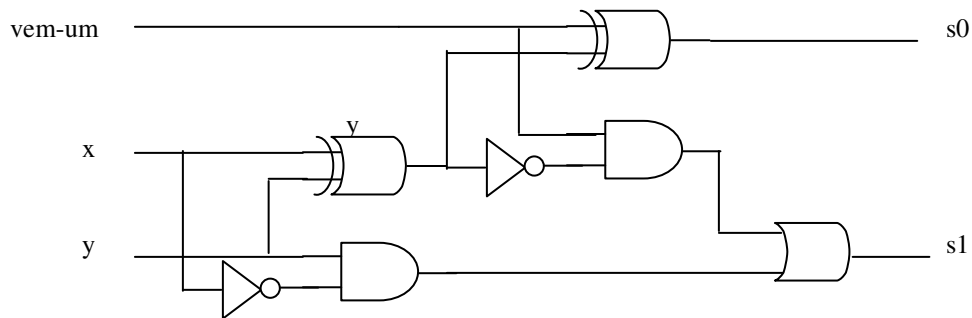
A relação ( **s1** ) poderá ser definida pela soma dos produtos (SoP):

$$(x' \cdot y' \cdot v) + (x' \cdot y \cdot v') + (x \cdot y' \cdot v) + (x \cdot y \cdot v') = \sum m(1, 2, 3, 7)$$

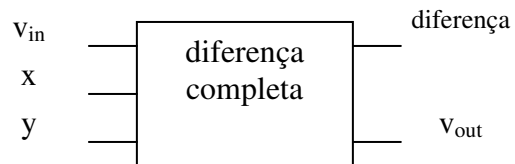
ou definida por um produto das somas (PoS):

$$(X + Y + V) \cdot (X' + Y + V) \cdot (X' + Y' + V') \cdot (X' + Y' + V) = \prod M(0, 4, 5, 6)$$

O circuito equivalente é mostrado abaixo.



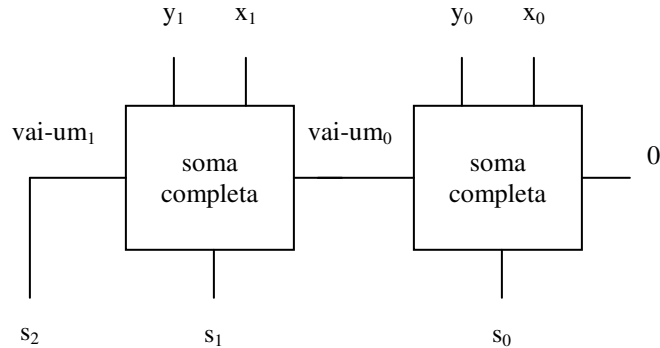
O diagrama de blocos descrito a seguir resume esse circuito:



## Operações aritméticas em paralelo

Circuitos aritméticos podem ser combinados para operarem em paralelo.

Um circuito para a adição de dois pares de **bits** poderia ser o descrito abaixo:

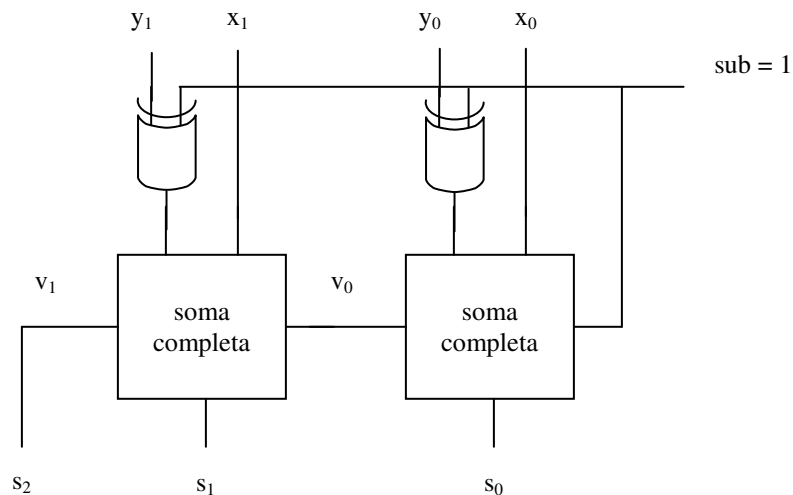


Um circuito para a subtração de dois pares de **bits** também pode usar o esquema acima. Entretanto, se for desejada uma otimização dos recursos envolvidos, o próprio circuito de adição poderá realizar a soma algébrica desses bits através da implementação da regra para a operação de soma utilizando o complemento de 2. Nesse caso, a saída de cada bloco implementará a relação:

$$s = x + \text{inverso}(y) + \text{"vai-um"}$$

para prover a soma adicional de uma unidade ao **bit** mais à direita ( bit<sub>0</sub> ), a entrada em ( 0 ) do primeiro bloco à direita deverá ser alterada para ( 1 ). Esse valor poderá ser fornecido por uma linha adicional de controle que também servirá para tomar os inversos dos valores dos subtraendos ( y<sub>i</sub> ), os valores dos minuendos ( x<sub>i</sub> ) deverão ser tomados sem outras alterações.

O circuito abaixo ilustra essas modificações.



Simplificação por agregação de produtos vizinhos adjacentes (mapas de Veitch-Karnaugh)

Expressões do tipo  $X \cdot Y + X \cdot Y'$  podem ser simplificadas pelos mapas de Veitch-Karnaugh.

a\b	0	1
0	a'b'	a'b
1	a b'	a b

2 variáveis

a\b	00	01	11	10
0	a'b'c'	a'b'c	a'b c	a'b c'
1	a b'c'	a b'c	a b c	a b c'

3 variáveis

a\b\c	00	01	11	10
00	a'b'c'd'	a'b'c'd	a'b'c d	a'b'c d'
01	a'b'c d'	a'b'c d	a' b c d	a'b c d'
11	a b c d'	a b c d	a b c d	a b c d'
10	a b'c d'	a b'c d	a b'c d	a b'c d'

4 variáveis

## Simplificação pelo método de Quine-McCluskey

Mapas de Veitch-Karnaugh podem ser aplicados para até 5 ou 6 variáveis, mas para um número maior outros métodos deverão ser aplicados. Um método que pode ser implementado de maneira sistemática é o de Quine-McCluskey, o qual pode ser resumido pelos seguintes passos:

- produzir uma expansão de uma função na forma de soma de produtos (SoP);
- reduzir ao máximo os termos do tipo  $X \cdot Y + X \cdot Y'$  ;
- identificar um conjunto mínimo de fatores **primos implicantes** equivalente à função.

Exemplo:

Dada a função abaixo, identificar um conjunto de fatores **primos implicantes**.

$$f(a, b, c, d) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$$

1º passo: Identificar os grupos que possuem a mesma quantidade de 1's:

N	a	b	c	d	f(a,b,c,d)	Termos	N	Grupos	
0	0	0	0	0	1	0 0 0 0	0	0 0 0 0	0 bit em 1
1	0	0	0	1	1	0 0 0 1			
2	0	0	1	0	1	0 0 1 0	1	0 0 0 1	1 bit em 1
3	0	0	1	1	0		2	0 0 1 0	
4	0	1	0	0	0		8	1 0 0 0	
5	0	1	0	1	1	0 1 0 1			
6	0	1	1	0	1	0 1 1 0	5	0 1 0 1	2 bits em 1
7	0	1	1	1	1	0 1 1 1	6	0 1 1 0	
8	1	0	0	0	1	1 0 0 0	9	1 0 0 1	
9	1	0	0	1	1	1 0 0 1	10	1 0 1 0	
10	1	0	1	0	1	1 0 1 0			
11	1	0	1	1	0		7	0 1 1 1	3 bits em 1
12	1	1	0	0	0		14	0 1 1 1	
13	1	1	0	1	0				
14	1	1	1	0	1	1 1 1 0			
15	1	1	1	1	0				

2º passo: Agrupar os termos com os mesmos bits de diferença:

N	Grupos		Grupos I	1 bit		Grupos II	2 bits	OBS.:
0	0 0 0 0	#	0, 1	0 0 0 X	#	0, 1, 8, 9	X 0 0 X	
			0, 2	0 0 X 0	#	0, 2, 8, 10	X 0 X 0	
1	0 0 0 1	#	0, 8	X 0 0 0	#	0, 8, 1, 9		repetido
2	0 0 1 0	#				0, 8, 2, 10		repetido
8	1 0 0 0	#	1, 5	0 X 0 1				
			1, 9	X 0 0 1	#	2, 6, 10, 14	X X 1 0	
5	0 1 0 1	#	2, 6	0 X 1 0	#	2, 10, 6, 14		repetido
6	0 1 1 0	#	2, 10	X 0 1 0	#			
9	1 0 0 1	#	8, 9	1 0 0 X	#			
10	1 0 1 0	#	8, 10	1 0 X 0	#			
7	0 1 1 1	#	5, 7	0 1 X 1				
14	1 1 1 0	#	6, 7	0 1 1 X				
			6, 14	X 1 1 0	#			
			10, 14	1 X 1 0	#			

