

ARQUITETURA DE COMPUTADORES I

Estudo para a 1^a prova

SISTEMAS NUMÉRICOS POSICIONAIS

$$b^3 | b^2 | b^1 | b^0 | b^{-1} | b^{-2} | b^{-3}$$

'vírgula fracionária'

- Representação binar - magnitude

$$bs | b_{n-1} | b_{n-2} | \dots | b_1 | b_0$$

n bits de magnitude

bs: bit de sinal.
 ↗ 0: positivo
 ↗ 1: negativo

REPRESENTAÇÃO EM COMPLEMENTO DE 2

Se o número é...

↪ positivo (o bit de sinal é 0): já temos a representação direta em binário e podemos converter para decimal sem manipular o número.

↪ negativo (o bit de sinal é 1): sabemos que o sinal do número é negativo, mas ainda não podemos converter para decimal. Invertemos todos os bits e somamos 1 ao resultado para encontrar a verdadeira binária positiva deste número negativo.

Exemplo: número em complemento de 2 com 5 bits ($1\text{bs} + 4\text{mag}$) .

$$+5 = 00100_2$$

$$\begin{array}{r} -5 = \\ \hline 11000 \text{ compl. } 1 \\ + 1 \text{ soma } 1 \\ \hline 10111_2 \end{array}$$

Agora, vale registrar a operação, retornando ao número positivo

$$\begin{array}{r} 1011 \\ 0100 \\ + 1 \\ \hline 0101_2 \\ + 5 \end{array}$$

$$(A) \xrightarrow{\text{complemento de 2}} (-A) \xrightarrow{\text{complemento de 2}} (+A)$$

- Retirada do complemento de

Exemplo: número em 3 bits ($1\text{bs} + 2\text{mag}$)

bs	b_1	b_0	dec
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

→ 0 complemento de 2 não funciona com $+0$.

→ Sempre podemos representar um número negativo a mais que a quantidade de números positivos.

$$\begin{array}{r} 000 \\ 111 \\ + 1 \\ \hline 1000 \end{array} \rightarrow \text{complemento de 2 do zero}$$

$$\begin{array}{r} 100 \\ 101 \\ + 1 \\ \hline 0100 \end{array}$$

Precisaríamos de 4 bits para que o bit de sinal fosse representado. O espaço de representação não comporta o bs.

Overflow
(estoura o nº máx. de bits)

→ Por isso, o maior número que podemos representar com 8 bits é $+127$ e, o menor, -128 .

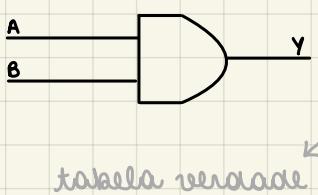
1 byte

0	0	0	0	0	0	0	0	$\rightarrow +0$
0	0	0	0	0	0	1	$\rightarrow +1$	
0	1	1	1	1	1	1	$\rightarrow +127$	
1	1	1	1	1	1	1	$\rightarrow -1$	
1	0	0	0	0	0	1	$\rightarrow -127$	
1	0	0	0	0	0	0	$\rightarrow -128$	

→ Quando o computador precisa fazer uma subtração, é mais fácil fazer uma soma poren aplicando complemento de 2 a um dos operandos para que seu sinal seja invertido.

FUNÇÕES LÓGICAS ELEMENTARES

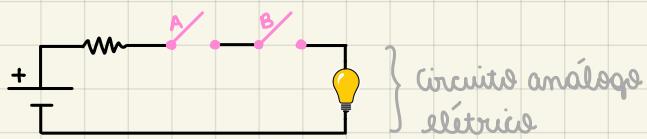
AND



tbla verdade

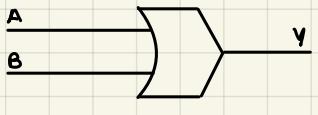
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$$Y = A \cdot B \rightarrow \text{Expressão booleana}$$



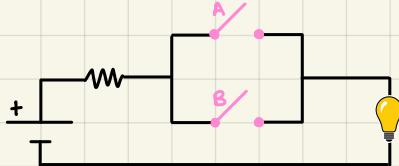
Circuito análogo elétrico

OR



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

$$Y = A + B$$



NOT



A	Y
0	1
1	0

$$Y = \bar{A}$$

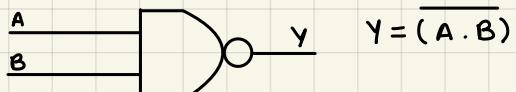
TEOREMAS DE DEMORGAN

$$(A+B) = \bar{\bar{A}} \cdot \bar{\bar{B}}$$

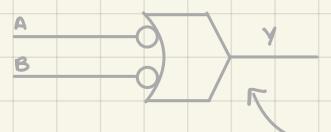
$$(\bar{A} \cdot \bar{B}) = \bar{\bar{A}} + \bar{\bar{B}}$$

O complemento de uma soma é igual ao produto dos complementos; o complemento de um produto é igual à soma dos complementos.

NAND

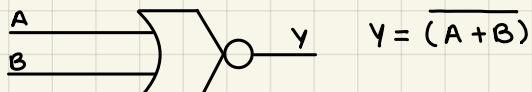


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

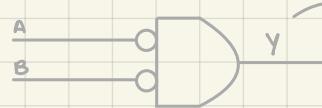


Pelo Teorema de De Morgan, sabemos que essa porta possui a mesma Tabela Verdade que a NAND.

