



# Interativa

## Circuitos Lógicos Digitais

**Autor:** Prof. Roberto Eduardo Bruzetti Leminski

**Colaboradoras:** Profa. Vanessa Santos Lessa

Profa. Larissa Rodrigues Damiani

## Professor conteudista: Roberto Eduardo Bruzetti Leminski

Possui mestrado em Engenharia Elétrica (ênfase em Microeletrônica) pela Escola Politécnica da Universidade de São Paulo (2001) e é formado em Tecnologia em Materiais, Processos e Componentes Eletrônicos (MPCE) pela Faculdade de Tecnologia de São Paulo (1998). É docente dos cursos de Ciência da Computação e Sistemas de Informação da Universidade Paulista desde 2007, com passagem por disciplinas dos cursos de Ciências Contábeis e Administração. Foi líder da disciplina de Circuitos Digitais do curso de Ciência da Computação.

### Dados Internacionais de Catalogação na Publicação (CIP)

L554c Leminski, Roberto Eduardo Bruzetti.

Circuitos Lógicos Digitais / Roberto Eduardo Bruzetti Leminski. –  
São Paulo: Editora Sol, 2021.

164 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e  
Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Conversão. 2. Código. 3. Circuito. I. Título.

CDU 621.382

U512.27 – 21

Prof. Dr. João Carlos Di Genio  
**Reitor**

Prof. Fábio Romeu de Carvalho  
**Vice-Reitor de Planejamento, Administração e Finanças**

Profa. Melânia Dalla Torre  
**Vice-Reitora de Unidades Universitárias**

Profa. Dra. Marília Ancona-Lopez  
**Vice-Reitora de Pós-Graduação e Pesquisa**

Profa. Dra. Marília Ancona-Lopez  
**Vice-Reitora de Graduação**

### **Unip Interativa – EaD**

Profa. Elisabete Brihy  
Prof. Marcello Vannini  
Prof. Dr. Luiz Felipe Scabar  
Prof. Ivan Daliberto Frugoli

### **Material Didático – EaD**

Comissão editorial:

Dra. Angélica L. Carlini (UNIP)  
Dr. Ivan Dias da Motta (CESUMAR)  
Dra. Kátia Mosorov Alonso (UFMT)

Apoio:

Profa. Cláudia Regina Baptista – EaD  
Profa. Deise Alcantara Carreiro – Comissão de Qualificação e Avaliação de Cursos

Projeto gráfico:

Prof. Alexandre Ponzetto

Revisão:

Kleber Souza  
Bruna Baldez



# Sumário

## Circuitos Lógicos Digitais

APRESENTAÇÃO .....	7
INTRODUÇÃO .....	7

### Unidade I

1 SISTEMAS DE NUMERAÇÃO E CONVERSÕES .....	9
1.1 O sistema decimal .....	9
1.2 Conversão do sistema decimal para outros sistemas .....	10
1.3 Conversão de outros sistemas para o sistema decimal .....	13
1.4 Sistema binário para octal e hexadecimal .....	14
1.5 Bits e bytes .....	17
2 CONVERSÃO DE NÚMEROS REAIS E OPERAÇÕES COM NÚMEROS BINÁRIOS .....	18
2.1 Conversão de números reais .....	18
2.2 Operações aritméticas com números binários .....	21
2.3 Números binários na forma de complemento de 2 (Comp-2) .....	25
3 INTRODUÇÃO À LÓGICA PROPOSICIONAL E TEORIA DAS PORTAS LÓGICAS .....	31
3.1 A lógica matemática .....	31
3.2 Operadores lógicos .....	33
3.3 Expressões lógicas .....	38
3.4 Portas lógicas .....	42
3.5 Associação de portas lógicas .....	44
4 CONSTRUÇÃO E SIMPLIFICAÇÃO DE CIRCUITOS LÓGICOS .....	46
4.1 Leis da lógica .....	47
4.2 Obtenção e simplificação de circuitos lógicos .....	54
4.3 Interruptores como portas lógicas .....	58

### Unidade II

5 MAPAS DE KARNAUGH .....	71
5.1 Mapas de Karnaugh para três entradas .....	77
5.2 Saídas indiferentes no mapa de Karnaugh .....	87
6 PROJETO DE CIRCUITOS LÓGICOS .....	92
6.1 Mapas de Karnaugh para quatro entradas .....	92
6.2 Simuladores para circuitos lógicos digitais .....	101
6.3 Circuitos combinacionais e sequenciais .....	103
6.4 Display de sete segmentos (SSD) .....	105

### **Unidade III**

7 CIRCUITOS CODIFICADORES E DECODIFICADORES .....	119
7.1 Codificadores e decodificadores.....	119
7.2 Código BCD 8421 .....	120
7.3 Código BCH.....	124
7.4 Código Excesso 3 .....	128
7.5 Código de Gray.....	133
7.6 Decodificador BCD 8420 para SSD .....	137
8 CIRCUITOS ARITMÉTICOS .....	140
8.1 Circuitos meio somadores.....	141
8.2 Circuitos somadores completos .....	145
8.3 Circuitos meio subtratores .....	148
8.4 Circuitos subtratores completos .....	149

## APRESENTAÇÃO

A disciplina *Circuitos Lógicos Digitais* tem como objetivo fornecer ao aluno o conhecimento dos fundamentos dos circuitos básicos de um computador e sua ligação com a lógica matemática e a lógica de programação. Além disso, visa tornar possível o conhecimento da interface entre as linguagens de programação (*software*) e os circuitos de um computador (*hardware*), permitindo assim a melhor compreensão do funcionamento dos principais comandos de uma linguagem de programação. O uso da matemática, tanto da área da lógica quanto da aritmética e das bases numéricas (principalmente a base binária), será apresentado e terá seus usos computacionais explicados.

Serão também exibidas algumas ferramentas para o projeto e a simulação de circuitos lógicos em específico e de circuitos elétricos em geral, sendo fornecida uma pequena lista.

Por fim, em diversas partes do presente livro-texto será feita a ligação dos conceitos apresentados com técnicas de programação estruturada, especificamente com a linguagem Python. Trata-se da ligação de um complemento aos conteúdos ora exibidos, não sendo necessário nenhum domínio de técnicas de programação.

## INTRODUÇÃO

A palavra **computação** deriva do verbo computar, cujos significados incluem: contar, avaliar ou calcular. Embora atualmente seja automático associar computação aos dispositivos eletrônicos como computadores, celulares e *tablets*, na verdade estamos lidando com uma ciência que é filha da matemática.

Podemos reportar o início da computação atual ao século XIX com o conceito de algoritmo desenvolvido por Ada Lovelace (1815-1852), a máquina diferencial de Charles Babbage (1791-1871), obviamente, a lógica matemática de George Boole (1815-1864) e Augustus De Morgan (1806-1871).

Apresentaremos, aqui, os fundamentos da matemática e da lógica que levaram ao desenvolvimento da computação eletrônica, bem como os conceitos de circuitos lógicos, que são a base de todos os dispositivos computacionais.

Este material está dividido em oito tópicos, que são apresentados resumidamente a seguir:

- **Tópico 1:** serão apresentados e discutidos os sistemas e bases numéricas decimal, binário, octal e hexadecimal; também serão exibidas as metodologias de conversão das diferentes bases numéricas para números naturais.
- **Tópico 2:** serão demonstradas as metodologias da conversão de bases numéricas para números reais (números negativos e não inteiros), além das operações aritméticas de soma e subtração para números binários.

- **Tópico 3:** serão introduzidos os conceitos fundamentais da lógica proposicional, também chamada de álgebra de Boole; além das portas lógicas, que são o fundamento da disciplina *Circuitos Lógicos Digitais*.
- **Tópico 4:** será exibida a construção de circuitos utilizando portas lógicas, a partir de uma expressão lógica. Também serão apresentados os conceitos de equivalência e simplificação de circuitos por meio exclusivamente das propriedades da lógica matemática.
- **Tópico 5:** serão apresentados os mapas de Karnaugh, que são ferramentas tanto para simplificar circuitos lógicos já existentes quanto para obter a expressão que resulta em uma saída desejada; na prática, consiste em projetar um circuito. Aqui serão trabalhados circuitos com duas e três entradas.
- **Tópico 6:** será mostrada a resolução dos mapas de Karnaugh para circuitos com quatro entradas, além da combinação dos resultados dos mapas com as regras de simplificação da lógica matemática apresentada no tópico 4. Também será explicado o conceito de circuito sequencial, iniciando com o *display* de sete segmentos (SSD), e algumas opções de simuladores para portas lógicas.
- **Tópico 7:** serão demonstrados os conceitos de circuitos codificadores e decodificadores, além de circuitos envolvendo *displays* de LEDs de sete segmentos. Aqui serão utilizados quase todos os conceitos apresentados nos tópicos anteriores.
- **Tópico 8:** por fim, serão apresentados circuitos aritméticos que realizam a soma e a subtração de números binários, conforme o tópico 2.

Com este conjunto de tópicos, serão abordados todos os fundamentos e principais arranjos de circuitos lógicos digitais, bem como os principais circuitos sequenciais utilizados em eletrônica digital.

Bons estudos!



# Unidade I

## 1 SISTEMAS DE NUMERAÇÃO E CONVERSÕES

### 1.1 O sistema decimal

O sistema de numeração que utilizamos é o sistema decimal, por meio dos algarismos arábicos. Mesmo idiomas que possuem outros alfabetos representam valores numéricos. Esse sistema de numeração, com 10 algarismos, é tão amplamente difundido que é fácil pensarmos que se trata de algo natural, e não de uma convenção.

Entender como funciona o sistema decimal, que é o sistema numérico que usamos no nosso cotidiano, nos permitirá compreender os outros sistemas numéricos. Para isto, pegaremos um número real ao acaso, por exemplo, 7234,56. Agora, faremos a decomposição dele:

$$\begin{aligned}7234,56 &= \\&= 7.000 + 200 + 30 + 4 + 0,5 + 0,06 = \\&= (7 \cdot 1.000) + (2 \cdot 100) + (3 \cdot 10) + (4 \cdot 1) + (5 \cdot 0,1) + (6 \cdot 0,01) = \\&= (7 \cdot 10^3) + (2 \cdot 10^2) + (3 \cdot 10^1) + (4 \cdot 10^0) + (5 \cdot 10^{-1}) + (6 \cdot 10^{-2})\end{aligned}$$

O sistema decimal na verdade é uma forma de representar valores como uma somatória de potências de 10, que é a **base do sistema**. A primeira casa à esquerda da vírgula representa  $10^0$ , lembrando que qualquer número elevado a zero é igual a 1, e o expoente das potências acresce em 1 para cada dígito à esquerda e decresce em 1 para cada dígito à direita.

Ao longo da história, outros sistemas de numeração foram utilizados: por exemplo, babilônicos utilizavam um sistema de base 60; galeses e irlandeses usavam um sistema de base 20.

Além do sistema decimal, na ciência da computação utilizamos outros sistemas numéricos, com diferentes bases: binário (base 2), octal (base 8) e hexadecimal (base 16). Todos os sistemas funcionam da mesma forma, sendo os valores representados como uma somatória de potências da base e utilizando-se de uma quantidade de algarismos igual à base do sistema (2 algarismos no sistema binário, 8 no sistema octal, 16 no sistema hexadecimal). Assim, temos os seguintes algarismos para os sistemas de numeração (o uso de letras para representar valores numéricos no sistema hexadecimal será explicado adiante):

- Sistema decimal (10 algarismos): 0, 1, 2, 3, 4, 5, 6, 7, 8, e 9.
- Sistema binário (2 algarismos): 0 e 1.
- Sistema octal (8 algarismos): 0, 1, 2, 3, 4, 5, 6 e 7.
- Sistema hexadecimal (16 algarismos): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F.

O uso do sistema binário na ciência da computação se deve ao fato de ele possuir apenas dois algarismos (0 e 1); assim, um dígito de um valor numérico representado neste sistema pode significar dois estados elétrica e eletronicamente: com carga elétrica (1) ou sem carga elétrica (0), ligado (1) ou desligado (0), magnetizado (1) ou desmagnetizado (0). Nos tópicos 3 e 4 ficarão mais claras as vantagens de podermos representar um valor por meio de apenas dois estados em vez dos dez estados/algarismos do sistema decimal.

O uso dos sistemas octal e hexadecimal se deve ao fato de a base de ambos ser uma potência de 2 ( $8 = 2^3$  e  $16 = 2^4$ ), o que permite uma fácil conversão do sistema binário para esses. Porém, o mais importante é que bases maiores permitem que um mesmo valor numérico seja representado com uma quantidade menor de dígitos, como será visto a seguir.



### Observação

Para indicar em que base um número está representado, colocamos entre parênteses com a base na sequência, em subscrito:

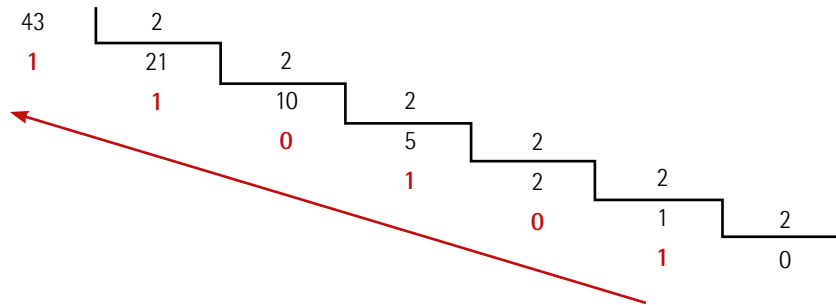
$(1234,67)_{10}$  significa que é um valor na base 10, por exemplo.

É importante observar que a lógica e o funcionamento de todos os sistemas numéricos são equivalentes, mudando apenas a base e a quantidade de algarismos. Assim, por exemplo, a conversão do sistema decimal para o sistema binário seguirá o mesmo raciocínio da conversão do sistema decimal para o octal: no primeiro caso realiza-se uma sucessão de divisões por 2, e no segundo uma sucessão de divisões por 8.

## 1.2 Conversão do sistema decimal para outros sistemas

Para converter um número inteiro do sistema decimal para outro sistema, divide-se o número sucessivamente pela base do sistema desejado, anotando-se os restos das divisões, até não ser mais possível realizar a divisão, ou seja, quando o quociente for igual a zero. A sequência composta dos restos das sucessivas divisões, **na ordem inversa em que foram obtidos**, corresponde ao número no novo sistema de numeração.

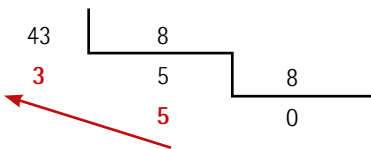
Exemplo: o número  $(43)_{10}$  convertido para o sistema binário ficaria:



Assim,  $(43)_{10} = (101011)_2$

De forma semelhante, ao convertermos o mesmo valor para o sistema octal, realizaremos o processo idêntico, porém dividindo por 8.

Exemplo: convertendo o número  $(43)_{10}$  agora para o sistema octal ficaria:



Assim,  $(43)_{10} = (53)_8$

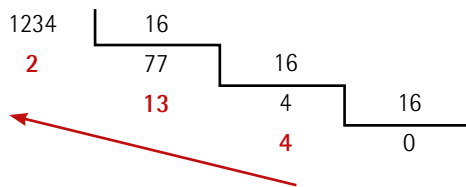
Para convertermos para o sistema hexadecimal (base 16), precisamos de 16 algarismos. Com o objetivo de não se criarem novos algarismos, utilizam-se as letras de A a F, aproveitando-se a existência da ordem alfabética. A tabela a seguir apresenta os valores correspondentes, no sistema decimal, às letras utilizadas como algarismos no sistema hexadecimal:

**Tabela 1 – Letras utilizadas como algarismos no sistema hexadecimal**

Sistema hexadecimal	Sistema decimal
A	10
B	11
C	12
D	13
E	14
F	15

Portanto, a conversão do sistema decimal para o sistema hexadecimal é realizada da mesma forma, porém substituem-se os restos da divisão maiores que 9 pela letra/algarismo hexadecimal correspondente.