3º passo: Agrupar Identificar os termos não utilizados em algum agrupamento:

$$f(a, b, c, d) = (1,5) + (5,7) + (6,7) + (0,1,8,9) + (0,2,8,10) + (2,6,10,14) \\ = a'c'd + a'b d + a' b c + b'c' + b'd' + c d'$$

4º passo: Montar a tabela de primos implicantes (não cobertos por outros termos):

primos	mintermos	0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	$b' c'$	X	X					X	X		
0, 2, 8, 10	$b' d'$	X		X				X		X	
2, 6, 10, 14	$c d'$			X		X				X	X
1, 5	$a' c' d$		X		X						
5, 7	$a' b d$				X		X				
6, 7	$a' b c$					X	X				

5º passo: Identificar os mintermos cobertos por apenas um único conjunto de primos:

primos	mintermos	0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	$b' c'$	X	X					X	X		
0, 2, 8, 10	$b' d'$	X		X				X		X	
2, 6, 10, 14	$c d'$			X		X				X	X
1, 5	$a' c' d$		X		X						
5, 7	$a' b d$				X		X				
6, 7	$a' b c$					X	X				

Há dois mintermos essenciais que satisfazem a condição:  $(b'c')$  e  $(c d')$   
e, por isso, devem estar presentes na resposta:  $f(a,b,c,d) = (b'c') + (c d') + \dots$

6º passo: Eliminar todos os termos redundantes cobertos por esses:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	$b' c'$	#	X	X					X	X		
0, 2, 8, 10	$b' d'$		X		X				X		X	
2, 6, 10, 14	$c d'$	#			X		X				X	X
1, 5	$a' c' d$			X		X						
5, 7	$a' b d$				X		X					
6, 7	$a' b c$					X	X					

7º passo: Escolher outros mintermos que possam eliminar redundâncias:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	$b' c'$											
0, 2, 8, 10	$b' d'$											
2, 6, 10, 14	$c d'$											
1, 5	$a' c' d$					X						
5, 7	$a' b d$	#				X		X				
6, 7	$a' b c$							X				

A função equivalente será dada por:  $f(a,b,c,d) = (b'c') + (c d') + (a' b d)$

Um inconveniente deve ser indicado: devido ao crescimento exponencial do número de combinações, uma solução encontrada pelo método de Quine-McCluskey pode não ser única.

Uma maneira de demonstrar a afirmação anterior é aplicando o método de Petrick:

1º passo: Eliminar as linhas com os mintermos essenciais e colunas correspondentes:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
0, 1, 8, 9	$b' c'$	#	X	X					X	X		
0, 2, 8, 10	$b' d'$		X		X				X		X	
2, 6, 10, 14	$c d'$	#			X		X				X	X
1, 5	$a' c' d$			X		X						
5, 7	$a' b d$					X		X				
6, 7	$a' b c$						X	X				

2º passo: Rotular as linhas restantes:

primos	mintermos		0	1	2	5	6	7	8	9	10	14
1, 5	$a' c' d$	K				X						
5, 7	$a' b d$	L				X		X				
6, 7	$a' b c$	M						X				

3º passo: Construir a soma de produtos das linhas adicionando as colunas:

$$(K + L) \cdot (L + M)$$

4º passo: Simplificar a soma de produtos ( $X + XY = X$ ):

$$K.L + L.L + K.M + L.M = K.L + L + K.M + L.M = L + K.M$$

5º passo: Escolher as soluções com as menores quantidades de termos:

$$L + K.M \rightarrow L$$

6º passo: Expandir a solução e contar o número de variáveis:

$$L = a' b d \rightarrow 3 \text{ termos}$$

7º passo: Escolher as soluções com as menores quantidades de variáveis:

$$L = (a' b d)$$

Após a escolha, montar a função equivalente final com todos os termos selecionados:

$$f(a,b,c,d) = (a' b d) + (b' c') + (c d')$$



## Codificadores e decodificadores

Seqüências de **bits** podem ser usadas para codificar valores numéricos. Dois códigos binários têm aplicações especiais: o BCD (Binary-Coded Decimal) e o de Gray.