NOR

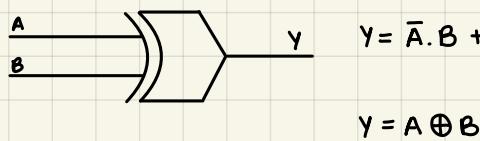


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



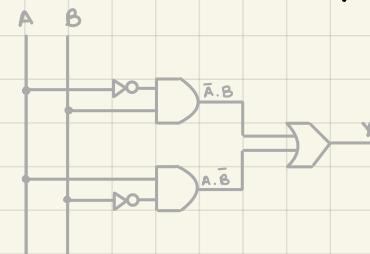
Pelo Teorema de De Morgan, possui a mesma TV que a NOR.

XOR

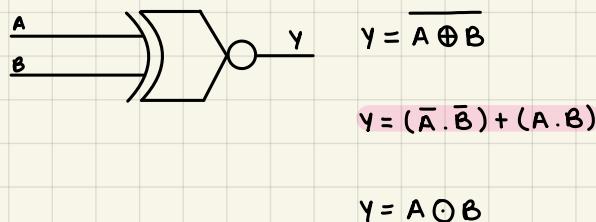


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

("ou exclusivo") → porta comparadora de diferença

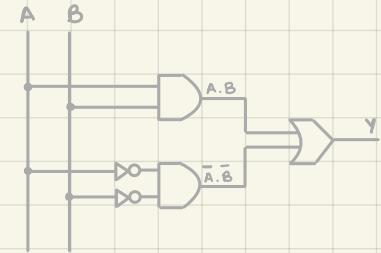


XNOR



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

→ porta comparadora de igualdade



Essa forma de escrever a equação é chamada de forma canônica ou soma de produtos (SOP). Ela nos dá 4 mintermos.
S' UMA delas, existe a 4 mintermos.

Sabemos mais sobre isso depois.

A L G E B R A B O O L E A N A

Propriedades básicas e teoremas

1 - Comutativa

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

3 - Associativa

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

$$(A + B) + C = A + (B + C)$$

5 - Negação

$$(\bar{A}) = \bar{\bar{A}}$$

$$\bar{0} = 1$$

$$\bar{1} = 0$$

2 - Distributiva

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

4 - Identidade

$$1 \cdot A = A$$

$$0 + A = A$$

6 - Complemento

$$A \cdot \bar{A} = 0$$

$$A + \bar{A} = 1$$

7 - Lei da inversão

$$(\bar{\bar{A}}) = \bar{A} = A$$

AND

$$0 \cdot 0 = 0$$

OR

$$0 + 0 = 0$$

$$\bar{0} = 1$$

8 - Lei da idempotência

$$A \cdot A = A$$

$$A + A = A$$

$$0 \cdot 1 = 0$$

$$1 \cdot 1 = 1$$

$$A \cdot 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 1$$

$$\bar{1} = 0$$

$$\bar{\bar{A}} = A$$

9 - Elementos nulos

$$0 \cdot A = 0$$

$$1 + A = 1$$

$$A \cdot 1 = A$$

$$A \cdot A = A$$

$$A \cdot \bar{A} = 0$$

$$A + 1 = 1$$

$$A + A = A$$

$$A + \bar{A} = 1$$

Teorema de soma ou maioria variável

$$A \cdot (A + B) = A$$

$$A \cdot (\bar{A} + B) = A \cdot B$$

$$A + (A \cdot B) = A$$

$$A + (\bar{A} \cdot B) = A + B$$

FORMA CANÔNICA - EQUAÇÃO DA SOMA DE MINTERMOS

Embora existam muitas representações possíveis usando equações e circuitos para a mesma função booleana, as tabelas-verdade constituem uma representação padrão de uma função. Isso significa que, para qualquer função, pode haver muitas equações e circuitos possíveis, mas há apenas uma tabela-verdade. A representação por T.V. é única.

Exemplo: as equações $Y = AB + \bar{A}$ e $Y = \bar{A}\bar{B} + \bar{A}B + AB$ representam a mesma função booleana, pois possuem a mesma T.V.

A	B	Y	A	B	Y	A	B	Y	A	B	Y
0	0	1	0	1	1	1	0	0	1	1	1
0	1	0	1	0	1	1	1	1	1	1	1

A comparação de tabelas-função bem quanto uma função tem poucas entradas, mas se tivessemos função com 36 entradas e, portanto, aprox. 4 milhares de saídas?!

Poderemos abordar esse problema levando em consideração que, em muitas vezes, a quantidade de 1s de saída em uma T.V. pode ser muito pequena em comparação ao número de 0s de saída. E se pudéssemos obter uma equação que já nos mostra todos (e apenas) os casos nos quais a saída da tabela-verdade será 1? Haveria uma representação mais compacta, mas ainda assim, de uma função booleana?

SIM! * É possível se criar uma representação padrão que descreve apenas as situações em que a saída da função é 1, assumindo-se que nos demais casos ela é 0!!!!