Exemplo: convertendo  $(1234)_{10}$  para o sistema hexadecimal:



Assim,  $(1234)_{10} = (4D2)_{16}$



## Observação

Muitas linguagens de programação e afins, como o HTML5, utilizam-se de três números hexadecimais de dois dígitos para definir a intensidade de cores compostas, sendo vermelho, verde e azul, nesta ordem. Por exemplo, a cor #F32A3C significa vermelho com intensidade F3 (243, em decimal), verde em intensidade 2A (42) e azul em intensidade 3C (60).

Caso fôssemos converter do sistema decimal para qualquer outra base, utilizaríamos a mesma metodologia indicada. Porém, esse método de divisões é aplicável somente para conversões do sistema decimal para outros. Isto se deve ao fato de usarmos a tabuada da divisão para números decimais. Posteriormente, veremos como converter de outros sistemas para o sistema decimal e, no tópico 2, as tabuadas de soma, subtração e multiplicação para o sistema binário.

A conversão de um número inteiro negativo seguiria o mesmo procedimento, resultando em outro número também negativo na outra base. A tarefa de conversão pode ser automatizada: o código a seguir em Python realiza a conversão de um número no sistema decimal para qualquer outra base. O algoritmo consiste em dividir pela base desejada enquanto for possível (*while quociente > 0*). Os restos são armazenados sequencialmente em uma variável de texto chamada **restos**, que é exibida em ordem inversa na saída (*restos[::-1]*).

```
valor=int(input("Digite o valor no Sistema Decimal: "))
base=int(input("Digite a base para qual se deseja converter: "))
quociente=abs(valor)
restos=""
while quociente > 0:
    restos+=str(quociente%base)
    quociente=quociente//base
if (valor > 0):
    print("{}10 = ({}){}".format(valor,restos[::-1],base))
else:
    print("{}10 = (-{}){}".format(valor,restos[::-1],base))
```

Figura 1 – Código em Python para a conversão de número inteiro do sistema decimal para outras bases

## 1.3 Conversão de outros sistemas para o sistema decimal

Para converter um número de outro sistema numérico para o sistema decimal, decompomos o número como mostrado em 1.1, lembrando que a base é a do sistema correspondente. Por exemplo, para convertermos o número  $(10010010)_2$ , transformaremos o número em uma somatória de potências de 2. Nos números inteiros o dígito mais à direita corresponde à base elevada a zero, aumentando em 1 para cada dígito à esquerda. Assim, a maior potência neste exemplo será  $2^7$ :

$$\begin{aligned}(10010010)_2 &= \\&= (1 \cdot 2^7) + (0 \cdot 2^6) + (0 \cdot 2^5) + (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (0 \cdot 2^0) = \\&= 128 + 0 + 0 + 16 + 0 + 0 + 2 + 1 = \\&= 146\end{aligned}$$

$$\text{Assim, } (10010010)_2 = (146)_{10}$$

De forma análoga, podemos converter valores que estejam em qualquer outra base para o sistema decimal. Novamente, o dígito mais à direita sempre será a base elevada ao expoente zero, aumentando em 1 para cada dígito à esquerda.

Exemplo: convertendo os números a seguir das bases indicadas para o sistema decimal.

$$\text{a) } (13265)_8$$

$$\begin{aligned}(13265)_8 &= (1 \cdot 8^4) + (3 \cdot 8^3) + (2 \cdot 8^2) + (6 \cdot 8^1) + (5 \cdot 8^0) = \\&= (1 \cdot 4.096) + (3 \cdot 512) + (2 \cdot 64) + (6 \cdot 8) + (5 \cdot 1) = \\&= 4.096 + 1.536 + 128 + 48 + 5 = (5813)_{10}\end{aligned}$$

$$\text{b) } (AC3D)_{16}$$

$$\begin{aligned}(AC3D)_{16} &= (A \cdot 16^3) + (C \cdot 16^2) + (3 \cdot 16^1) + (D \cdot 16^0) = \\&= (10 \cdot 16^3) + (12 \cdot 16^2) + (3 \cdot 16^1) + (13 \cdot 16^0) = \\&= (10 \cdot 4.096) + (12 \cdot 256) + (3 \cdot 16) + (13 \cdot 1) = \\&= 40.960 + 3.072 + 48 + 13 = (44093)_{10}\end{aligned}$$

c)  $(2654)_7$ ,

Embora a base 7 não seja utilizada em nenhuma situação prática, o processo de conversão é o mesmo:

$$\begin{aligned}(2654)_7 &= (2 \cdot 7^3) + (6 \cdot 7^2) + (5 \cdot 7^1) + (4 \cdot 7^0) = \\ &= (2 \cdot 343) + (6 \cdot 49) + (5 \cdot 7) + (4 \cdot 1) = \\ &= 686 + 294 + 35 + 4 = \mathbf{(1019)_{10}}\end{aligned}$$

Como na conversão de um número decimal para outra base, a conversão de um número inteiro negativo seguiria o mesmo procedimento, resultando em outro número negativo no sistema decimal.

A conversão também pode ser automatizada: o código a seguir em Python realiza a conversão de um número em outra base para o sistema decimal. O algoritmo consiste em armazenar o valor a ser convertido em uma variável de texto (**valor**), e cada caractere é convertido para o valor numérico correspondente e multiplicado pela base selecionada elevada ao expoente, que se inicia em zero e é incrementado em 1 a cada passagem. A variável **decimal** é somada a cada produto obtido, ficando o valor final ao término da execução.

```
valor=input("Digite o valor positivo a ser convertido: ")
base=int(input("Digite a base para qual se deseja converter: "))
expoente = 0
decimal=0
for i in range (len(valor)-1,-1,-1):
    decimal+=int(valor[i])*(base**expoente)
    expoente+=1
print("{}{}{} = {}10".format(valor,base,decimal))
```

Figura 2 – Código em Python para a conversão de número em outra base para o sistema decimal

## 1.4 Sistema binário para octal e hexadecimal

Os valores 8 e 16, que são as bases dos sistemas octal e hexadecimal, respectivamente, são potências de 2:  $8 = 2^3$  e  $16 = 2^4$ . Assim, é possível realizar as conversões do sistema binário para esses outros dois e vice-versa **sem a necessidade de passagem pelo sistema decimal**.

Para converter um número binário a uma das outras bases, separamos o número binário em grupos de dígitos, grupos de três dígitos para conversão ao sistema octal e de quatro dígitos para o hexadecimal, **da direita para a esquerda**, e simplesmente substituímos pelo valor equivalente no outro sistema, de acordo com a tabela a seguir.

**Tabela 2 – Representação de valores em diferentes sistemas numéricos**

Decimal	Binário	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Exemplo: convertendo os números a seguir do sistema binário para os sistemas octal e hexadecimal.

**a)  $(101100111)_2$**

Para octal, separamos o número em grupos de três dígitos:

$$(101\ 100\ 111)_2$$

$$(101)_2 = (5)_8$$

$$(100)_2 = (4)_8$$

$$(111)_2 = (7)_8$$

$$(101100111)_2 = (547)_8$$

Para hexadecimal, separamos o número em grupos de quatro dígitos:

$$(1\ 0110\ 0111)_2$$

$$(1)_2 = (1)_{16}$$

$$(0110)_2 = (6)_{16}$$

$$(0111)_2 = (7)_8$$

$$(101100111)_2 = (167)_{16}$$

$$\text{b) } (110111001000)_2$$

Para octal, separamos o número em grupos de três dígitos:

$$(110\ 111\ 001\ 000)_2$$

$$(110)_2 = (6)_8$$

$$(111)_2 = (7)_8$$

$$(001)_2 = (1)_8$$

$$(000)_2 = (0)_8$$

$$(110111001000)_2 = (6710)_8$$

Para hexadecimal, separamos o número em grupos de quatro dígitos:

$$(1101\ 1100\ 1000)_2$$

$$(1101)_2 = (D)_{16}$$

$$(1100)_2 = (C)_{16}$$

$$(1000)_2 = (8)_8$$

$$(101100111)_2 = (DC8)_{16}$$

Para converter dos sistemas octal e hexadecimal para o sistema binário, o caminho é o inverso: cada dígito do número será convertido em um conjunto de dígitos binários, três dígitos ao passar de octal para binário, e quatro dígitos ao passar de hexadecimal para binário; caso o número binário não tenha a quantidade de dígitos suficientes, devemos complementar com zeros à esquerda.

Exemplo: convertendo os números a seguir para o sistema binário.

$$\text{a) } (4321)_8$$

$$(4)_8 = (100)_2$$

$$(3)_8 = (011)_2$$



$$(2)_8 = (010)_2$$

$$(1)_8 = (001)_2$$

$$(4321)_8 = (100011010001)_2$$

$$\text{b) } (9F5A)_{16}$$

$$(9)_{16} = (1001)_2$$

$$(F)_{16} = (1111)_2$$

$$(5)_{16} = (0101)_2$$

$$(A)_{16} = (1010)_2$$

$$(9F5A)_{16} = (1001111101011010)_2$$

A conversão direta entre os sistemas octal e hexadecimal também pode ser feita de forma semelhante; neste caso, cada grupo de quatro dígitos no sistema octal corresponde a um grupo de três dígitos no sistema hexadecimal. Na prática, dados os valores elevados que resultam das potências envolvidas, é mais simples e rápido fazer a conversão passando de uma das bases para binário, e de binário para a outra base.

## 1.5 Bits e bytes

Na informática, a menor unidade de armazenamento possível é o *bit*, que pode armazenar um dígito binário (1 ou 0). A forma de armazenamento depende do meio de armazenamento, por exemplo, eletrônico, magnético ou óptico.

Um conjunto de 8 *bits* forma 1 *byte*. Assim sendo, um *byte* pode representar 256 ( $2^8$ ) valores diferentes. As palavras em um computador são geralmente agrupadas em conjuntos de *bytes*. Comumente, cada um desses conjuntos possui um valor correspondente a uma potência de 2 *bytes*: 1, 2, 4, 8, 16, ...

Como os computadores utilizam-se do sistema binário, os prefixos utilizados para indicar a ordem de grandeza são diferentes dos empregados pelo sistema internacional de medidas (SI). Logo, em vez de corresponder a uma variação de 1.000 ( $10^3$ ) vezes, cada prefixo equivale a uma variação de 1.024 ( $2^{10}$ ) vezes. Desta forma, por exemplo, 1 *kbyte* não corresponde a 1.000 *bytes*, mas a 1.024 *bytes*. A tabela a seguir apresenta os valores correspondentes aos principais prefixos utilizados na ciência da computação:

**Tabela 3 – Prefixos utilizados na ciência da computação**

Prefixo	Valor	Valor exato	Valor do prefixo no SI
Quilo (k)	$2^{10}$	1.024	$10^3$ (um mil)
Mega (M)	$2^{20}$	1.048.576	$10^6$ (um milhão)
Giga (G)	$2^{30}$	1.073.741.824	$10^9$ (um bilhão)
Tera (T)	$2^{40}$	1.099.511.627.776	$10^{12}$ (um trilhão)

## 2 CONVERSÃO DE NÚMEROS REAIS E OPERAÇÕES COM NÚMEROS BINÁRIOS

### 2.1 Conversão de números reais

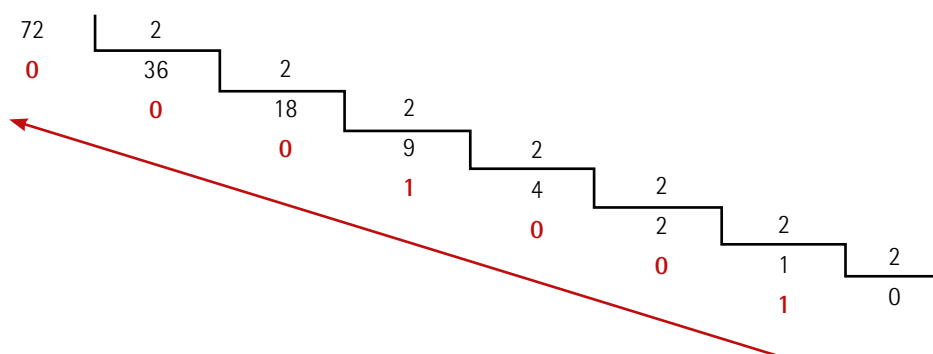
Para converter um número real não inteiro do sistema decimal para os demais sistemas, o processo é feito em duas etapas: a parte inteira do número é convertida como mostrado no tópico 1. Já a parte fracionária do número é convertida multiplicando-a pelo valor da base sucessivamente até chegar ao número cuja parte fracionária seja zero. A parte fracionária do número convertido corresponderá às partes inteiras dos resultados das multiplicações, **na ordem em que foram obtidas**.

Um detalhe importante é que **apenas a parte fracionária** do produto anterior é multiplicada novamente pela base.

Exemplo: convertendo o número  $(72,625)_{10}$  para os sistemas binário, octal e hexadecimal.

#### Sistema binário

Primeiramente, convertemos a parte inteira:



Assim,  $(72)_{10} = (1001000)_2$

Depois, convertemos a parte fracionária:

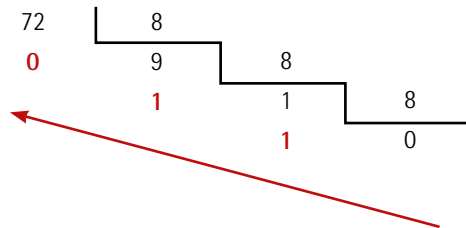
$$\begin{aligned}
 0,625 \cdot 2 &= 1,25 = 1 + 0,25 \\
 0,25 \cdot 2 &= 0,50 = 0 + 0,50 \\
 0,50 \cdot 2 &= 1,00 = 1 + 0,00
 \end{aligned}$$

Uma seta vermelha indica a leitura dos restos de cima para baixo, resultando na sequência binária 101.

Desta forma,  $(0,625)_{10} = (0,101)_2$ . Portanto,  $(72,625)_{10} = (1001000,101)_2$

## Sistema octal

Primeiramente, convertemos a parte inteira:



Assim,  $(72)_{10} = (110)_8$

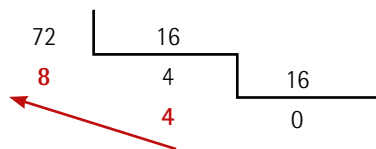
Depois, convertemos a parte fracionária:

$$0,625 \cdot 8 = 5,0 = 5 + 0,00 \downarrow$$

Desta forma,  $(0,625)_{10} = (0,5)_8$ . Portanto,  $(72,625)_{10} = (110,5)_8$

## Sistema hexadecimal

Primeiramente, convertemos a parte inteira:



Assim,  $(72)_{10} = (48)_{16}$

Depois, convertemos a parte fracionária:

$$0,625 \cdot 16 = 10,0 = 10 + 0,00 \downarrow$$

Como  $(10)_{10} = (A)_{16}$ , temos  $(0,625)_{10} = (0,A)_{16}$ . Portanto,  $(72,625)_{10} = (48,A)_{16}$



### Observação

Como as conversões para outras bases da parte inteira e da parte fracionária são feitas independentemente, não é obrigatório que a conversão da parte inteira seja feita primeiro.

### Conversão para o sistema decimal

A conversão de um número real de outro sistema para o sistema decimal é feita da mesma forma que a conversão de um número inteiro; porém, os dígitos à esquerda da vírgula contarão como expoentes negativos, decrescendo **da esquerda para a direita**.