## Código BCD

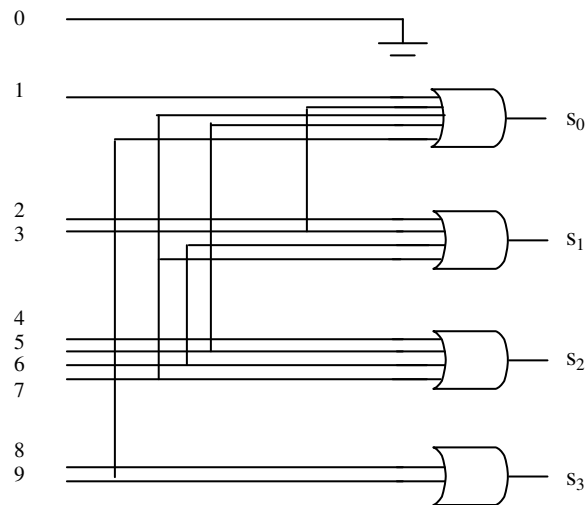
O código BCD é basicamente uma forma de codificar valores numéricos na base 10 em seus equivalentes binários.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

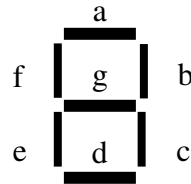
Exemplo:

Codificar o valor decimal 369 em código BCD:  $369 = (0011) (0110) (1001)$

Um circuito codificador “*decimal-para-BCD*” é aquele capaz de mapear um conjunto de entradas (0-9) em um outro conjunto de quatro valores binários ( $s_3s_2s_1s_0$ ), se apenas uma das entradas for acionada.



Um **display** de 7-segmentos pode ser usado para mostrar valores numéricos decimais codificados em BCD. Para isso, um decodificador de “BCD-para-7-segmentos” deve ser utilizado. O esquema a seguir identifica os sinais necessários (**abcdefg**).



A tabela abaixo indica as funções que representam o mapeamento dos sinais de entrada em BCD e as saídas no **display**.

Decimal	BCD	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	0	0	0	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1

### Código de Gray

O código de Gray BCD é forma de expressar seqüências binárias nas quais dois valores sucessivos tenham apenas um **bit** de diferença (distância de Hamming = 1).

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

Esse código também é conhecido com “*código binário refletido*”, por causa da característica abaixo:

eixo de reflexão		000
		001
	00	011
	0	010
	1	110
	10	111
		101
		100

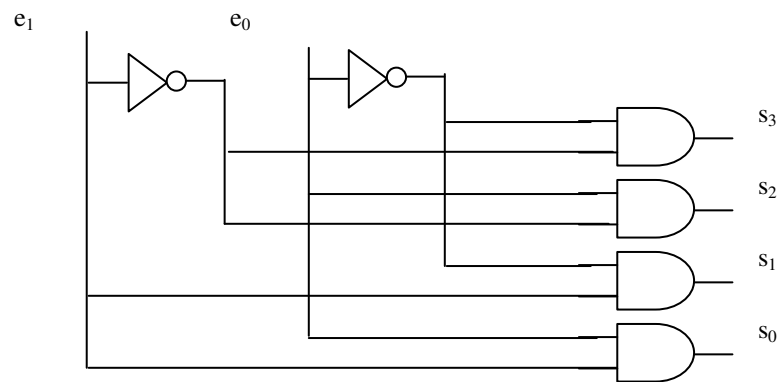
Decodificadores em geral são circuitos lógicos capazes de ativar uma saída de acordo com uma seleção de sinais de entrada.

Um decodificador de nível alto ativa uma saída quando uma das entradas estiver em nível 1 e as outras em nível 0.

Exemplo:

Montar um decodificador em nível alto para a tabela abaixo:

$e_1$	$e_0$	$s_3 s_2 s_1 s_0$
0	0	1 0 0 0
0	1	0 1 0 0
1	0	0 0 1 0
1	1	0 0 0 1

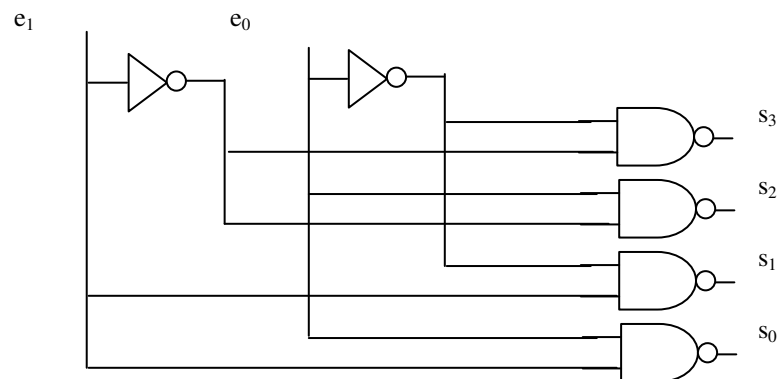


Um decodificador de nível baixo ativa uma saída quando uma das entradas estiver em nível 0 e as outras em nível 1.