A forma canônica para funções booleanas que estudaremos é conhecida como soma de mintermos. Ela só ocorre se a equação estiver na forma de uma soma de produtos (SOP) e se cada produto for um mintermo.

MINTERMO é um termo de produto que contém todas as variáveis de entrada da função exatamente uma vez, seja na forma afirmada ou negada (complementada).

A conversão de qualquer equação para a forma canônica de mintermos pode ser realizada seguindo simplesmente alguma rotina:

① Primeiro, manipulemos a equação até que ela esteja na forma de uma soma de produtos.

Suponha que nos seja dada a equação $F(A, B, C) = (A+B)(\bar{A}+AC)B$. Nós a manipulamos como segue:

$$F = (A+B)(\bar{A}+AC)B$$

$F = (A+B)(\bar{A}B + ACB)$ propriedade distributiva

$$F = A(\bar{A}B + ACB) + B(\bar{A}B + ACB)$$
 prop. distributiva

$$F = A\bar{A}B + AACB + B\bar{A}B + BACB$$
 prop. distributiva

$$F = 0 \cdot B + ACB + \bar{A}B + ACB$$
 complemento, comutativa, idempotência

zero ou alguma coisa \rightarrow o zero não importa, o que define o resultado é a outra coisa

$$F = \cancel{ACB} + \cancel{\bar{A}B} + \cancel{ACB}$$
 elementos nulos, identidade

$$F = ACB + \bar{A}B$$
 idempotência

② Segundo, exponda os termos até se tornarem mintermos.

$$F = ACB + \bar{A}B$$

$$F = ACB + \bar{A}B \cdot 1 \quad \text{identidade}$$

mesma coisa que $\bar{A}B$

$$F = ACB + \bar{A}B (\underbrace{C + \bar{C}}_1) \quad \text{complemento}$$

$$F = ACB + \bar{A}BC + \bar{A}\bar{B}C \quad \text{distributiva}$$

③ Por uma questão de clareza, disponha as literais dentro de cada termo em ordem alfabética (ou na ordem em que aparecem em uma T.V. pré-determinada).

$$F = \bar{A}B\bar{C} + \bar{A}BC + ABC$$

- Para cada linha da tabela verdade, se a saída for 1, os valores de entrada formarão um minterme. A representação binária das literais de um minterme é feita de acordo com a regra: se a variável estiver negada, ela assume o valor 0, caso contrário, ela assume o valor 1.

$$F = \underbrace{\bar{A} \ B \ \bar{C}}_{0 \ 1 \ 0} + \underbrace{\bar{A} \ B \ C}_{0 \ 1 \ 1} + \underbrace{A \ B \ C}_{1 \ 1 \ 1}$$

Já que convertemos os mintermos para binário, podemos também convertê-los para decimal, obtendo uma representação ainda mais compacta.

$$Y = F_{SOP}(A, B, C) = \{2, 3, 7\}$$

Por fim, podemos montar a tabela-verdade:

decimal	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1 → $\bar{A}BC$
3	0	1	1	1 → $\bar{A}BC$
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1 → ABC

$$F = \sum_m(2, 3, 7)$$

MAPAS DE KARNAUGH

- método gráfico usado para simplificar uma equação booleana ou converter uma tabela verdade no seu circuito lógico correspondente.
- O M.K. é uma forma ordenada utilizada para minimizar uma expressão lógica, que geralmente produz um circuito com configuração mínima.

General instructions:

- 1 - Select the K-map according to the number of variables
- 2 - Identify minterms or maxterms as given in the problem.
- 3 - For SOP put 1's in positions of K-map respective to the minterms (0's elsewhere).
- 4 - For POS put 0's in positions of K-map respective to the maxterms (1's elsewhere).
- 5 - make rectangular groups containing total terms in power of two.

Sobre a formação de grupos:

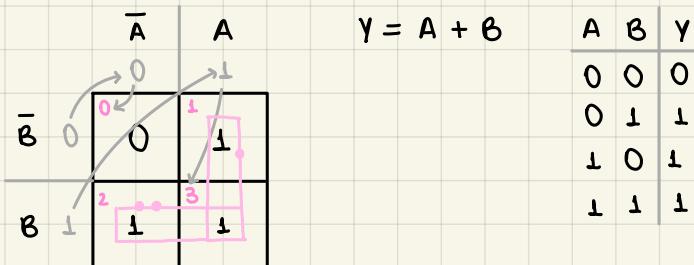
- Para SOP, agruparemos os 1's em bloco de 1's adjacentes.
- Formar os maiores grupos retangulares possíveis que tenham uma quantidade de termos (1's) em potência de 2 (1, 2, 4, 8, 16...).
- Formar o menor número possível de blocos, podendo haver intervalos entre grupos para a formação de grupos de maior tamanho.

Como escrever a expressão final simplificada:

- Será a "soma" (+) das expressões de cada bloco (que, por sua vez, serão produtos).
- Cada grupo formará uma expressão. Para isso, eliminam-se as variáveis que mudam de estado dentro do bloco ($\bar{x} \rightarrow x$ ou $x \rightarrow \bar{x}$). As literais que não mudam de estado são exibida / montada na expressão, sempre com um AND (.) entre elas.