#### Lembrete

Em qualquer sistema de numeração, o dígito imediatamente à esquerda da vírgula corresponderá à base elevada a zero. O expoente cresce para a esquerda e decresce para a direita.

Exemplo: convertendo os números a seguir para o sistema decimal.

a)  $(11011,0101)_2$

$$(11011,0101)_2 =$$

$$= (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) + (0 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + (0 \cdot 2^{-3}) =$$

$$= (1 \cdot 16) + (0 \cdot 8) + (0 \cdot 4) + (1 \cdot 2) + (1 \cdot 1) + (0 \cdot 0,5) + (1 \cdot 0,25) + (1 \cdot 0,125) =$$

$$= 16 + 0 + 0 + 2 + 1 + 0 + 0,25 + 0,125 = (19,375)_{10}$$

b)  $(567,431)_8$

$$(567,431)_8 =$$

$$= (5 \cdot 8^2) + (6 \cdot 8^1) + (7 \cdot 8^0) + (4 \cdot 8^{-1}) + (3 \cdot 8^{-2}) + (1 \cdot 8^{-3}) =$$

$$= (5 \cdot 64) + (6 \cdot 8) + (7 \cdot 1) + (4 \cdot 0,125) + (3 \cdot 0,015625) + (1 \cdot 0,001953125) =$$

$$= 320 + 48 + 7 + 0,5 + 0,046875 + 0,001953125 = (375,548828125)_{10}$$

c)  $(A34,BF)_{16}$

$$(A34,BF)_{16} =$$

$$= (A \cdot 16^2) + (3 \cdot 16^1) + (4 \cdot 16^0) + (B \cdot 16^{-1}) + (F \cdot 16^{-2}) =$$

$$= (10 \cdot 256) + (3 \cdot 16) + (4 \cdot 1) + (11 \cdot 0,0625) + (15 \cdot 0,00390625) =$$

$$= 2560 + 48 + 4 + 0,6875 + 0,05859375 = (2612,74609375)_{10}$$

## Dízimas periódicas

Como o número 10, que é a base do sistema decimal, não é uma potência de 2, muitos números ao serem convertidos para outros sistemas numéricos se tornam dízimas periódicas. Por exemplo, ao converter  $(0,3)_{10}$  para a base binária, uma dízima é obtida:

$$0,3 \cdot 2 = 0,6 = 0 + 0,6$$

$$0,6 \cdot 2 = 1,2 = 1 + 0,2$$

$$0,2 \cdot 2 = 0,4 = 0 + 0,4$$

$$0,4 \cdot 2 = 0,8 = 0 + 0,8$$

$$0,8 \cdot 2 = 1,6 = 1 + 0,6$$

$$0,6 \cdot 2 = 1,2 = 1 + 0,2$$

$$0,2 \cdot 2 = 0,4 = 0 + 0,4$$

$$0,4 \cdot 2 = 0,8 = 0 + 0,8$$

$$0,8 \cdot 2 = 1,6 = 1 + 0,6$$

É fácil constatar que a sequência "...1001..." irá repetir indefinidamente. Isto implica que, ao armazenar um número real convertido para o sistema binário, como cada número possui um limite de *bytes* em um computador, teremos que eliminar uma parte das infinitas casas decimais. Portanto, **o simples fato de armazenarmos um número na memória de um computador pode resultar em um erro.**



### Observação

Na matemática, erro significa uma diferença entre um valor exato e outro que está sendo utilizado em determinado cálculo.

## 2.2 Operações aritméticas com números binários

A ciência da computação se baseia em operações com números binários armazenados em *bits*; basicamente as operações são as mesmas operações aritméticas com as quais estamos acostumados no sistema decimal.

O princípio das operações aritméticas é o mesmo para qualquer base numérica: para podermos realizá-las, devemos entender como são as tabuadas das operações aritméticas no sistema binário.



b)  $(1111100,101)_2 + (1111110,11)_2$

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \phantom{,} \phantom{1} \phantom{0} \phantom{1} \\
 \phantom{+} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \phantom{,} \phantom{1} \phantom{0} \phantom{1} \\
 + \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \phantom{1} \phantom{,} \phantom{1} \phantom{1} \\
 \hline
 1 \phantom{0} 1 \phantom{0} 1 \phantom{1} 1 \phantom{0} 1 \phantom{1} \phantom{,} 0 \phantom{1} 1
 \end{array}$$

Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$(1111100,101)_2 + (1111110,11)_2 = (101111011,011)_2$

corresponde a:  $(124,625)_{10} + (254,75)_{10} = (379,375)_{10}$

## Subtração de números binários

No caso dos números binários, como o maior algarismo é 1, a **tabuada da subtração** fica da seguinte forma:

$0 - 0 = 0$

$1 - 0 = 1$

$10 - 1 = 1$

$11 - 10 = 1$

É possível constatar isso fazendo a conversão do sistema binário para o decimal, como foi feito anteriormente para a soma.

A subtração é realizada da mesma forma que no sistema decimal, da direita para a esquerda. Quando for necessário subtrair um número maior de um menor, por exemplo,  $0 - 1$ , "empresta-se" 10 do dígito seguinte e remove-se 1 deste dígito. Caso não seja possível, empresta-se do número imediatamente à esquerda para o dígito seguinte do número que está sendo subtraído. Para o caso de números com partes fracionárias, alinham-se as vírgulas e realiza-se a subtração da mesma maneira.

Exemplo: efetuando as subtrações dos números binários a seguir.

a)  $(100100111)_2 - (11110)_2$

$$\begin{array}{r}
 \phantom{-} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \\
 \phantom{-} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \\
 - \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \\
 \hline
 1 \phantom{0} 0 \phantom{0} 0 \phantom{0} 0 \phantom{1} 0 \phantom{0} 1
 \end{array}$$

Diagrama de empréstimo (borrowing) para a subtração binária:

- Do dígito 1 (posição 4) para o dígito 0 (posição 5):  $(-1)$  e  $(+10)$
- Do dígito 0 (posição 5) para o dígito 0 (posição 6):  $(-1)$  e  $(+10)$

Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$$(100100111)_2 - (11110)_2 = (100001001)_2$$

$$\text{corresponde a: } (295)_{10} - (30)_{10} = (265)_{10}$$

$$\text{b) } (11100,101)_2 - (1010,11)_2$$

$$\begin{array}{r} \phantom{11100,101} \\ - 1010,11 \\ \hline 10001,111 \end{array}$$

Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$$(11100,101)_2 - (1010,11)_2 = (10001,111)_2$$

$$\text{corresponde a: } (28,625)_{10} - (10,75)_{10} = (17,875)_{10}$$

## Multiplicação de números binários

Para os números binários, a **tabuada da multiplicação** fica da seguinte forma:

$$0 \times 0 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Ao realizar a multiplicação de dois números de diversos dígitos, procede-se como na multiplicação no sistema decimal: multiplica-se dígito a dígito, da direita para a esquerda, e a cada dígito desloca-se do novo produto um dígito para a esquerda; em seguida, somam-se todos os produtos. Caso os números possuam parte fracionária, contam-se quantos estão à direita da vírgula, nos dois números, e a vírgula é colocada nesta posição, contando a partir da direita dos números.

Exemplo: efetuando os produtos dos números binários a seguir.



a)  $(10111)_2 \times (1010)_2$

				1	0	1	1	1
x					1	0	1	0
				0	0	0	0	0
		1	0	1	1	1	1	0
	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0
1	1	1	0	0	1	1	0	

Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$$(10111)_2 \times (1010)_2 = (11100110)_2$$

$$\text{corresponde a: } (23)_{10} \times (10)_{10} = (230)_{10}$$

b)  $(111,001)_2 \times (10,1)_2$

				1	1	1,	0	0	1
x							1	0,	1
				1	1	1	0	0	1
		0	0	0	0	0	0	0	0
	1	1	1	0	0	0	1	0	0
1	0	0	0	1	1	1	1	0	1

Como os dois números que estão sendo multiplicados possuem, juntos, quatro dígitos à direita da vírgula, o produto deles também terá seus quatro últimos dígitos à direita da vírgula. Assim,  $(111,001)_2 \times (10,1)_2 = (10001,1101)_2$

Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$$(111,001)_2 \times (10,1)_2 = (10001,1101)_2$$

$$\text{corresponde a: } (7,125)_{10} \times (2,5)_{10} = (17,8125)_{10}$$

## 2.3 Números binários na forma de complemento de 2 (Comp-2)

Do ponto de vista computacional, realizar uma soma é mais fácil do que subtrair, uma vez que podemos tratar cada dígito de cada número separadamente, sem a necessidade de "emprestar um" de dígitos seguintes. Assim, o ideal seria podermos realizar uma subtração na forma de soma de um valor positivo com um valor negativo: por exemplo,  $A - B$  se tornaria  $A + (-B)$ , que possui o mesmo resultado aritmético.

Mas isto levanta a questão de como armazenar os valores negativos. Como os números são armazenados computacionalmente em um conjunto de *bits* de tamanho fixo, a forma mais simples seria utilizar o primeiro *bit* do conjunto para representar o sinal.

Vamos supor um número inteiro armazenado em um conjunto de 4 *bits*: cada número terá exatamente quatro dígitos, sendo que o primeiro representará o sinal. Vamos definir que o primeiro dígito é o sinal, sendo zero para um valor positivo e 1 para um valor negativo. Assim, poderíamos armazenar  $2^4 = 16$  valores, conforme mostrado na tabela a seguir:

**Tabela 4 – Representação de valores em um sistema binário convencional com 4 bits**

Decimal	Binário	Decimal	Binário
-7	1111	+0	0000
-6	1110	+1	0001
-5	1101	+2	0010
-4	1100	+3	0011
-3	1011	+4	0100
-2	1010	+5	0101
-1	1001	+6	0110
-0	1000	+7	0111

Ao analisarmos a tabela, dois problemas surgem. O primeiro e mais evidente é que existem duas formas de representação do valor zero, uma positiva e outra negativa, sendo que, por definição, tal valor não é nem positivo nem negativo.

O segundo problema ocorre quando tentamos somar um número positivo com um negativo. Por exemplo, se tentarmos fazer a operação  $6 + (-4)$ , ao utilizarmos as regras de soma já vistas, teremos:

$$\begin{array}{r}
 \phantom{+} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \\
 \phantom{+} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \\
 + \phantom{0} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \\
 \hline
 1 \phantom{0} \phantom{0} \phantom{1} \phantom{1}
 \end{array}$$

Mesmo se desconsiderarmos o quinto dígito que surgiu, pela tabela anterior vemos que  $0011 = (+3)_{10}$ , o que obviamente está incorreto. Se invertermos a notação do *bit* do sinal, zero passando a indicar negativo e 1 passando a indicar positivo, o resultado não irá se alterar. Se desconsiderarmos o *bit* do sinal e realizarmos a soma, ainda resultará em um valor numericamente incorreto:

$$\begin{array}{r}
 \phantom{+} \phantom{1} \phantom{1} \phantom{1} \\
 \phantom{+} \phantom{1} \phantom{0} \phantom{0} \\
 + \phantom{1} \phantom{0} \phantom{1} \phantom{1} \\
 \hline
 1 \phantom{0} \phantom{1} \phantom{1}
 \end{array}$$

A solução para que possamos somar adequadamente um número positivo com um negativo é utilizar a representação binária denominada **Complemento de 2**, ou simplesmente **Comp-2**. Consequentemente, todos os números terão a mesma quantidade de dígitos, que será definida pela quantidade de *bits* na qual o número será armazenado.

Para utilizarmos essa notação, o procedimento é o indicado a seguir.

### Números não negativos

Simplesmente fazemos a conversão para o sistema binário, complementando com zeros à esquerda para atingir o número de *bits*, caso seja necessário.

Para converter um número não negativo dessa notação para o sistema decimal, segue-se o procedimento já visto no tópico 1.

### Números negativos

Fazemos a conversão do módulo do número, desconsiderando o sinal, como descrito anteriormente, e as seguintes etapas adicionais:

- Invertamos os valores dos dígitos (1 torna-se zero e vice-versa).
- Soma-se 1 ao valor obtido na etapa anterior.

Comp-2 para o sistema decimal:

- Invertamos os valores dos dígitos (1 torna-se zero e vice-versa).
- Soma-se 1 ao valor obtido na etapa anterior.

Teremos então o módulo do número no sistema binário padrão, que pode ser convertido normalmente para o sistema decimal.



### Observação

Esta forma de representação numérica (Comp-2) se aplica somente a números inteiros.

Neste formato, todos os números não negativos ficarão com o primeiro dígito (mais à esquerda) sendo zero, e todos os números negativos ficarão com o primeiro dígito sendo 1.

Exemplo: convertendo os números do sistema decimal para Comp-2 de 1 byte (8 bits).

a)  $(112)_{10}$

Fazendo a conversão como mostrada no tópico 1, temos que  $(112)_{10} = (1110000)_2$ . Adicionando um zero à esquerda para completar os 8 bits, temos  $(112)_{10} = (01110000)_{\text{Comp-2}}$

b)  $(-93)_{10}$

Fazendo a conversão para o sistema binário do módulo do valor, temos que  $(93)_{10} = (1011101)_2$ . Adicionando um zero à esquerda para completar os 8 bits, temos 01011101.

A próxima etapa é inverter os valores (veja que não se trata de inverter a ordem dos dígitos, mas **cada dígito**):

01011101 se torna 10100010.

A seguir, somamos 1 ao valor e finalizamos a conversão:

$$\begin{array}{r}
 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \\
 + \\
 \hline
 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1
 \end{array}$$

Assim,  $(-93)_{10} = (10100011)_{\text{Comp-2}}$

Exemplo: convertendo os números da notação Comp-2 para o sistema decimal.

c)  $(1100111)_{\text{Comp-2}}$

Trata-se de um número negativo, pois inicia-se em 1, de 7 bits. Invertendo os sete dígitos, temos:

1100111 se torna 0011000

A seguir, somamos 1 ao valor e finalizamos a conversão:

$$\begin{array}{r}
 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \\
 + \\
 \hline
 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1
 \end{array}$$

Como  $(0011001)_2 = (25)_{10}$ , temos que  $(1100111)_{\text{Comp-2}} = (-25)_{10}$

d)  $(101101)_{\text{Comp-2}}$

Trata-se de um número negativo de 6 *bits*. Invertendo os seis dígitos, temos:

101101 se torna 010010.

A seguir, somamos 1 ao valor e finalizamos a conversão:

$$\begin{array}{rcccccc} & 0 & 1 & 0 & 0 & 1 & 0 \\ + & & & & & & 1 \\ \hline & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$$

Como  $(010011)_2 = (19)_{10}$ , temos que  $(101101)_{\text{Comp-2}} = (-19)_{10}$

A tabela a seguir apresenta todos os valores no formato Comp-2 de 4 *bits*. É importante observar que a representação dos números negativos irá ser diferente de acordo com a quantidade de *bits* utilizada no armazenamento.