Exemplo:

Montar um decodificador em nível baixo para a tabela abaixo:

$e_1$	$e_0$	$s_3 s_2 s_1 s_0$
0	0	0 1 1 1
0	1	1 0 1 1
1	0	1 1 0 1
1	1	1 1 1 0

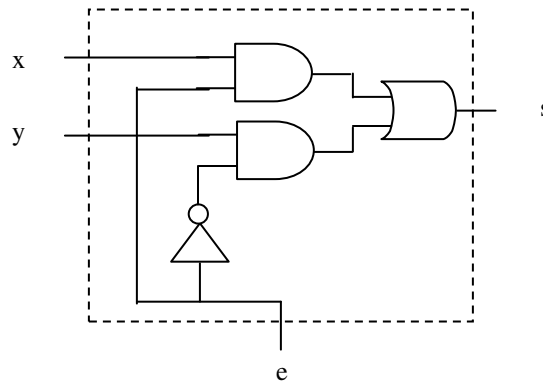


### Multiplexadores (MUX) e demultiplexadores (DEMUX)

Multiplexadores (ou seletor de dados) são circuitos lógicos capazes de atuar como chaves digitais: recebem várias entradas e selecionam uma delas, em um certo instante, e realizam sua transferência para a saída, mediante um código de seleção. Podem ser usados para rotear dados, seqüenciar operações, realizar conversões paralelo-série e gerar formas de ondas.

Exemplo:

Dados os sinais de entrada (  $x$  ) e (  $y$  ), escolher a saída mediante um sinal de seleção.

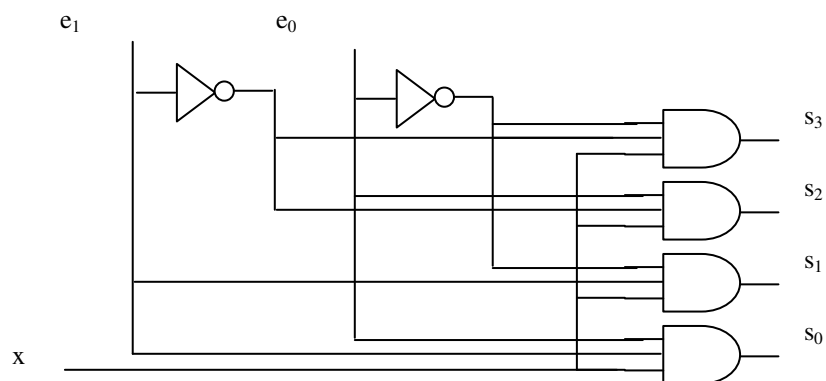


e	$s = x \cdot e + y \cdot e'$
0	y
1	x

Demultiplexadores (ou distribuidores de dados) são circuitos capazes de receber um sinal de entrada e distribuí-lo em várias saídas. Podem ser usados para distribuir um mesmo sinal de ativação ou seqüenciamento (**clock**) para vários circuitos.

Exemplo:

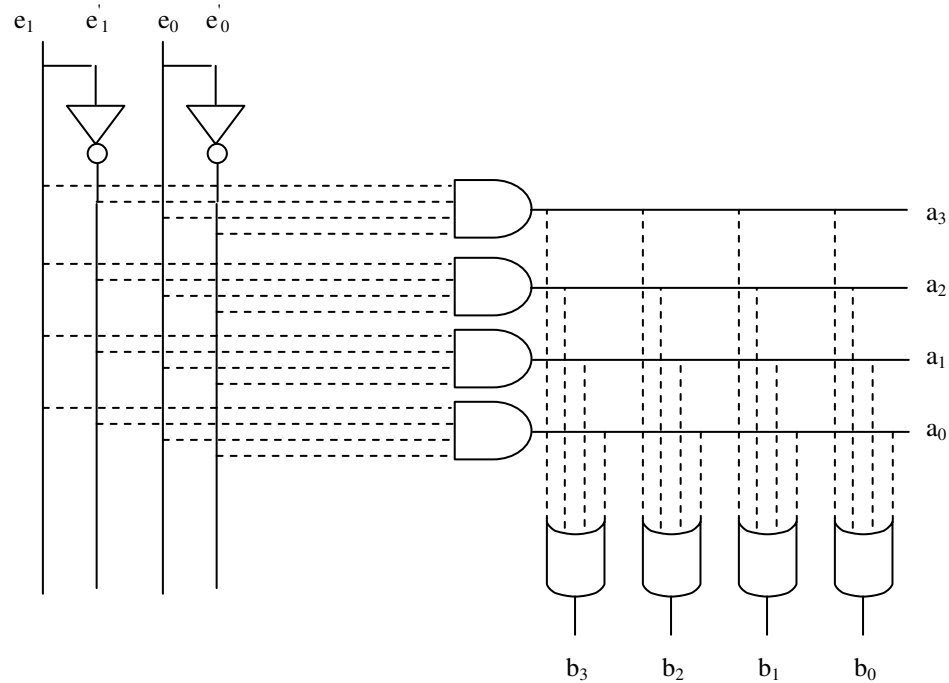
Dados um sinal de entrada (  $x$  ) e dois sinais de ativação  $e_0$  e  $e_1$ , distribuí-lo à saída.



$e_1$	$e_0$	$e_1 \ e_0 \ x$	$s_3 \ s_2 \ s_1 \ s_0$
0	0	$s_3 = 0 \ 0 \ X$	$X \ 0 \ 0 \ 0$
0	1	$s_2 = 0 \ 1 \ X$	$0 \ X \ 0 \ 0$
1	0	$s_1 = 1 \ 0 \ X$	$0 \ 0 \ X \ 0$
1	1	$s_0 = 1 \ 1 \ X$	$0 \ 0 \ 0 \ X$