M.K. para duas variáveis

Exemplo: $Y = f_{SOP}(A, B) = \{1, 2, 3\}$



• M.K. para três variáveis

Exemplo: $Y = f_{\text{SOP}}(A, B, C) = \{1, 2, 5, 6, 7\}$

\bar{A}	\bar{B}	\bar{C}	C
0	0	0	0
0	0	1	1
0	1	0	0
1	0	1	1
1	0	1	0
1	1	0	1
1	1	1	1

$$y = \bar{B}C + B\bar{C} + AB$$

O mapa "se desloca" tanto verticalmente quanto horizontalmente, e é fechado tridimensionalmente, contínuo em sua borda.

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

• M.K. para quatro variáveis

Exemplo: $Y = f_{\text{SOP}}(A, B, C, D) = \{1, 3, 4, 10, 11, 14, 15\}$

\bar{A}	\bar{B}	\bar{C}	\bar{D}	C	D	\bar{C}	\bar{D}
0	0	0	1	1	1	1	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0
1	0	1	1	0	1	0	1
1	1	0	0	1	1	1	1
1	1	0	1	0	1	1	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

3 grupos = 3 termos de produto resumidos

$$\bar{ABC}\bar{D} + \bar{AB}\bar{D} + AC$$

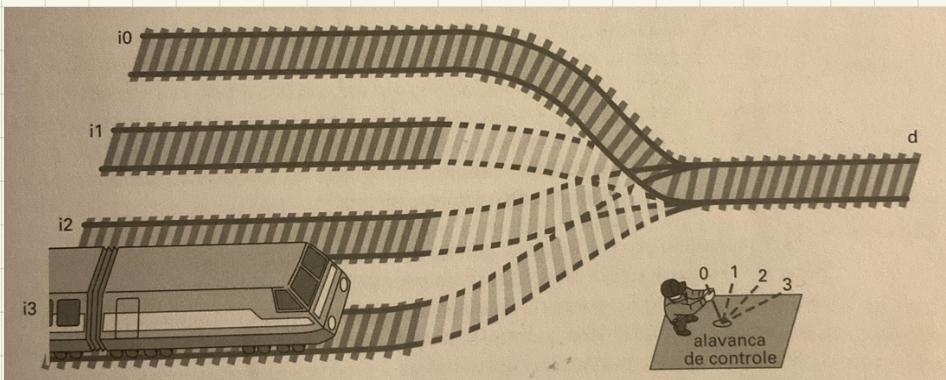
YAY! acertei essa sozinha !!

MUXES

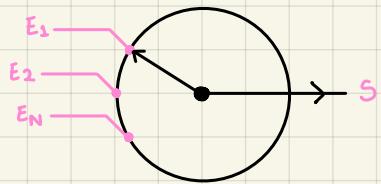
Um multiplexador (MUX) é um bloco construtivo tipicamente usado em circuitos digitais. Um MUX $N \times 1$ tem N entradas de dados e 1 saída, permitindo que apenas 1 das entradas seja passada para a saída. Algumas vezes, os MUX são chamados de seletores porque selecionam uma das entradas para ser passada à saída.

O MUX é como um aparelho de mudança de via em uma ferrovia, que põe em conexão diversas vias de entrada com uma única via de saída. A chave de controle estabelece a conexão entre a via adequada de entrada e a via de saída. O surgimento de um trem na saída dependerá se há um trem presente na via de entrada selecionada no momento.

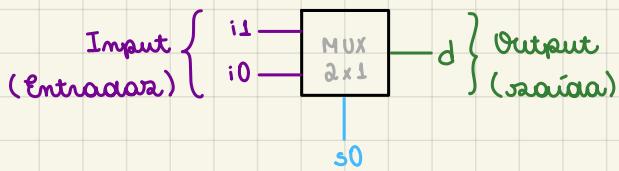
Em um MUX, o controle não é a chave, mas sim entradas de seleção, as quais representam a conexão desejada em binário. Se inverza de um trem aparecer ou não na saída, o MUX produz um output de 1 ou 0, dependendo de se o input selecionado tem um 1 ou um 0.



N : 1

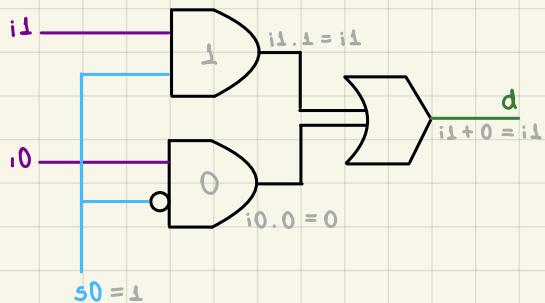


• MUX 2 x 1



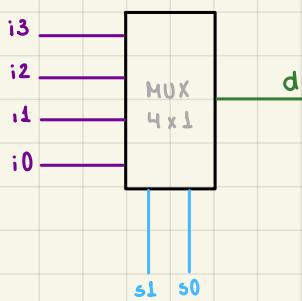
- Tem duas entradas de dados i_1 e i_0 , uma entrada de seleção s_0 e uma saída de dados d .

- Se $s_0 = 0$, o valor de i_0 passará para a saída. Se $s_0 = 1$, o valor que passará será o de i_1 .