**Tabela 5 – Representação de valores em um sistema binário Comp-2 com 4 *bits***

Decimal	Comp-2	Decimal	Comp-2
-8	1000	0	0000
-7	1001	1	0001
-6	1010	2	0010
-5	1011	3	0011
-4	1100	4	0100
-3	1101	5	0101
-2	1110	6	0110
-1	1111	7	0111

O código em Python na figura a seguir realiza a conversão de um número no sistema decimal para Comp-2, com uma quantidade de *bits* a ser definida pelo usuário.

```
bits=int(input("Digite o número de bits:"))
print("Número máximo a ser convertido: ", -(2**(bits-1)))
valor=int(input("Digite o valor negativo a ser convertido: "))
quociente=abs(valor)
restos=[]
while quociente > 0:
    restos.insert(0,quociente%2)
    quociente=quociente//2
while len(restos)<bits:
    restos.insert(0,0)
for i in range (len(restos)):
    if restos[i] == 1:
        restos[i]=0
    else:
        restos[i]=1
i=len(restos)-1
restos[i]+=1
while restos[i]>1:
    restos[i]=0
    if i>=0:
        restos[i-1]+=1
    i-=1
print("Convertido para Comp-2: ")
for i in restos:
    print(i,end="")
```

Figura 3 – Código em Python para a conversão de número inteiro do sistema decimal para a notação Comp-2

A vantagem dessa notação é que a soma entre números negativos e positivos se torna possível: simplesmente realizamos a soma binária como vista anteriormente e teremos o resultado na notação de Comp-2. Caso o resultado possua um dígito a mais que o limite do número de *bits*, desconsidera-se o dígito mais à esquerda.

Exemplo: realizando a soma de  $(102)_{10} + (-59)_{10}$  em um sistema Comp-2 de 1 byte.

Fazendo as conversões:

$$(102)_{10} = (01100110)_{\text{Comp-2}}$$

$$(-59)_{10} = (11000101)_{\text{Comp-2}}$$

Realizando a soma:

Fazendo a conversão para o sistema binário do módulo do valor, temos que  $(93)_{10} = (1011101)_2$ . Adicionando um zero à esquerda para completar os 8 *bits*, temos 01011101.

A próxima etapa é inverter os valores (veja que não se trata de inverter a ordem dos dígitos, mas **cada dígito**):

01011101 se torna 10100010.

A seguir, somamos 1 ao valor e finalizamos a conversão:

$$\begin{array}{r} \phantom{+} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{0} \\ \phantom{+} \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \phantom{0} \\ + \phantom{1} \phantom{0} \phantom{1} \phantom{1} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \\ \hline 1 \phantom{0} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \phantom{0} \phantom{1} \phantom{1} \end{array}$$

$$\text{Assim, } (01100110)_{\text{Comp-2}} + (11000101)_{\text{Comp-2}} = (00101011)_{\text{Comp-2}}$$

Como o primeiro dígito é zero, trata-se de um número positivo. Fazendo a conversão do sistema binário para o decimal, é possível verificar que:

$$(00101011)_{\text{Comp-2}} = (43)_{10}$$



## Saiba mais

A fim de saber como números são armazenados computacionalmente, leia o capítulo 6 do livro a seguir:

SEBESTA, R. W. *Conceitos de linguagens de programação*. Porto Alegre: Bookman, 2011.

Indicamos também a seguinte obra:

MOKARZEL, F.; SOMA, N. *Introdução à ciência da computação*. Rio de Janeiro: Campus, 2008.

## 3 INTRODUÇÃO À LÓGICA PROPOSICIONAL E TEORIA DAS PORTAS LÓGICAS

### 3.1 A lógica matemática

Os fundamentos da lógica remontam à Grécia Antiga, mais especificamente a Aristóteles (384 a.C.-322 a.C.), que definiu os conceitos de **termo**, **proposição** e **silogismo**. A lógica aristotélica, que era um ramo da filosofia, era tão completa que permaneceu quase inalterada até o século XIX, quando foi criada a **lógica matemática** por George Boole (1815-1864) e Augustus De Morgan (1806-1871). Com eles, a lógica passou a também ser um ramo da matemática; a lógica matemática é o fundamento de toda a teoria de circuitos lógicos digitais.

O fundamento da lógica, tanto da aristotélica quanto da lógica matemática, que também é chamada de lógica booleana ou álgebra de Boole, reside no conceito de proposição.



Figura 4 – A) George Boole e B) Augustus De Morgan

Disponível em: A) <https://bit.ly/3p9AGJ4>; B) <https://bit.ly/3uJ40fo>. Acesso em: 1º jun. 2021.

### Proposição

Uma proposição é uma sentença na qual afirma-se algo, normalmente implicando relação entre dois elementos, que são os **termos** da lógica aristotélica. Uma proposição tem que ser construída de tal forma que a ela possa ser atribuído um valor lógico **verdadeiro** (V, ou 1 na álgebra booleana) ou **falso** (F, ou 0). Ela deve satisfazer aos dois seguintes princípios fundamentais da lógica:

- **Princípio do terceiro excluído:** uma proposição somente pode ser verdadeira ou falsa, não havendo uma terceira alternativa possível.
- **Princípio da não contradição:** uma proposição não pode ser ao mesmo tempo verdadeira e falsa.

Assim, uma proposição somente pode ser verdadeira ou falsa, não podendo ter os dois valores lógicos simultaneamente, porém ela é capaz de alternar entre eles.

Exemplos de proposições:

- Dois mais dois é igual a três (proposição falsa).
- Um cubo possui seis faces (proposição verdadeira).
- A Terra não é plana (proposição verdadeira).





## Observação

Não são proposições perguntas, interjeições e frases nas quais parte dos elementos, sujeito ou predicado, sejam indefinidos. Por exemplo, a sentença "é difícil" (como definir verdadeiro ou falso neste caso?).

Usualmente as proposições são representadas por letras maiúsculas (A, B, C etc.).

## Tabela verdade

Uma tabela verdade é uma tabela na qual se listam, para uma ou mais proposições, todas as suas possíveis combinações de valores lógicos (verdadeiro ou falso). Em circuitos lógicos digitais, **utiliza-se 1 para representar verdadeiro e 0 para falso**.

Cada linha de uma tabela verdade representará também quaisquer eventuais associações entre proposições, as quais passaremos a chamar de **entradas**. O número máximo de linhas de uma tabela verdade será 2 elevado ao número de entradas na tabela; tal caso irá cobrir todas as possíveis combinações entre as entradas.

A tabela verdade também irá indicar o valor lógico de cada combinação entre as entradas que surjam por meio da associação de operadores lógicos, que serão apresentados a seguir. A tabela a seguir mostra as diferentes tabelas verdade para uma, duas e três entradas. Perceba que não há ocorrência de linhas redundantes em uma tabela verdade típica.

**Tabela 6 – Exemplos de tabelas verdade para uma, duas e três entradas**

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

## 3.2 Operadores lógicos

A lógica matemática se utiliza de diversos operadores para correlacionar as entradas entre si. A tabela a seguir ilustra os operadores utilizados em circuitos lógicos digitais e seus símbolos. Cada um desses operadores será detalhado adiante, tendo seu funcionamento explicado, além de ter a sua notação apresentada nesta disciplina.

**Tabela 7 – Operadores da lógica matemática utilizados em circuitos lógicos digitais**

Operador	Símbolo (lógica matemática)
Negação (não)	$\sim$
E (conjunção)	$\wedge$
OU (disjunção)	$\vee$
OU exclusivo (disjunção exclusiva)	$\oplus$

## Operador negação

O operador **negação** inverte o valor lógico de uma entrada. Sua notação é  $\sim A$  (lê-se "não A" ou "negação de A"). A tabela verdade a seguir ilustra os possíveis valores lógicos de uma entrada A e de sua negação:

**Tabela 8**

A	$\sim A$
0	1
1	0

É importante observar que a negação da negação corresponde à própria entrada, ou seja,  $\sim(\sim A) = A$ .

No presente material, usamos um traço sobre a entrada ou associação de entradas, assim a notação que será adotada de agora em diante será para indicar a negação da entrada A.

## Operador E

O operador lógico **E** associa duas proposições ou mais entradas na forma  $A \wedge B$  (lê-se "A e B"). **Esta operação somente terá valor lógico verdadeiro quando todas as entradas forem verdadeiras.**

Na álgebra booleana, essa operação pode ser representada por uma operação de multiplicação entre os valores lógicos das entradas associadas, como fica claro nas tabelas verdade a seguir para duas e três entradas.

**Tabela 9**

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 10

A	B	C	$A \wedge B \wedge C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

**Lembrete**

Pela multiplicação de binários, temos:

$$1 \times 1 = 1$$

$$0 \times 1 = 0$$

$$1 \times 1 \times 1 = 1$$

O operador E é comutativo e associativo, da mesma forma que a operação aritmética de multiplicação:

$$A \wedge B = B \wedge A$$

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

Em circuitos lógicos digitais, utiliza-se o símbolo da multiplicação para representar o operador: assim, a operação lógica  $A \wedge B$  será indicada por **A . B**. Esta será a notação adotada daqui em diante ao longo deste curso.

**Operador OU**

O operador lógico **OU** associa duas proposições ou mais entradas na forma  $A \vee B$  (lê-se "A ou B"). **Esta operação terá valor lógico verdadeiro quando ao menos uma das entradas for verdadeira, sendo falsa somente quando todas as entradas forem falsas.**

Na álgebra booleana, esta operação pode ser representada por uma soma entre os valores lógicos das entradas associadas, como fica claro nas tabelas verdade a seguir para duas e três entradas.

Tabela 11

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 12

A	B	C	$A \vee B \vee C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



## Observação

Como visto na soma de binários,  $1 + 1 = 10$ . No caso deste operador, qualquer soma que resulte em valor diferente de 0 (falso) será 1 (não falso, ou verdadeiro).

O operador OU é comutativo e associativo, da mesma forma que a operação aritmética de soma:

$$A \vee B = B \vee A$$

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

Em circuitos lógicos digitais, utiliza-se o símbolo da adição para representar esse operador: assim, a operação lógica  $A \vee B$  será indicada por **A + B**. Esta será a notação adotada daqui em diante ao longo deste curso.

## Operador OU exclusivo

O operador lógico **OU exclusivo** associa duas entradas na forma  $A \oplus B$  (lê-se "A ou exclusivo B", ou "ou A ou B"). Esta operação é verdadeira apenas quando somente uma das proposições for verdadeira, e é falsa quando ambas ou nenhuma das proposições forem verdadeiras.

A tabela verdade a seguir ilustra o funcionamento deste operador para duas entradas:

**Tabela 13**

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Uma regra mais ampla pode ser obtida para esse operador, como veremos mais adiante, sendo possível conectar mais de duas entradas por meio de tal operador. Assim, na tabela verdade a seguir, podemos verificar que  $A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$ , ou seja, o operador também é comutativo e associativo:

**Tabela 14**

A	B	C	$A \oplus B$	$(A \oplus B) \oplus C$	$B \oplus C$	$A \oplus (B \oplus C)$
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	0	0	0
1	0	0	1	1	0	1
1	0	1	1	0	1	0
1	1	0	0	0	1	0
1	1	1	0	1	0	1

Pelos resultados obtidos por meio da associatividade do operador, podemos expandir a regra de funcionamento para uma associação com qualquer quantidade de entradas: **a associação com OU exclusivo será verdadeira somente quando o número de entradas com valor lógico verdadeiro for ímpar.**

As tabelas a seguir apresentam um resumo do funcionamento dos quatro operadores lógicos, para duas e três entradas. As regras de funcionamento dos operadores exibidas podem ser expandidas para quaisquer quantidades de entradas.

**Tabela 15 – Resumo dos operadores lógicos para duas entradas**

A	B	$\bar{A}$	$\bar{B}$	$A \cdot B$	$A + B$	$A \oplus B$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

**Tabela 16 – Resumo dos operadores lógicos para três entradas**

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$A \cdot B \cdot C$	$A + B + C$	$A \oplus B \oplus C$
0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	1	1
0	1	1	1	0	0	0	1	0
1	0	0	0	1	1	0	1	1
1	0	1	0	1	0	0	1	0
1	1	0	0	0	1	0	1	0
1	1	1	0	0	0	1	1	1

## 3.3 Expressões lógicas

Uma expressão lógica é um conjunto de uma ou mais entradas, associadas, se necessário, por meio de operadores lógicos. Portanto, a menor expressão lógica possível será apenas uma entrada, sem nenhum operador atuando sobre ela.

Assim, como na aritmética, existe uma ordem de resolução (precedência) dos operadores. A ordem é a seguinte:

- parênteses, quando houver;
- negação;
- E, na ordem;
- OU e OU exclusivo, na ordem.

Uma expressão lógica irá resultar em uma única saída, a qual pode ser obtida a partir da tabela verdade. Para cada uma das combinações das entradas que compõem a expressão lógica, haverá um único valor lógico como resultado. Cada expressão pode ser resolvida por partes, como se fosse uma expressão algébrica ou aritmética, sendo essas partes associadas por meio do operador correspondente a fim de obter uma saída única. Podemos fazer a analogia com uma equação numérica: para cada valor das variáveis (entradas), a equação (expressão) apresenta um único valor como resultado.

Exemplo: montando as tabelas verdade para as expressões lógicas a seguir.

a)  $S = (P + \bar{Q}) \cdot (R + \bar{P})$

A ordem de resolução desta expressão lógica será:

$$\bar{Q}$$

$$(P + \bar{Q})$$

$$\bar{P}$$

$$(R + \bar{P})$$

$$S = (P + \bar{Q}) \cdot (R + \bar{P})$$

**Tabela 17**

P	Q	R	$\bar{Q}$	$(P + \bar{Q})$	P	$(R + \bar{P})$	S
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	0	0	1	1	0
0	1	1	0	0	1	1	0
1	0	0	1	1	0	0	0
1	0	1	1	1	0	1	1
1	1	0	0	1	0	0	0
1	1	1	0	1	0	1	1

b)  $S = \overline{(A + B)} \oplus \bar{C}$

A ordem de resolução desta expressão lógica será:

$$\bar{B}$$

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

$$\overline{(A + \bar{B})}$$

$$\bar{C}$$

$$S = \overline{(A + \bar{B})} \oplus \bar{C}$$

Tabela 18

A	B	C	$\bar{B}$	$(A + \bar{B})$	$\overline{(A + \bar{B})}$	$\bar{C}$	S
0	0	0	1	1	0	1	1
0	0	1	1	1	0	0	0
0	1	0	0	0	1	1	0
0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	1	1
1	1	1	0	1	0	0	0

c)  $S = (X \cdot \bar{W}) + (Y \oplus Z)$

A ordem de resolução desta expressão lógica será:

$\bar{W}$

$(X \cdot \bar{W})$

$(Y \oplus Z)$

$S = (X \cdot \bar{W}) + (Y \oplus Z)$

Tabela 19

X	Y	W	Z	$\bar{W}$	$(X \cdot \bar{W})$	$(Y \oplus Z)$	S
0	0	0	0	1	0	0	0
0	0	0	1	1	0	1	1
0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	1
0	1	0	0	1	0	1	1
0	1	0	1	1	0	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	0	0
1	0	0	0	1	1	0	1
1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	1	1	1	1
1	1	0	1	1	1	0	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	0	0



Podemos automatizar a resolução de uma expressão lógica por meio das linguagens de programação; para efeitos de exemplo será utilizado o Python, mas a maioria delas pode ser utilizada para tal finalidade.

Para isto, criaremos uma lista com os valores lógicos (*True* e *False*, no Python) e faremos um conjunto de comandos de repetição aninhados, sendo que cada entrada será a variável de controle de cada um desses comandos, variando dentro da lista criada anteriormente. A saída, dentro do último comando de repetição, será o valor lógico de cada entrada, com o valor da expressão lógica.

Este programa é bastante simples e grosseiro, uma vez que o código precisa ser alterado a cada nova expressão lógica, mas ilustra os conceitos de lógica matemática e de lógica de programação e algoritmos.

Os operadores lógicos funcionam da mesma forma que já foi descrita na maioria das linguagens de programação e possuem a precedência idêntica, embora cada linguagem possua sua notação. Para o Python, a notação dos operadores é mostrada no quadro a seguir.