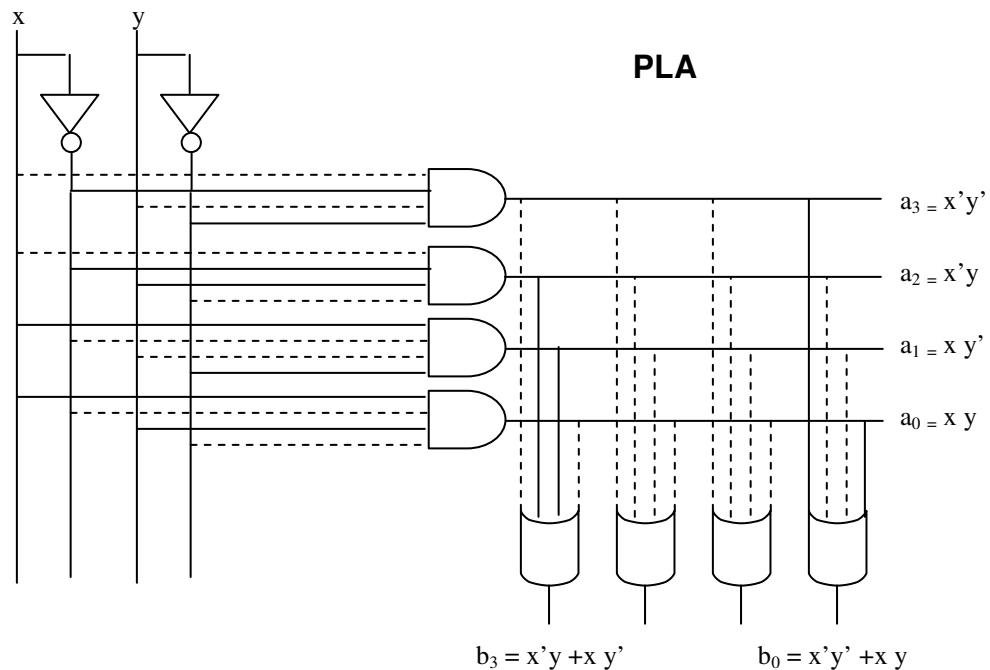
## Dispositivos lógicos programáveis

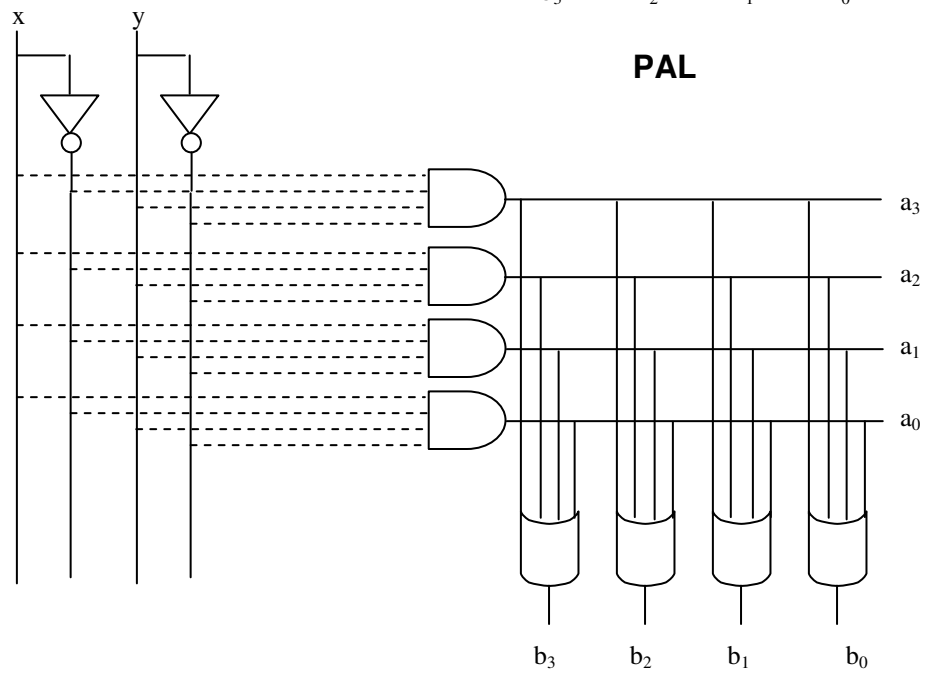
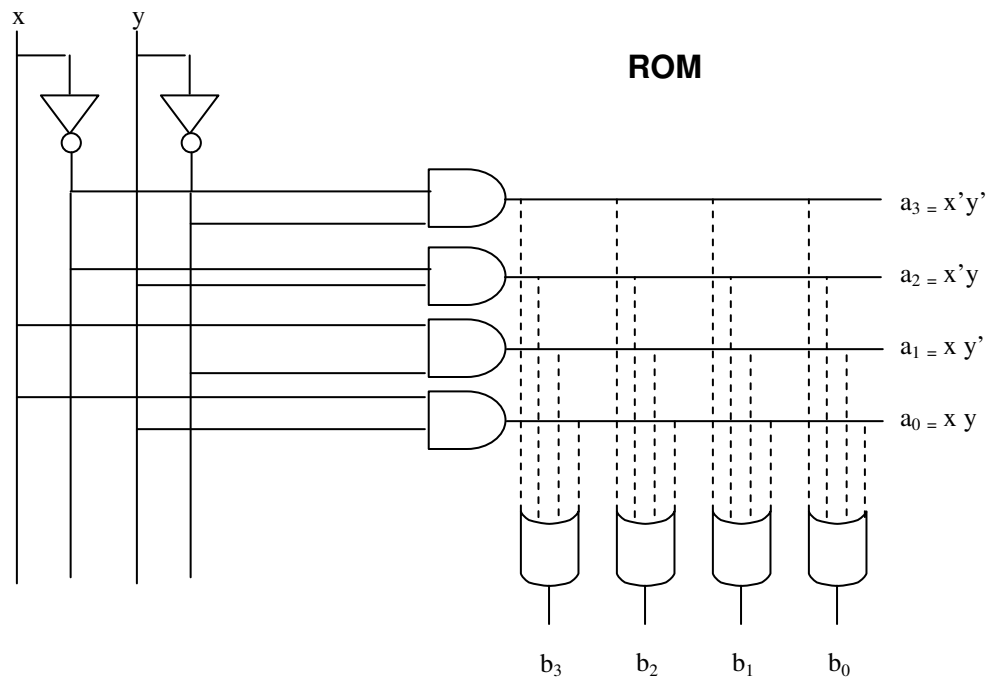
PLDs (**Programmable Logic Devices** ou FPGAs – **Field Programmable Gate Arrays**) são arranjos de portas lógicas que servem para uma rápida implementação de circuitos razoavelmente complexos. A figura abaixo ilustra uma organização de um arranjo de portas AND e OR.