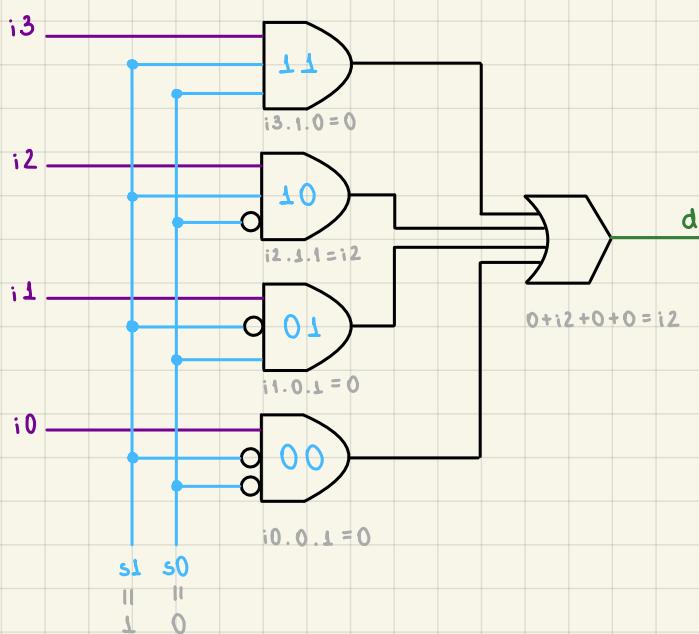
- Um multiplexador sempre tem apenas uma saída de dados, não importando quantas entradas.

• MUX 4 x 1

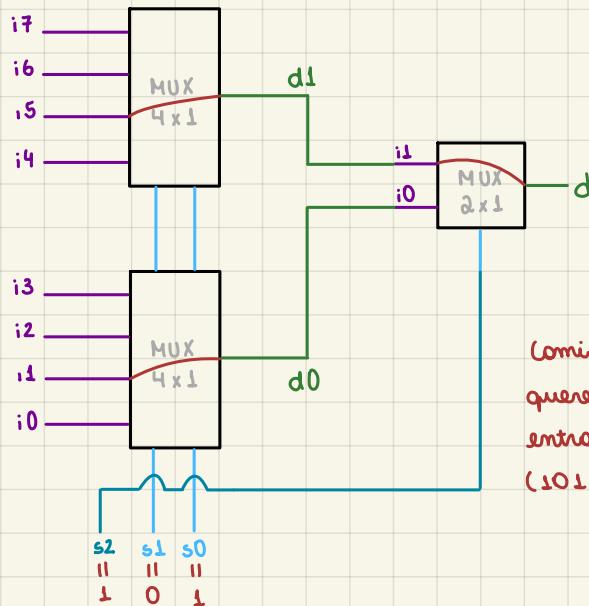
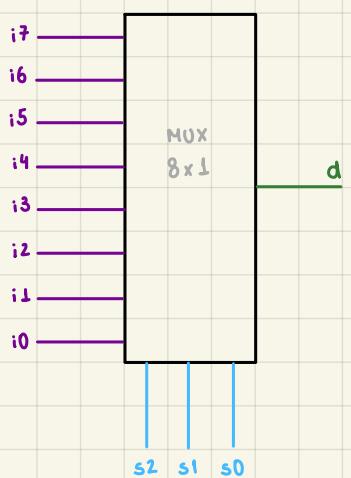


nos seletores, colocamos a representação binária da posição da entrada desejada.

Exemplo: s0 deve selecionar a entrada i2 para passar

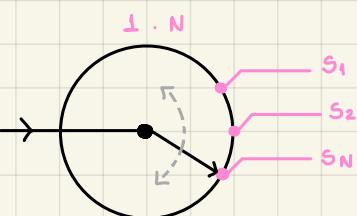


- MUX 8x1 (expansão)

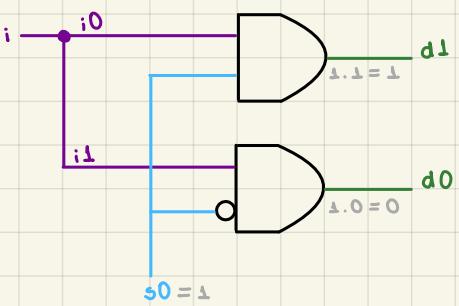
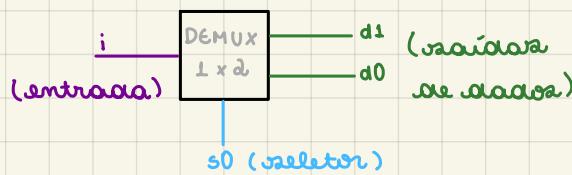