**Quadro 1 – Resumo dos operadores lógicos para três entradas**

Operador lógico	Notação no Python
Negação	not
E	and
OU	or
OU exclusivo	^

Por exemplo, colocando a expressão lógica  $S = \overline{(A + B)} \oplus \bar{C}$ , resolvida anteriormente no **exemplo b**, o programa e sua saída ficariam como apresentado na figura a seguir. Não se esqueça de comparar ambas as tabelas verdade.

```
val=[False, True]
print("A\tB\tC\tSaída")
for A in val:
    for B in val:
        for C in val:
            print(A, "\t", B, "\t", C, "\t", (not(A or not B) ^ (not C)))
```

#### Saída do Programa:

A	B	C	Saída
False	False	False	True
False	False	True	False
False	True	False	False
False	True	True	True
True	False	False	True
True	False	True	False
True	True	False	True
True	True	True	False

>>>

Figura 5 – Programa em Python para a resolução da expressão lógica apresentada e sua respectiva saída

### 3.4 Portas lógicas

Portas lógicas são dispositivos que representam os operadores lógicos descritos e recebem uma ou mais entradas lógicas de entrada para produzir uma única saída, de acordo com a operação lógica correspondente. Elas são usadas em circuitos eletrônicos para representar as operações lógicas correspondentes.

Embora a lógica matemática remonte ao século XIX, foi somente pouco antes da Segunda Guerra Mundial que o norte-americano Claude Elwood Shannon (1916-2001) utilizou os fundamentos da álgebra booleana para a solução de problemas de circuitos publicando o trabalho denominado *Symbolic Analysis of Relay and Switching Circuits* (SHANNON, 1938), que é considerado o fundamento da eletrônica digital. Foi a partir deste trabalho que foram desenvolvidas as portas lógicas, que constituem a base de circuitos lógicos digitais.

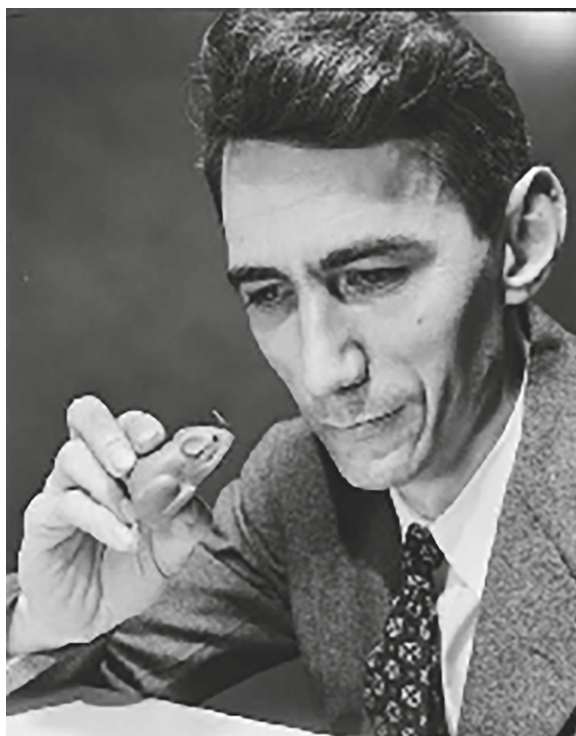


Figura 6 – Claude Shannon

Disponível em: <https://bit.ly/2TBQrNh>. Acesso em: 1º jun. 2021.

As portas lógicas são geralmente indicadas por seu nome em inglês. Elas, seus correspondentes na lógica matemática, bem como sua simbologia pela norma American National Standards Institute (ANSI) são apresentados no quadro a seguir. Embora existam outras simbologias, esta será a empregada ao longo de todo o curso.



## Saiba mais

Para conhecer o trabalho de Shannon, leia:

SHANNON, C. E. A symbolic analysis of relay and switching circuits. *Transactions American Institute of Electrical Engineers*, v. 57, 1938. Disponível em: <https://bit.ly/3whYPQk>. Acesso em: 9 jun. 2021.

## Quadro 2 – Portas lógicas e seus operadores lógicos correspondentes

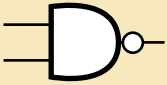
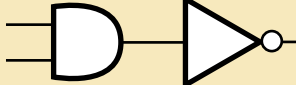

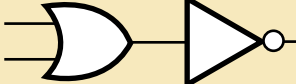

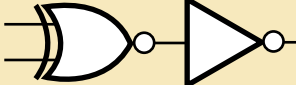
Porta lógica	Símbolo	Operação
NOT (Negação)		$\bar{A}$
AND (E)		$A \cdot B$
OR (OU)		$A + B$
XOR (OU exclusivo)		$A \oplus B$
NAND (Negação de E)		$\overline{A \cdot B}$
NOR (Negação de OU)		$\overline{A + B}$
NXOR (Negação de OU exclusivo)		$\overline{A \oplus B}$

Adaptado de: Wikimedia (s.d)a.

Os terminais à esquerda de cada porta lógica são as entradas, e o terminal à direita representa a saída única de cada uma delas. Relembrando: cada porta lógica terá uma única saída, independentemente da quantidade de entradas que tiver.

As três últimas portas lógicas são as negações (valores lógicos invertidos; por este motivo a porta **NOT** também é chamada de inversor) das portas lógicas (a porta **NXOR** também é denominada de porta coincidência, pois é verdadeira quando as entradas assumem valores iguais, isto é, quando elas coincidem). Desta forma, elas equivalem a conectarmos uma porta **NOT** na saída da porta lógica correspondente ao operador. Tal associação terá ainda apenas uma saída, conforme indicado no quadro a seguir. Assim, montando a tabela verdade para todas as portas lógicas, teremos:

**Quadro 3 – Portas lógicas e seus operadores lógicos correspondentes**

Porta lógica	Símbolo	Circuito equivalente
NAND		
NOR		
NXOR		

Adaptado de: Wikimedia (s.d)a.

Assim, montando a tabela verdade para todas as sete portas lógicas, pelas regras dos operadores apresentadas anteriormente, o resultado é o indicado na tabela a seguir (note que na verdade trata-se da expansão da tabela "Operadores da lógica matemática utilizados em circuitos lógicos digitais").

**Tabela 20 – Resumo das portas lógicas para duas entradas**

A	B	NOT A	NOT B	A AND B	A OR B	A NAND B	A NOR B	A XOR B	A NXOR B
1	1	0	0	1	1	0	0	0	1
1	0	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0	1	0
0	0	1	1	0	0	1	1	0	1

É importante observar que embora cada uma das portas lógicas individualmente, por representar um operador lógico, seja comutativa, a associação de duas ou mais delas não o é.

## 3.5 Associação de portas lógicas

Duas ou mais portas lógicas podem ser associadas, formando um circuito mais complexo. Ele será detalhado ao longo dos tópicos seguintes, mas aqui serão explicados os fundamentos de ligação delas.

Cada circuito lógico pode ser expresso por uma expressão lógica, que possuirá uma ou mais entradas e apenas uma saída. Algumas observações podem ser feitas em relação à associação de portas lógicas:

- As entradas de uma associação de portas lógicas serão as entradas básicas do circuito ou as saídas de outras portas lógicas, e sempre haverá apenas uma única saída para cada porta lógica.
- Quando houver a negação de uma ou mais entradas, as quais estejam ligadas por uma porta lógica, a porta lógica da negação (NOT) virá antes da entrada da porta lógica que associa as duas proposições.

A construção de um circuito lógico pode ser feita da forma inversa à qual resolvemos uma tabela verdade: a última operação lógica será a saída final do circuito, e suas entradas serão definidas pelas expressões que este operador conecta. Repetimos esse processo até chegarmos nas entradas originais do circuito.

Ao fazermos o inverso, ou seja, construir a tabela verdade de uma associação de portas lógicas, haverá uma coluna para cada entrada, e uma coluna para cada porta lógica. O processo de obtenção da expressão lógica será análogo: partiremos da última porta lógica, saída final do circuito, em direção às entradas originais do circuito.



### Observação

Para obter o circuito lógico correspondente a uma dada expressão lógica, não é necessário resolver a tabela verdade da expressão.

Exemplo: montando os circuitos lógicos e as tabelas verdade correspondentes às expressões lógicas a seguir.

Como podemos observar, conforme o número de entradas e portas lógicas aumenta, torna-se cada vez mais difícil, se não impossível, evitar o cruzamento entre as linhas que conectam os diferentes componentes do circuito. Isto dificulta a visualização e compreensão do circuito. Uma forma de evitar que isto ocorra e facilitar a visualização é, no desenho do circuito, colocar as entradas em linhas verticais, com as portas lógicas conectando-se horizontalmente nelas.

Assim, redesenhando os circuitos dos três últimos exemplos, teremos:

$$S = (M + \bar{N} + O) \cdot M \cdot (N \oplus \bar{O})$$

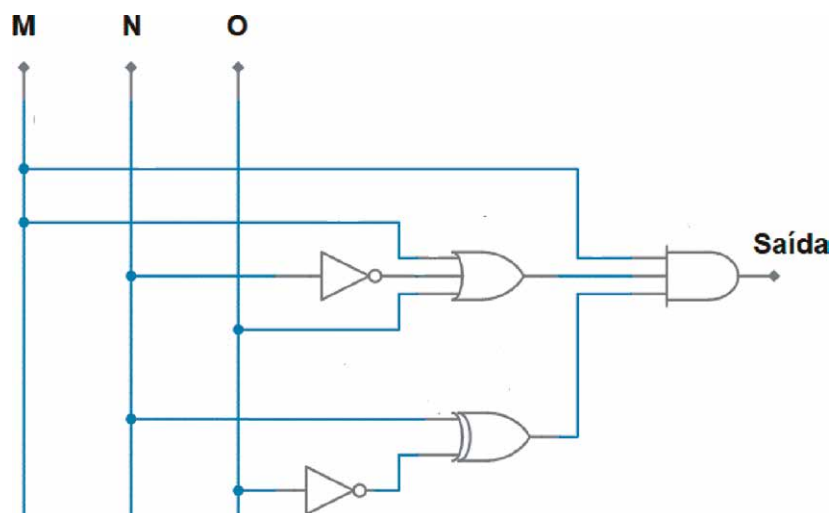


Figura 7

$$S = (X + Y) + (W \cdot Z) + (\bar{Y} \oplus W \cdot \bar{X})$$

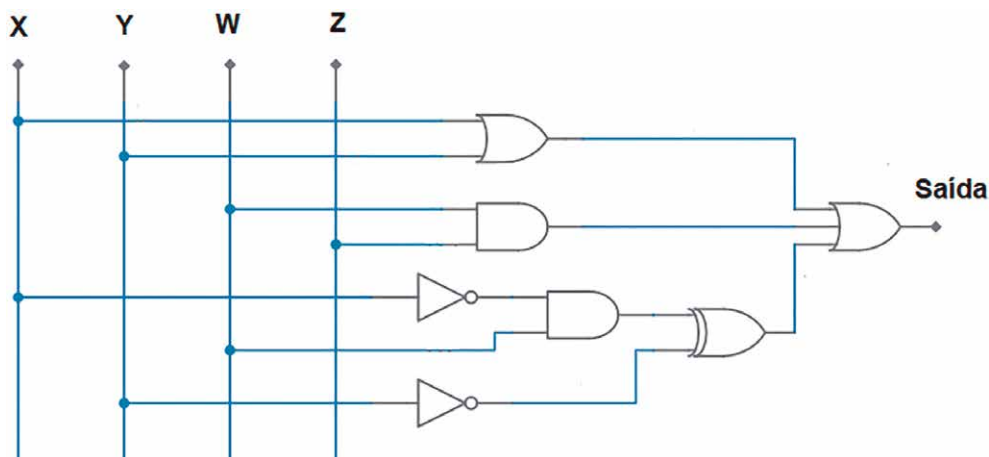


Figura 8

$$S = (A \oplus B) + (C \cdot D) \cdot (A \oplus D) + (C \cdot B)$$

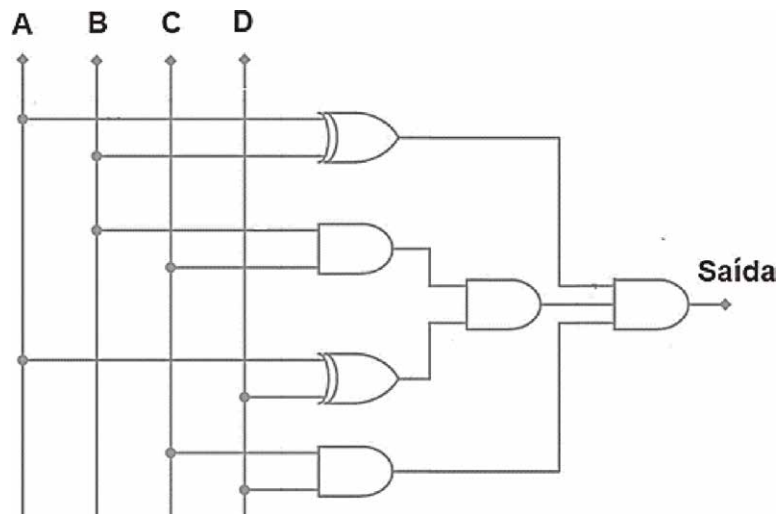


Figura 9

Cada bolinha indica que aquelas duas linhas estão conectadas, enquanto um cruzamento sem ela significa que as duas linhas não se conectam. Ao longo deste material serão utilizadas ambas as formas apresentadas de se desenhar um circuito, de acordo com o tamanho e número de portas lógicas de cada um.

## 4 CONSTRUÇÃO E SIMPLIFICAÇÃO DE CIRCUITOS LÓGICOS

Uma expressão lógica e, por consequência, seu circuito lógico correspondente podem ter infinitas **expressões equivalentes**, ou seja, que apresentam uma saída igual para todas as combinações de entradas. **Dois expressões equivalentes irão produzir a mesma tabela verdade**, isto é, a saída

de cada linha da tabela será igual, supondo, é claro, que a sequência das entradas seja a mesma em ambas as tabelas. Do ponto de vista de circuitos lógicos digitais, é interessante que sempre seja utilizado o menor circuito possível (com menos portas lógicas): um circuito menor significa menos custo, além de menor consumo de energia, aquecimento e tempo de resposta.

Assim, serão apresentadas as principais regras de equivalência lógica, visando, sempre que possível, substituir uma expressão lógica por uma equivalente que possua menos operadores (e portas lógicas).

## 4.1 Leis da lógica

Existem algumas leis que regem a manipulação de expressões lógicas e seus operadores. Elas estabelecem a equivalência (indicada pelo símbolo  $\equiv$ ) entre duas expressões. As leis mais importantes envolvendo os operadores apresentados serão mostradas a seguir e demonstradas pela construção de suas respectivas tabelas verdade quando necessário.



### Observação

A lógica matemática possui outros operadores que não os mostrados no tópico anterior, para os quais não existe uma porta lógica correspondente. Tais operadores não serão abordados neste livro-texto.

#### Lei idempotente

$$A + A \equiv A$$

$$A \cdot A \equiv A$$

Uma entrada ligada a si própria por uma porta **AND** ou por uma porta **OR** irá resultar na própria entrada, sendo desnecessária a porta lógica. Esse raciocínio pode ser expandido para conectar a mesma entrada com si múltiplas vezes, utilizando uma dessas duas portas lógicas com três ou mais entradas.