Há três tipos comuns de arranjos lógicos combinacionais:

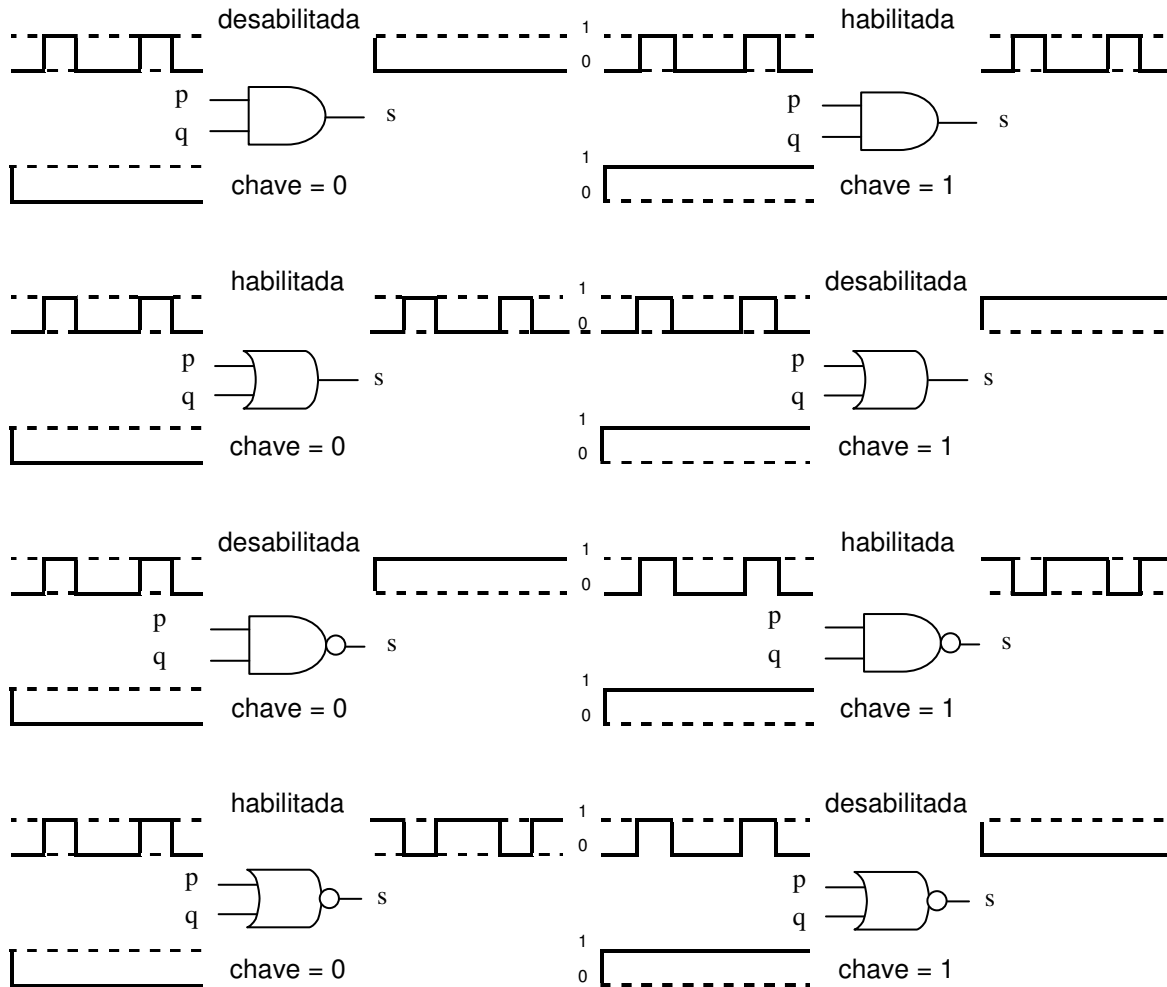
- PLA (**Programmable Logic Array**) - AND's e OR's programáveis
- ROM (**Read-Only Memory**) - AND's fixas e OR's programáveis
- PAL (**Programmable Array Logic**) - AND's programáveis e OR's fixas





## Circuitos com chaveamento

As portas lógicas básicas podem ser usadas para controlar a passagem (chavear) de um sinal de entrada (  $p$  ) para a saída, se a outra entrada (  $q$  ) for usada como o controle (chave).



## Exemplo:

Montar um *circuito direcionador de pulsos*: envia um pulso de entrada (  $p$  ) para uma saída ou outra dependo de uma chave de controle (  $q$  ).