DEMULTIPLEXADORES

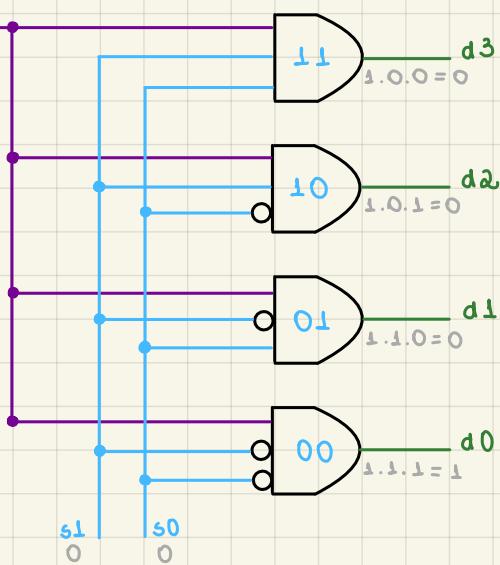
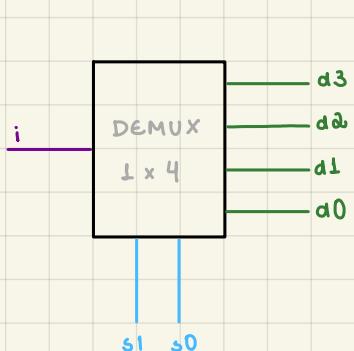
Têm o comportamento oposto ao de um MUX. Um DEMUX $1 \times N$ tem uma entrada de dados e, com base nos valores de $\log_2(N)$ linhas de seleção, para cada entrada para uma das N saídas. As outras saídas permanecem em 0.



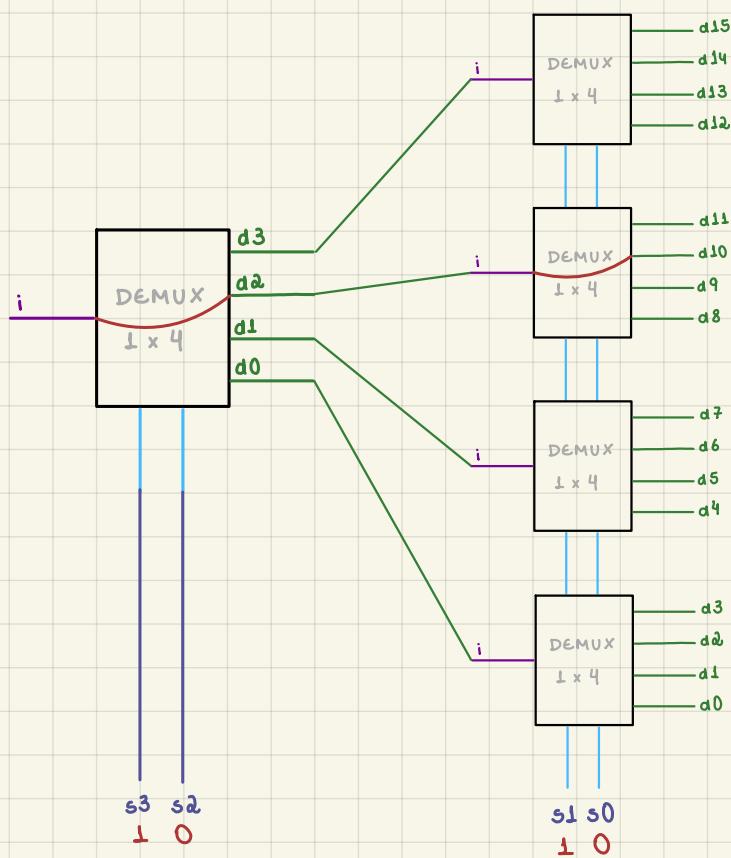
- DEMUX 1×2



- DEMUX 1×4



- Implemente um DEMUX 1×16 utilizando 4 saídas DEMUX 1×4

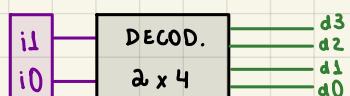


DECODIFICADORES

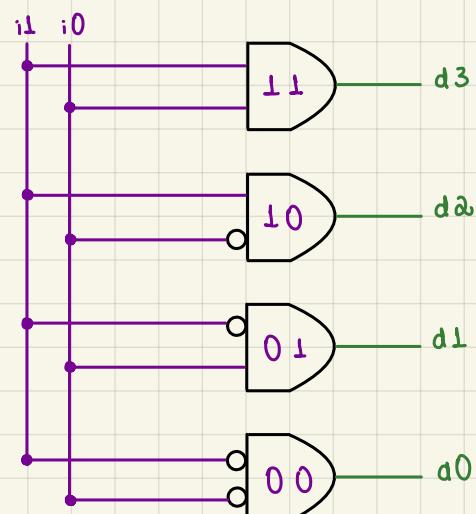
Decodifica um número binário de n bits de entrada selecionando exatamente uma das 2^n saídas de decodificador em 1. Por exemplo, um decodificador de 2 entradas tem $2^2 = 4$ saídas, d_3, d_2, d_1, d_0 . Se as duas entradas i_1, i_0 forem 00, então d_0 será 1 e as demais saídas serão 0. Se $i_1, i_0 = 01$, d_1 será 1. Se $i_1, i_0 = 10$, d_2 será 1. Se $i_1, i_0 = 11$, d_3 será 1.

$$\begin{aligned}d_3 &= i_1 \cdot i_0 \rightarrow 1 \cdot 1 \\d_2 &= i_1 \cdot \bar{i}_0 \rightarrow 1 \cdot 0 \\d_1 &= \bar{i}_1 \cdot i_0 \rightarrow 0 \cdot 1 \\d_0 &= \bar{i}_1 \cdot \bar{i}_0 \rightarrow 0 \cdot 0\end{aligned}$$

$i_1 \cdot i_0$	d_3	d_2	d_1	d_0
0 0	0	0	0	1
0 1	0	0	1	0
1 0	0	1	0	0
1 1	1	0	0	0



O decodificador permite que, ao analisar a saída, possamos descobrir qual foi a entrada



- Um decodificador frequentemente tem uma entrada extra chamada de enable. Quando o enable é 1, o decodificador atua normalmente, mas, quando é 0, todas as saídas serão 0 e nenhuma será 1.