Figura 10 – Os circuitos lógicos  $A + A$  e  $A \cdot A$  são equivalentes e ambos equivalem simplesmente à entrada  $A$

#### Lei da absorção

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

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

Tabela 21 – Demonstração da lei da absorção

A	B	$(A + B)$	$(A + B) \cdot A$	$(A \cdot B)$	$(A \cdot B) + A$
0	0	0	0	0	0
0	1	1	0	1	0
1	0	1	1	1	1
1	1	1	1	1	1

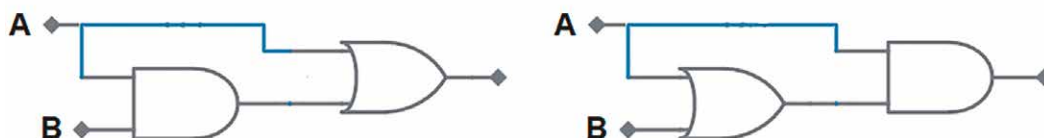


Figura 11 – Os circuitos lógicos  $(A + B) \cdot A$  e  $(A \cdot B) + A$  são equivalentes e ambos equivalem simplesmente à entrada A

### Lei da negação

$$\overline{(\overline{A})} \equiv A$$

Esta equivalência é bastante intuitiva: a negação de uma negação é a própria expressão lógica.

### Lei associativa

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

$$(A + B) + C \equiv A + (B + C) \equiv A + B + C$$

Esta equivalência já foi demonstrada no tópico 3, na apresentação dos operadores lógicos.

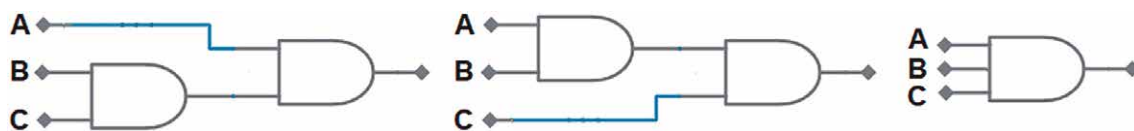


Figura 12 – Um conjunto de entradas pode ser ligado com portas AND com qualquer um desses arranjos, que são equivalentes

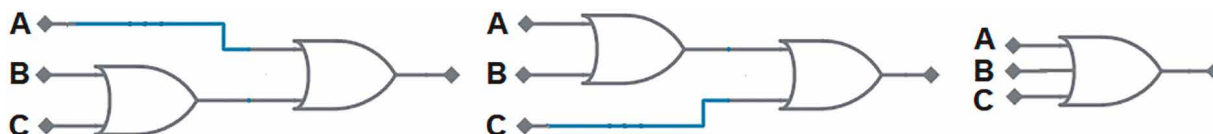


Figura 13 – Um conjunto de entradas pode ser ligado com portas OR com qualquer um desses arranjos, que são equivalentes



## Lei de De Morgan

$$\overline{A + B} \equiv \overline{A} \cdot \overline{B}$$

$$\overline{A \cdot B} \equiv \overline{A} + \overline{B}$$

A lei que recebe o nome de Augustus De Morgan é uma das mais importantes relações de equivalência lógica, e não é intuitiva: **a associação de duas negações com uma porta lógica não é a negação da associação das duas entradas com a mesma porta lógica**, como pensaríamos usando a propriedade comutativa.

**Tabela 22 – Demonstração da lei de De Morgan para  $\overline{A + B} \equiv \overline{A} \cdot \overline{B}$**

A	B	$\overline{A}$	$\overline{B}$	$\overline{A + B}$	$A \cdot B$	$\overline{A \cdot B}$
0	0	1	1	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	0	0	0	1	0

**Tabela 23 – Demonstração da lei de De Morgan para  $\overline{A \cdot B} \equiv \overline{A} + \overline{B}$**

A	B	$\overline{A}$	$\overline{B}$	$\overline{A \cdot B}$	$A + B$	$\overline{A + B}$
0	0	1	1	1	0	1
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	1	0	0	0	1	0

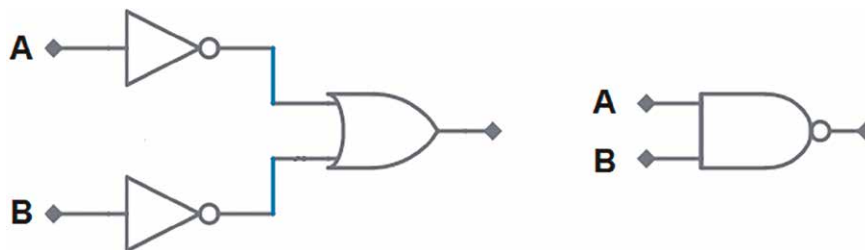


Figura 14 – A associação de duas negações por meio de uma porta OR é equivalente à associação das duas entradas (sem negação) por uma porta NAND

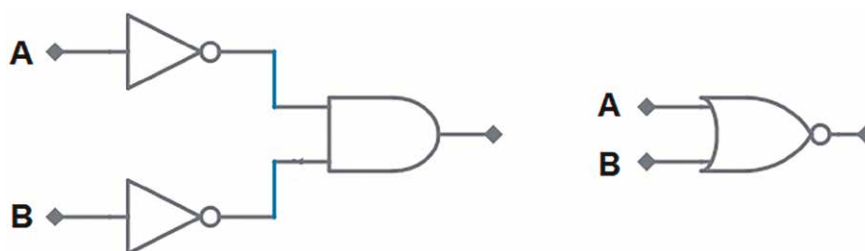


Figura 15 – A associação de duas negações por meio de uma porta AND é equivalente à associação das duas entradas (sem negação) por uma porta NOR

## Lei distributiva

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

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

Esta lei permite dividir a associação de três entradas em um par de associações conectadas. As tabelas verdade e os circuitos equivalentes são apresentados a seguir:

**Tabela 24 – Demonstração da lei distributiva para  $A + (B \cdot C) \equiv (A + B) \cdot (A + C)$**

A	B	C	$B \cdot C$	$A + (B \cdot C)$	$(A + B)$	$(A + C)$	$(A + B) \cdot (A + C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

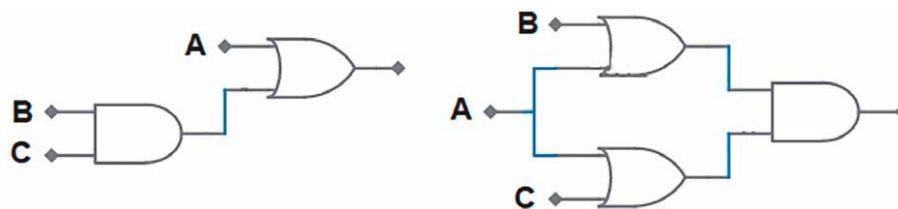


Figura 16 – O circuito lógico  $A + (B \cdot C)$  é equivalente ao circuito lógico  $(A + B) \cdot (A + C)$

**Tabela 25 – Demonstração da lei distributiva para  $A \cdot (B + C) \equiv (A \cdot B) + (A \cdot C)$**

A	B	C	$B + C$	$A \cdot (B + C)$	$(A \cdot B)$	$(A \cdot C)$	$(A \cdot B) + (A \cdot C)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

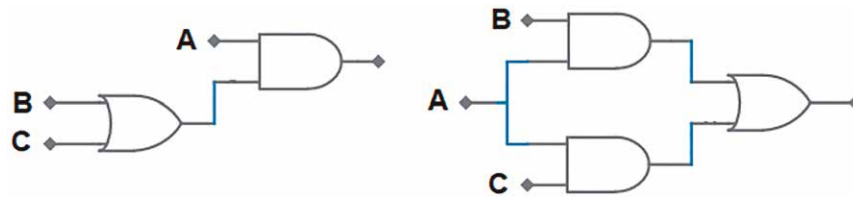


Figura 17 – O circuito lógico  $A \cdot (B + C)$  é equivalente ao circuito lógico  $(A \cdot B) + (A \cdot C)$

## Lei da identidade

$$A \cdot T \equiv A$$

$$A + C \equiv A$$

Para entendermos a lei da identidade, é necessário explicar os conceitos de **tautologia** e **contradição**. Na lógica matemática, a tautologia é uma expressão lógica para a qual todas as saídas são verdadeiras, para quaisquer combinações de entradas, por exemplo, a expressão  $A + \bar{A}$ ; a contradição é o oposto, uma expressão lógica para a qual todas as saídas são falsas, para quaisquer combinações de entradas, por exemplo, a expressão  $A \cdot \bar{A}$ .

Assim, se associarmos uma entrada ou expressão a uma tautologia por meio de uma porta lógica AND, o valor lógico dessa associação será o mesmo da entrada ou expressão que foi associada. De forma análoga, ocorrerá o mesmo se uma entrada ou expressão lógica for associada a uma contradição por meio de uma porta lógica OR.

**Tabela 26 – Demonstração da lei da identidade**

A	Tautologia (T)	Contradição (C)	$A \cdot T$	$A + C$
0	1	0	0	0
1	1	0	1	1

Esta lei, conforme visto na tabela, pode ser interpretada como:

$$A \cdot 1 \equiv A$$

$$A + 0 \equiv A$$

1 representa uma tautologia

0 representa uma contradição

## Lei complementar

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

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

Ao associar-se uma entrada com sua própria negação, temos 0 no caso da conjunção (E) e 1 no caso da disjunção (OU):

**Tabela 27 – Demonstração da lei complementar**

A	$\bar{A}$	$A \cdot \bar{A}$	$A + \bar{A}$
0	1	0	1
1	0	0	1

## Equivalência do OU exclusivo

$$A \oplus B \equiv (\bar{A} \cdot B) + (A \cdot \bar{B})$$

$$\overline{A \oplus B} \equiv (A \cdot B) + (\bar{A} \cdot \bar{B})$$

O operador OU exclusivo, bem como sua negação (portas lógicas XOR e NXOR, respectivamente), pode ser substituído por um conjunto de portas lógicas NOT, AND e OR.

**Tabela 28 – Demonstração da equivalência do OU exclusivo**

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot B$	$A \cdot \bar{B}$	$(\bar{A} \cdot B) + (A \cdot \bar{B})$	$A \oplus B$
0	0	1	1	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0

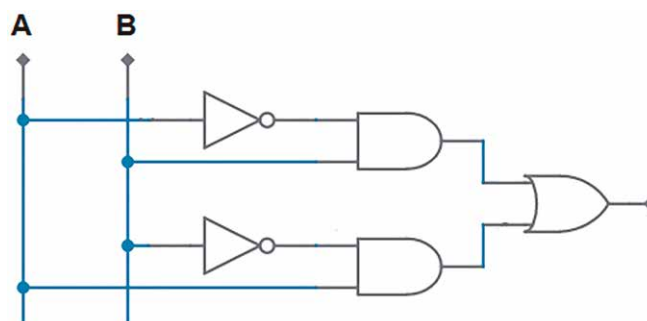


Figura 18 – Circuito lógico equivalente à porta lógica XOR

**Tabela 29 – Demonstração da equivalência da negação do OU exclusivo**

A	B	$\bar{A}$	$\bar{B}$	$A \cdot B$	$\bar{A} \cdot \bar{B}$	$(A \cdot B) + (\bar{A} \cdot \bar{B})$	$A \oplus B$
0	0	1	1	0	1	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	1	0	0	1	0	1	1

Podemos aplicar a lei de De Morgan no termo  $(\bar{A} \cdot \bar{B})$ , substituindo-o por  $\overline{(A + B)}$  e simplificando ainda mais o circuito. Assim, ficaria  $A \oplus B = (A \cdot B) + \overline{(A + B)}$ .

## Circuitos lógicos

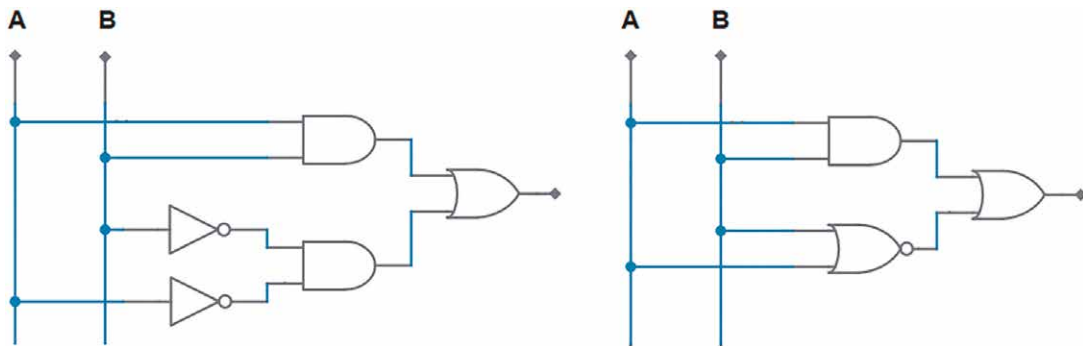


Figura 19 – Circuitos lógicos equivalentes à porta lógica NXOR

Todas as leis da lógica apresentadas são uma equivalência de duas direções: podemos sempre substituir uma expressão por outra equivalente. Usualmente, procuramos reduzir o número de portas lógicas para obter um circuito lógico menor, mas em alguns casos, durante a resolução de uma expressão lógica, podemos realizar a operação inversa, visando à substituição de outra parte do circuito por uma equivalente.

Além das regras de equivalência apresentadas, uma forma de simplificar expressões lógicas que contenham apenas as portas lógicas NOT, AND e OR é por redução algébrica, exatamente como se os operadores lógicos fossem os operadores aritméticos correspondentes.

Exemplo: consideremos a expressão lógica  $(A \cdot B \cdot C) + (A \cdot D)$ . Como a entrada A é comum a ambos os termos que estão conectados pela "soma" (porta lógica OR), podemos isolá-la, obtendo assim:

$$(A \cdot B \cdot C) + (A \cdot D) \equiv A \cdot (B \cdot C + D)$$

A seguir, veremos como aplicar essas regras de equivalência para simplificar expressões e circuitos lógicos.

### 4.2 Obtenção e simplificação de circuitos lógicos

A situação mais comum ao se projetar circuitos lógicos é partir da saída desejada, obtendo e construindo o circuito que forneça essa saída. A forma mais simples de se obter uma expressão lógica é através da própria tabela verdade da saída desejada. Este método, embora garanta um circuito que atenda às necessidades, tem a desvantagem de resultar em um circuito muitas vezes grande e redundante, com um excesso de portas lógicas e de difícil simplificação ou redução.

Tal método pode ser implementado em poucas etapas:

- Montagem da tabela verdade para o circuito desejado.
- Identificação na tabela das linhas para as quais a saída tenha valor lógico 1.
- Para cada linha corresponderá uma associação de todas as entradas por meio da porta lógica AND.
- Para cada associação obtida no item anterior, caso uma entrada tenha valor lógico 0, a entrada receberá uma negação.
- Todas as expressões obtidas serão conectadas por meio de uma porta OR.



#### Lembrete

As portas lógicas XOR e NXOR podem ser substituídas por associações de portas NOT, AND e OR, sendo possível construir qualquer circuito apenas com essas três portas lógicas.

Uma forma mais elaborada, mas com a vantagem de obter circuitos menores, são os mapas de Karnaugh, que serão apresentados no tópico 5.

Exemplo: obtendo a expressão lógica e a tabela verdade, deve-se simplificar a expressão quando possível e desenhar o circuito lógico para os casos a seguir.

a) Um sistema de alarme possui três sensores S1, S2 e S3. O alarme será acionado (saída igual a 1) quando dois ou mais sensores forem ativados (valor lógico dos sensores igual a 1).

Assim, pelo funcionamento descrito, a tabela verdade para o circuito será:

**Tabela 30**

S1	S2	S3	Saída
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Identificamos agora as linhas para as quais a saída é igual a 1 e a expressão lógica correspondente:

**Tabela 31**

S1	S2	S3	Saída	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{S1} \cdot S2 \cdot S3$
1	0	0	0	
1	0	1	1	$S1 \cdot \overline{S2} \cdot S3$
1	1	0	1	$S1 \cdot S2 \cdot \overline{S3}$
1	1	1	1	$S1 \cdot S2 \cdot S3$

Assim, uma expressão lógica que resultará na saída desejada será obtida pela junção das quatro expressões unidas por uma porta OR:

$$\text{Saída} = (\overline{S1} \cdot S2 \cdot S3) + (S1 \cdot \overline{S2} \cdot S3) + (S1 \cdot S2 \cdot \overline{S3}) + (S1 \cdot S2 \cdot S3)$$

Para simplificar a expressão, podemos aplicar a lei distributiva nos dois primeiros termos, isolando S3:

$$(\overline{S1} \cdot S2 \cdot S3) + (S1 \cdot \overline{S2} \cdot S3) \equiv S3 \cdot (\overline{S1} \cdot S2 \cdot S1 \cdot \overline{S2})$$

Aplicando a equivalência do OU exclusivo:

$$S3 \cdot (\overline{S1} \cdot S2 \cdot S1 \cdot \overline{S2}) = S3 \cdot (S1 \oplus S2)$$

Aplicando novamente a lei distributiva no terceiro e quarto termos da expressão original, isolando S1 . S2:

$$(S1 \cdot S2 \cdot \overline{S3}) + (S1 \cdot S2 \cdot S3) \equiv (S1 \cdot S2) \cdot (\overline{S3} + S3)$$

Como  $(\overline{S3} + S3)$  é uma tautologia, podemos aplicar a lei da identidade e eliminar esse termo, obtendo:

$$(S1 \cdot S2) \cdot (\overline{S3} + S3) \equiv (S1 \cdot S2)$$

Juntando novamente, teremos:

$$\text{Saída} = S3 \cdot (S1 \oplus S2) + (S1 \cdot S2)$$

Podemos verificar se as simplificações estão corretas por meio da construção da tabela verdade da expressão:

**Tabela 32**

S1	S2	S3	$S1 \oplus S2$	$S3 \cdot (S1 \oplus S2)$	$(S1 \cdot S2)$	Saída
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	0
0	1	1	1	1	0	1
1	0	0	1	0	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	0	0	1	1

Assim, o circuito lógico ficará:

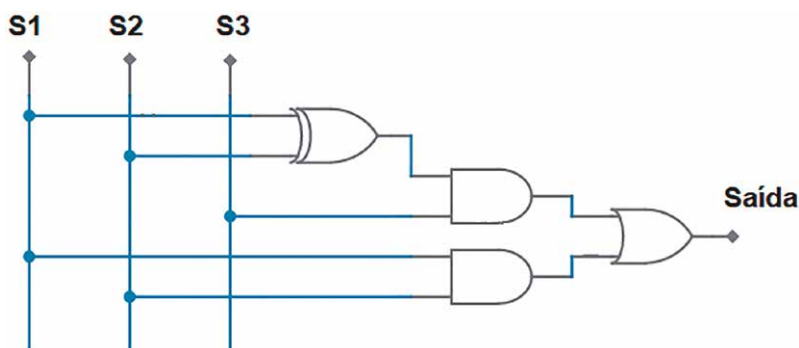


Figura 20

b) Um circuito lógico com três entradas P, Q e R possui a seguinte regra de funcionamento:

Quando  $P = Q$ , a saída será igual a R.

Quando  $P \neq Q$ , a saída será igual à negação de R.



Assim, pelo funcionamento descrito, a tabela verdade para este circuito será:

**Tabela 33**

P	Q	R	Saída
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Identificamos agora as linhas para as quais a saída é igual a 1 e a expressão lógica correspondente:

**Tabela 34**

P	Q	R	Saída	
0	0	0	0	
0	0	1	1	$\bar{P} \cdot \bar{Q} \cdot R$
0	1	0	1	$\bar{P} \cdot Q \cdot \bar{R}$
0	1	1	0	
1	0	0	1	$P \cdot \bar{Q} \cdot \bar{R}$
1	0	1	0	
1	1	0	0	
1	1	1	1	$P \cdot Q \cdot R$

Assim, uma expressão lógica que resultará na saída desejada será obtida pela junção das quatro expressões unidas por uma porta OR:

$$\text{Saída} = (\bar{P} \cdot \bar{Q} \cdot R) + (\bar{P} \cdot Q \cdot \bar{R}) + (P \cdot \bar{Q} \cdot \bar{R}) + (P \cdot Q \cdot R)$$

Como cada entrada, assim como sua respectiva negação, aparece duas vezes ao longo dos termos, podemos selecionar qualquer uma delas para realizar a simplificação que resultará no mesmo resultado. Arbitariamente, serão selecionados negação de P e P para serem isolados:

$$(\bar{P} \cdot \bar{Q} \cdot R) + (\bar{P} \cdot Q \cdot \bar{R}) = \bar{P} \cdot (\bar{Q} \cdot R + Q \cdot \bar{R}) = \bar{P} \cdot (Q \oplus R)$$

$$(P \cdot \bar{Q} \cdot \bar{R}) + (P \cdot Q \cdot R) = P \cdot (\bar{Q} \cdot \bar{R} + Q \cdot R) = P \cdot \overline{(Q \oplus R)}$$

Juntando novamente, teremos:

$$\text{Saída} = \bar{P} \cdot (Q \oplus R) + P \cdot \overline{(Q \oplus R)}$$

Fazendo P como um termo e  $(Q \oplus R)$  como outro, podemos observar que temos (negação do primeiro termo E o segundo termo) OU (primeiro termo E negação do segundo termo), o que é equivalente ao OU exclusivo. Assim, teremos como expressão final:

$$\text{Saída} = P \oplus Q \oplus R$$

A verificação dessa equivalência pode ser vista na tabela "Resumo dos operadores lógicos para três entradas". Assim, o circuito será uma única porta lógica XOR:



Figura 21

## 4.3 Interruptores como portas lógicas

Se pensarmos em uma entrada de um circuito lógico do ponto de vista de eletricidade, seus dois valores lógicos 0 e 1 podem ser representados como se a entrada fosse um interruptor: o valor lógico 0 seria o interruptor aberto, não permitindo a passagem de corrente elétrica, e o valor 1 seria o interruptor fechado, possibilitando a entrada de corrente. A figura a seguir ilustra esses dois estados dos interruptores usando a simbologia ANSI, que é a mesma adotada para as portas lógicas neste livro-texto.

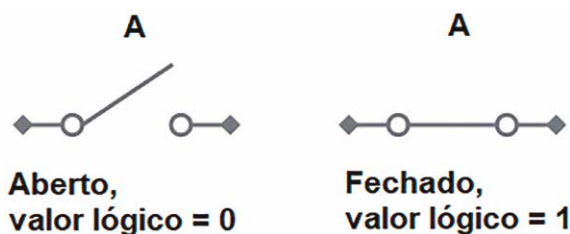


Figura 22 – Entrada de um circuito lógico representada como um interruptor elétrico

É possível representar assim as portas lógicas por meio da associação de interruptores. Uma porta AND seria uma associação em série de ambas as entradas, uma vez que se uma entrada for falsa, o circuito será interrompido (saída igual a 0).

Os operadores lógicos E e OU podem ser representados por meio de associações de chaves (ou interruptores) em um circuito análogo a um circuito elétrico, sendo que um interruptor fechado representa o valor lógico verdadeiro (1) para uma proposição e um interruptor aberto representa o valor falso (0) para a mesma proposição. Desta forma, temos:

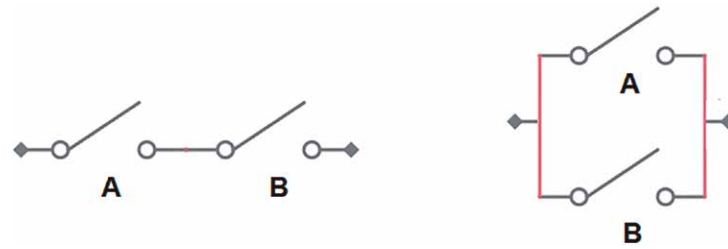


Figura 23 – Dois interruptores em série (à esquerda) representam uma porta lógica AND, e dois interruptores em paralelo (à direita) representam uma porta OR

As demais portas lógicas serão obtidas a partir dessas duas: a negação (porta NOT) será representada por um interruptor correspondente, e nas demais utilizaremos as regras de simplificação apresentadas anteriormente para reduzi-las a uma associação de NOT, AND e OR:

**Porta NAND:**  $\overline{(A \cdot B)} \equiv \bar{A} + \bar{B}$

**Porta NOR:**  $\overline{(A + B)} \equiv \bar{A} \cdot \bar{B}$

**Porta XOR:**  $A \oplus B \equiv (\bar{A} \cdot B) + (A \cdot \bar{B})$

**Porta NXOR:**  $\bar{A} \oplus \bar{B} \equiv (A \cdot B) + (\bar{A} \cdot \bar{B})$

Assim, as portas lógicas correspondentes ficariam:

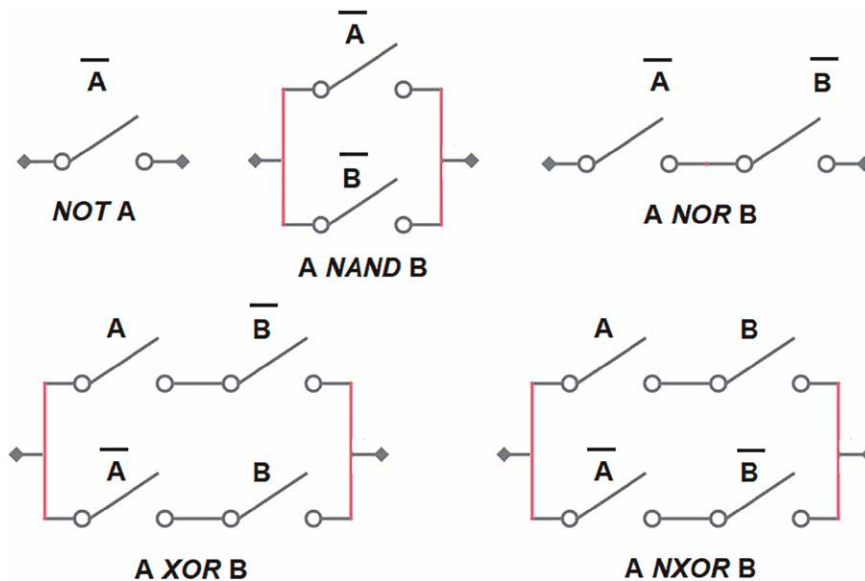


Figura 24 – Conexões entre dois interruptores A e B equivalentes às portas lógicas



## Observação

Na imagem de apresentação, para efeitos ilustrativos, todos os interruptores estão abertos, o que não ocorre em situação real, já que uma entrada e sua negação estarão em estados opostos.

Os transistores, que são a base da eletrônica digital, podem ser considerados como interruptores acionados eletricamente, assim como eram suas antecessoras, as válvulas. Desta forma, essa abordagem coloca as portas lógicas mais próximas de um circuito lógico digital real.

Exemplo: para as expressões lógicas a seguir, construir a associação de interruptores equivalentes e, a partir dela, obter as saídas para quando:

I)  $A = 0, B = 1$  e  $C = 0$

II)  $A = B = C = 1$

a)  $S = (A + B) \cdot (B \oplus C) \cdot \bar{C}$

Temos no caso específico três expressões,  $(A + B)$ ,  $(B \oplus C)$  e negação de  $C$  ligadas por uma porta AND de três entradas. Assim, as três expressões estarão ligadas em série:

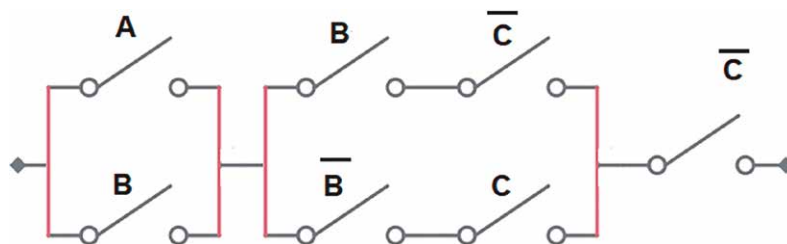


Figura 25

I)  $A = 0, B = 1$  e  $C = 0$

Neste caso, os interruptores  $A$  e  $C$  estarão abertos, e  $B$  fechado, com suas negações obviamente no estado oposto. Assim:

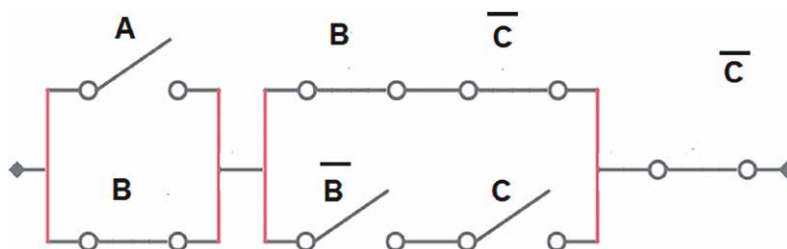


Figura 26

Assim, para esses valores lógicos, é possível observar que há um caminho sem interrupções ao longo do circuito. Portanto, o valor lógico dessa associação para as entradas dadas é **1**.

$$\text{II) } A = B = C = 1$$

Neste caso, os interruptores A, B e C estarão abertos, com suas negações obviamente no estado oposto. Assim:

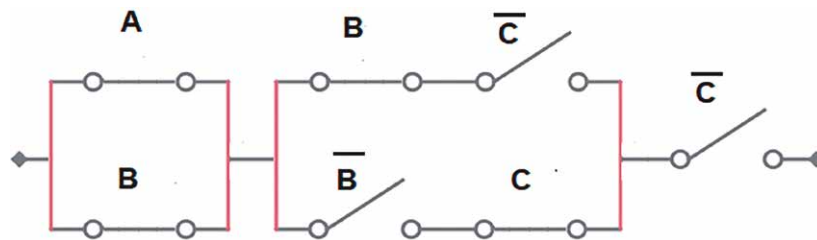


Figura 27

Para esses valores lógicos, é possível observar que não há um caminho possível sem interrupções ao longo do circuito. Assim, o valor lógico dessa associação para as entradas dadas é **0**.

$$\text{b) } S = (A \cdot B) + \overline{(A \oplus C)} + \overline{(B + C)}$$

Temos nesse caso três expressões,  $(A \cdot B)$ , negação de  $(A \oplus C)$  e negação de  $(B + C)$  ligadas por uma porta OR de três entradas. Portanto, as três expressões estarão ligadas em paralelo:

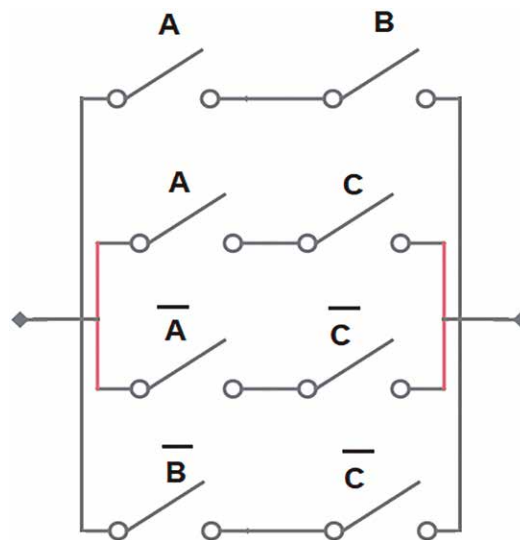


Figura 28

$$\text{I) } A = 0, B = 1 \text{ e } C = 0$$

Neste caso, os interruptores A e C estarão abertos, e B fechado, com suas negações obviamente no estado oposto. Assim:

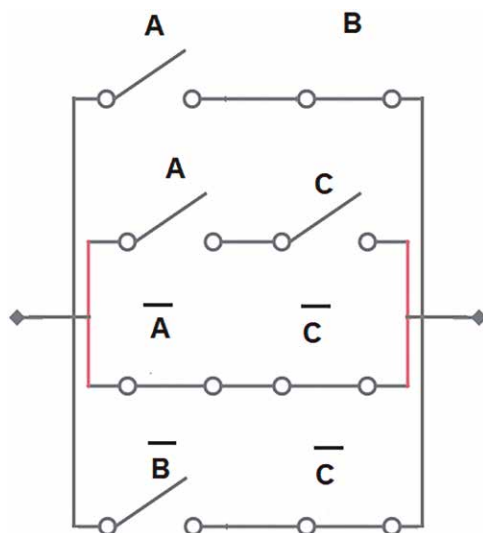


Figura 29

Logo, para esses valores lógicos, é possível observar que há um caminho sem interrupções ao longo do circuito, pela terceira linha, de cima para baixo. Assim, o valor lógico desta associação para as entradas dadas é **1**.