CODIFICADORES (fazem o caminho oposto de decodificadores)

O codificador coloca um valor binário na saída (n) dependendo qual das n entradas é 1. Por exemplo, um codificador 4×2 tem 4 entradas i_3, i_2, i_1, i_0 e duas saídas d_1 e d_0 . Quando a entrada é 0001, a saída é 00. Para 0010, a saída é 01. Para 0100, a saída é 10 e para 1000 a saída é 11. Em outras palavras, se convertermos as saídas binárias do codificador para decimal, saberemos em qual posição (em qual entrada) o 1 foi aplicado.

$i_3 = 1 \rightarrow$ resulta na saída de 3 em binário 11

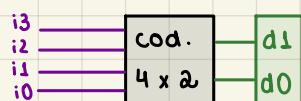
$i_2 = 1 \rightarrow$ resulta na saída de 2 em binário 10

$i_1 = 1 \rightarrow$ resulta na saída de 1 em binário 01

$i_0 = 1 \rightarrow$ resulta na saída de 0 em binário 00

i_3	i_2	i_1	i_0	d_1	d_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

→ outra opção
de entrada
não existem



- Restrições: sempre existirá uma única entrada ativa por vez.
- Projeto intuitivo: raciocínio direto $d_1 = i_3 + i_2, d_0 = i_3 + i_1$

- Projeto metodológico via mapa de Karnaugh

→ mapa para d_1

		$i_1 \cdot i_0$	$i_1 \cdot \bar{i}_0$	$\bar{i}_1 \cdot i_0$	$\bar{i}_1 \cdot \bar{i}_0$
		0 0	0 1	1 0	1 1
$i_3 \cdot i_2$	0 0	X	0	X	0
0 1	1	X	X	X	X
1 1	X	X	X	X	X
1 0	1	X	X	X	X

$x \rightarrow$ don't care if it's 0 or 1

$$d_1 = i_2 + i_3$$

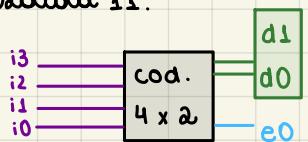
→ mapa para d_2

		$i_1 \cdot i_0$	$i_1 \cdot \bar{i}_0$	$\bar{i}_1 \cdot i_0$	$\bar{i}_1 \cdot \bar{i}_0$
		0 0	0 1	1 0	1 1
$i_3 \cdot i_2$	0 0	X	0	X	1
0 1	0	X	X	X	X
1 1	X	X	X	X	X
1 0	1	X	X	X	X

$$d_2 = i_1 + i_3$$

• Codificador com prioridade

dada com situação em que mais de uma entrada são 1 ao mesmo tempo. A prioridade é dada à entrada mais elevada que tem um 1 e fornece o valor binário dessa entrada. Por exemplo, se um codificador de prioridade 4x2 tiver as entradas i_3 e i_2 iguais a 1 (de modo que as entradas são 1010), a prioridade será dada a i_3 e, portanto, será produzida a saída 11.



$$e0 = i_0 + i_1 + i_2 + i_3$$

i_3	i_2	i_1	i_0	d_1	d_0	e_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

FULL-ADDER

$$\begin{array}{l} C_i = \text{carry in} \\ C_o = \text{carry out} \end{array} \quad \begin{array}{c} C_i \\ + A \\ B \\ \hline C_o \quad S \end{array}$$

A half adder is the simplest form of an adder. It takes two single-bit inputs, A and B, and produces two outputs: the sum (S) and the carry (C). The sum output represents the least significant bit of the addition, while the carry output indicates whether there is a carry-over to the next bit.

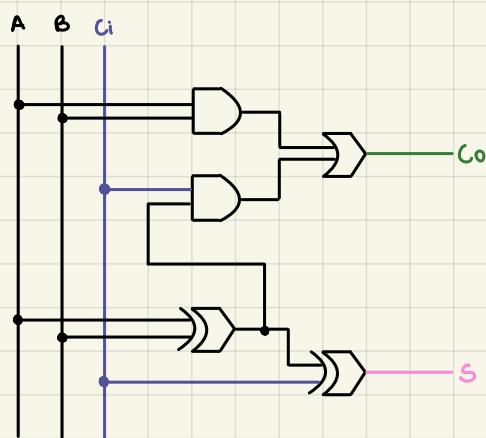
A full adder is an extension of a half-adder. It takes 3 inputs: A, B and a carry-in (C_i), and produces 2 outputs: the sum and the carry-out (C_o). The carry-in represents the carry-over from the previous bit, allowing full-adders to perform multi-bit additions.

Co é o sítimo da soma (acontece quando $S=1$). Vira o carry-in do próximo estágio.

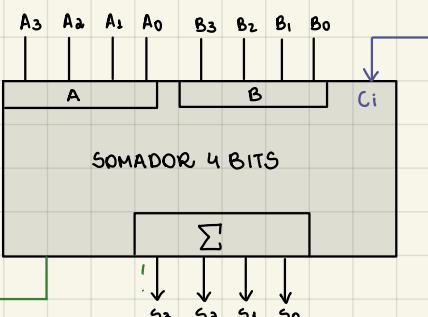
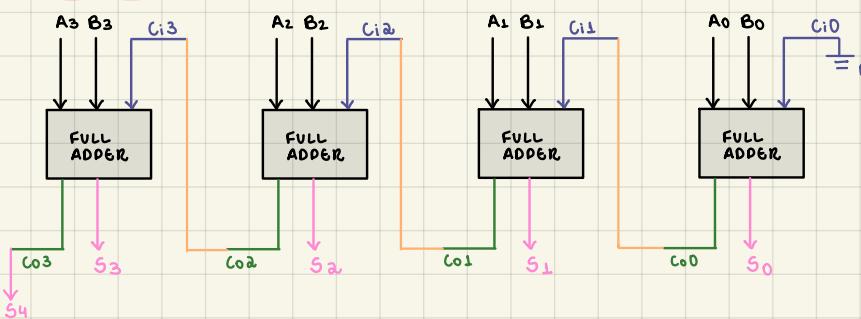
Bloco lógico:

TERMO	C_i	A	B	S	entradas		saídas
					Co	S	
0	0	0	0	0	0	0	• mintermox de S
1	0	0	1	0	1	0	
2	0	1	0	0	1	0	$S = f_{SOP}(C_i, A, B) = \{1, 2, 4, 7\}$
3	0	1	1	1	0	1	$S = C_i \oplus (A \oplus B)$
4	1	0	0	0	1	0	
5	1	0	1	1	0	1	
6	1	1	0	1	0	1	$C_o = f_{SOP}(C_i, A, B) = \{3, 5, 6, 7\}$
7	1	1	1	1	1	1	$C_o = A \cdot B + C_i \cdot y$
							$y = C_i(\bar{A}B + A\bar{B})$

Forma de implementar um somador completo que utiliza menor porta:



SOMADOR DE 4 BITS

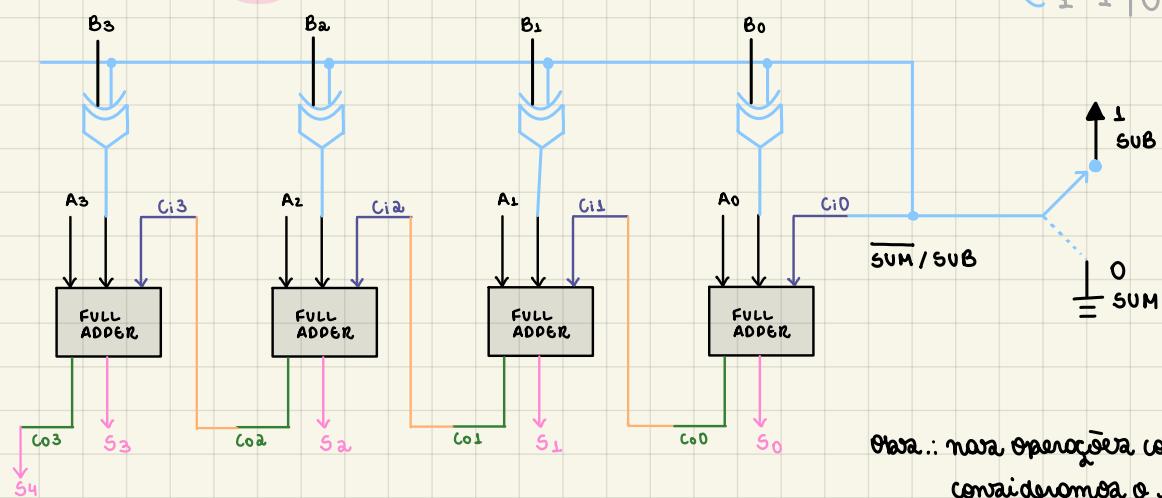


$$\begin{array}{r} \text{Co3} \text{ Ci3} \text{ Ci2} \text{ Ci1} \text{ 0} \\ + \quad \quad \quad \quad \quad \\ \text{A3} \text{ A2} \text{ A1} \text{ A0} \quad \text{B3} \text{ B2} \text{ B1} \text{ B0} \\ \hline \text{S4} \text{ S3} \text{ S2} \text{ S1} \text{ S0} \end{array}$$

T.V. da XOR		
Ci ₀	A	Y
0	0	0
0	a	a
1	a	̄a
1	1	0

XOR é uma inversão controlada. Ci₀ é o controle de inversão.

SOMADOR E SUBTRATOR DE 4 BITS



Obs.: para operações com sinal não considerar o resultado.

O overflow só acontece quando temos POS+POS ou NEG+NEG.

Para realizar a subtração, fazemos complemento de 2 na entrada. Para isso, o carry-in inicial é setado como 1 (adicionamos +1 à soma) e cada bit da segunda entrada é invertido por uma porta XOR.