II)  $A = B = C = 1$

Como no item II do exemplo anterior, os interruptores A, B e C estarão abertos, com suas negações obviamente no estado oposto. Assim:

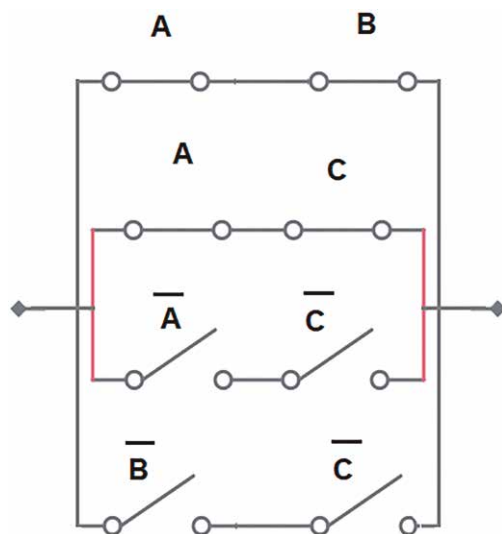


Figura 30

É possível observar que há mais de um caminho sem interrupções ao longo do circuito; sendo assim, o valor lógico dessa associação para as entradas dadas é **1**.



## Resumo

Nesta unidade, vimos que toda expressão lógica possui infinitas expressões equivalentes. Idealmente, um circuito lógico precisa ter, sempre que possível, a menor quantidade de portas lógicas, por uma economia de tamanho, custos e consumo de energia. As leis da lógica são relações de equivalência que permitem identificar as principais expressões e substituí-las, simplificando assim os circuitos lógicos.

Entendemos que uma situação bastante comum ocorre, no desenvolvimento de um circuito lógico, a partir da saída desejada do circuito para dela obter a expressão lógica que resulta nesta saída. Existe a possibilidade de obtermos uma expressão lógica para um circuito a partir de sua tabela verdade, mas essa forma de obtenção não é eficiente, resultando em circuitos muito grandes, com ampla quantidade de portas lógicas, e que necessitam de muita simplificação.

Observamos que é possível estabelecer uma relação entre portas lógicas e associação de interruptores, nas quais cada entrada do circuito seria um interruptor. Como transistores, que são a base da eletrônica digital, podem ser considerados como interruptores acionados eletricamente, aproximando assim as portas lógicas de circuitos com componentes reais.

Na sequência vimos que o sistema decimal é uma convenção utilizada para representar valores numéricos. Como outros sistemas de numeração utilizados atualmente, os valores são representados através de uma forma resumida de somatória de potências. A base delas é chamada de base numérica.

Além do sistema decimal que utilizamos cotidianamente, na ciência da computação são utilizados outros sistemas numéricos: binário (base 2), octal (base 8) e hexadecimal (base 16). Todos eles funcionam segundo o mesmo princípio, no qual o número de algarismos presentes na notação é igual à base do sistema.

A fim de realizar a conversão de outro sistema numérico para o sistema decimal, decompomos o número na somatória de potências correspondente e resolvemos as potências e produtos, somando-os. Já o uso do sistema binário faz com que os prefixos que indicam ordens de grandeza tenham valores diferentes do padrão adotado no sistema internacional de medidas (SI). Assim, todos eles serão ligeiramente maiores, em valores absolutos, quando utilizados na ciência da computação.

Aprendemos que não apenas números inteiros, mas também números reais (com parte fracionária) podem ser convertidos para outros sistemas numéricos. Porém, caso o valor esteja sendo convertido de uma base numérica para outra que não possua divisores em comum, podem ocorrer dízimas periódicas, que irão gerar um erro (imprecisão) no valor obtido.

Foi mostrado que, ao ser aplicado computacionalmente, o sistema binário convencional, na sua forma matemática, pode gerar alguns problemas durante a realização de certas operações aritméticas, devido ao armazenamento da informação em *bits* e *bytes*. Para resolver este problema, é adotada a notação denominada Complemento de 2 (Comp-2) para armazenar números inteiros. Essa forma, no entanto, é dependente da quantidade de *bits* utilizados no armazenamento dos valores, sendo que números negativos serão armazenados diferentemente em quantidades de *bits* diversas.

Entendemos que a lógica matemática se baseia em proposições, que são afirmações às quais se atribui um valor lógico verdadeiro ou falso, e em operadores, que conectam as proposições entre si. A combinação de proposições por meio de operadores forma as expressões lógicas, que possuem um valor lógico único para cada possível combinação de valores lógicos das proposições. Cada expressão pode representar um circuito lógico.

Por fim, vimos que portas lógicas são a representação dos operadores lógicos e podem ser combinadas entre si para construir os circuitos. Elas se dividem em sete: NOT (negação), AND (E), OR (OU), XOR (OU exclusivo), NAND (negação de E), NOR (negação de OU) e NXOR (negação de OU exclusivo). A porta NOT possui apenas uma entrada, enquanto as demais têm pelo menos duas entradas. Todas elas possuem apenas uma saída.

Circuitos lógicos construídos por meio de portas lógicas possuem apenas uma saída, e sua configuração e número de portas lógicas utilizados dependem da expressão lógica que os representa.





## Exercícios

**Questão 1.** Leia o texto a seguir, a respeito dos sistemas numéricos utilizados na ciência da computação.

O sistema numérico decimal, que utilizamos no nosso cotidiano, pode ser utilizado para modelar sistemas digitais. Porém, não é prático utilizar 10 símbolos para um universo em que, a cada elemento, existem apenas duas possibilidades: ligado ou desligado. Outros sistemas podem ser mais eficientes para esse caso. Na eletrônica digital, são três os sistemas numéricos de maior importância: binário, octal e hexadecimal. O primeiro é a base para todo o sistema digital, enquanto os demais têm finalidade de expressar grupos de dígitos binários (*bits*) por cada algarismo.

O sistema **binário** possui dois símbolos capazes de representar grandezas (0 e 1). O sistema **octal** possui oito símbolos (0, 1, 2, 3, 4, 5, 6, 7), sendo sua base um expoente inteiro da base do sistema binário – um algarismo octal corresponde a um grupo de 3 *bits*. O sistema **hexadecimal** também possui uma potência de 2 como base, mas cada símbolo hexadecimal representa 4 *bits*. Esse sistema possui 16 símbolos, sendo os 10 primeiros os mesmos utilizados pelo sistema decimal, e os demais possuem os seguintes valores equivalentes em decimal: A=10, B=11, C=12, D=13, E=14 e F=15.

LENZ, M. L.; MORAES, M. L. *Eletrônica digital* [recurso eletrônico]. Porto Alegre: Sagah, 2019. Adaptado.

Considerando esse contexto, avalie as afirmativas a seguir e a relação proposta entre elas.

I – No numeral  $(4325)_8$ , o algarismo 2 vale  $(16)_{10}$ .

Porque

II – O sistema hexadecimal recorre a 16 símbolos para a representação de valores que, quando combinados, podem expressar qualquer valor representável pelo sistema decimal.

A respeito dessas afirmativas, assinale a opção correta.

- A) As afirmativas I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- B) As afirmativas I e II são proposições verdadeiras, e a II não é uma justificativa correta da I.
- C) A afirmativa I é uma proposição verdadeira, e a II é uma proposição falsa.
- D) A afirmativa I é uma proposição falsa, e a II é uma proposição verdadeira.
- E) As afirmativas I e II são proposições falsas.

Resposta correta: alternativa B.

### Análise da questão

Em  $(4325)_8$ , temos um numeral composto de quatro algarismos e expresso em sistema octal. Nossa tarefa é determinar se o algarismo 2, que é o 3º algarismo (da esquerda para a direita), representa um valor equivalente a 16 unidades do sistema decimal. O sistema octal, assim como todos os outros estudados, é um sistema posicional. Isso quer dizer que a posição de cada algarismo é importante para determinar quanto, de fato, ele vale naquele contexto. No nosso sistema decimal, lemos o numeral 23 como "vinte e três" e o numeral 32 como "trinta e dois". Note que utilizamos os mesmos símbolos (2 e 3), mas a ordem de aparecimento deles faz com que o 2º numeral represente um valor maior. Quanto mais à esquerda for seu aparecimento, mais significativo será o algarismo dentro daquele numeral.

Pois bem, vamos voltar ao nosso objeto de estudo:  $(4325)_8$ . Para expressar esse valor em sistema decimal, levamos em consideração que:

- partimos de sistema cuja base é 8;
- cada expoente deve ser determinado pela posição do algarismo;
- o valor unitário do símbolo do algarismo deve multiplicar a potência.

Dessa forma, temos a seguinte composição:

$$\begin{array}{cccc} 4 & 3 & 2 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 4 \times 8^3 & 3 \times 8^2 & 2 \times 8^1 & 5 \times 8^0 \end{array}$$

Como estamos interessados no algarismo 2, vamos determinar o quanto ele vale, dentro deste numeral, quando expresso em sistema decimal. Temos:

$$2 \times 8^1 = 2 \times 8 = 16$$

Logo, sabemos que a afirmativa I é verdadeira. Quanto à afirmativa II, precisamos apenas determinar se o que é afirmado ali é correto. No sistema hexadecimal, temos 16 símbolos para representação de valores que, quando combinados, podem, sim, significar qualquer valor numérico, assim como acontece no sistema decimal.

Temos, portanto, que as afirmativas I e II são ambas verdadeiras. Porém, apesar de tratarem do mesmo contexto, II não justifica I. Ou seja, o algarismo 2 em  $(4325)_8$  vale  $(16)_{10}$ , mas não porque o sistema hexadecimal dispõe de 16 símbolos.

**Questão 2.** (Enade 2017) Os sistemas de refrigeração de piscinas de combustível em usinas nucleares evitam que a temperatura desses tanques exceda o limite de segurança. O circuito representado na figura a seguir atende aos requisitos necessários para o controle da ativação do sistema de resfriamento quando a temperatura está próxima de seu ponto crítico.

O diagrama de tempo ilustrado na figura apresenta uma amostra das temperaturas lidas desde o momento  $t_1$  ao  $t_8$ . Os sinais de entrada  $T_a$ ,  $T_b$  e  $T_c$  são de termômetros que medem a temperatura da piscina em diferentes pontos ao longo do dia e  $S$  é o terminal de acionamento do sistema.

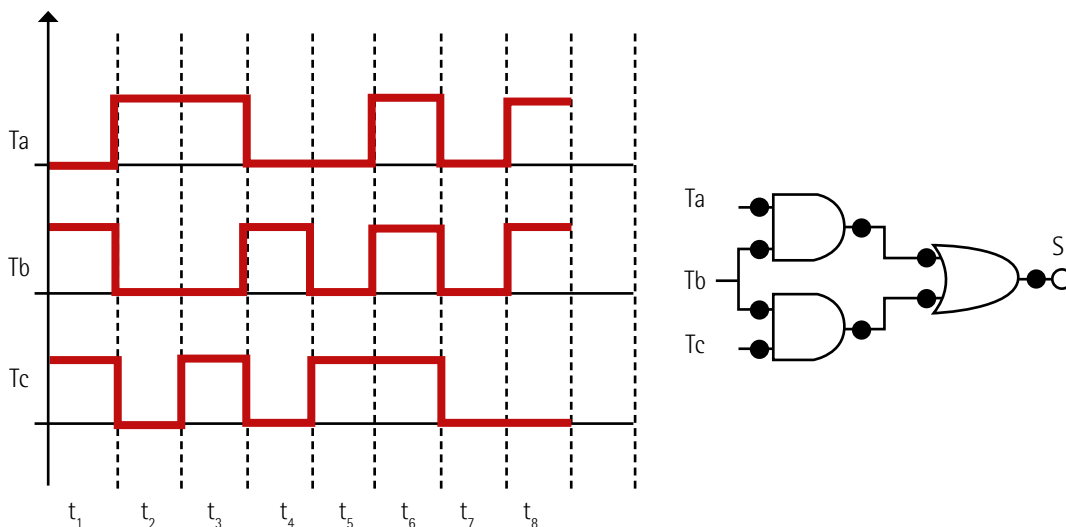


Figura 31

Nesse contexto, assinale a opção em que são apresentados os momentos nos quais o sistema foi acionado.

- A)  $t_1$ ,  $t_4$  e  $t_8$ .
- B)  $t_1$ ,  $t_6$  e  $t_8$ .
- C)  $t_2$ ,  $t_4$  e  $t_6$ .
- D)  $t_2$ ,  $t_6$  e  $t_8$ .
- E)  $t_3$ ,  $t_5$  e  $t_7$ .

Resposta correta: alternativa B.

## Análise da questão

Para começar a analisar o sistema de refrigeração, vamos entender primeiramente o funcionamento do circuito lógico apresentado. Temos um circuito contendo duas portas AND e uma porta OR. Nele,  $T_a$ ,  $T_b$  e  $T_c$  representam as entradas e  $S$  representa a saída, conforme mostrado a seguir.

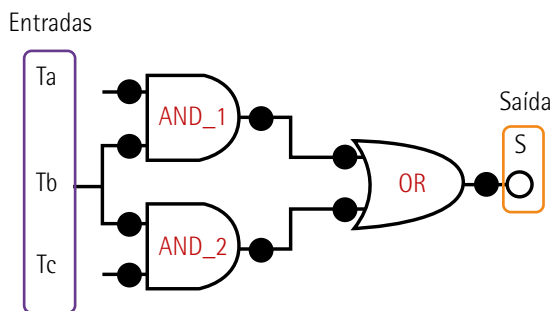


Figura 32

Podemos extrair a expressão lógica que corresponde ao circuito em estudo. Para isso, partimos das entradas e vamos em direção à saída. No meio do caminho, escrevemos as expressões correspondentes às saídas de cada uma das portas lógicas. Dessa forma, temos o que se mostra a seguir.

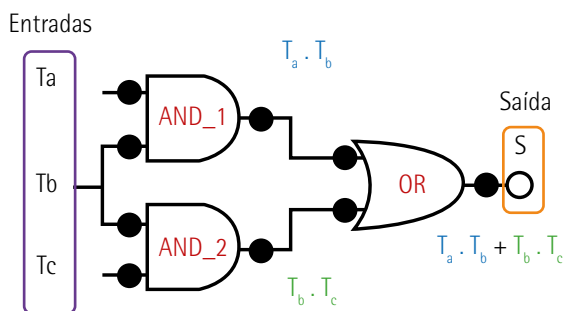


Figura 33

Caso você tenha sentido dificuldade nessa etapa, observe individualmente as portas lógicas do circuito.

- **AND\_1:** recebe como entradas o sinal  $T_a$  e o sinal  $T_b$ . Como a porta executa a função AND (E), ela entrega para o circuito a saída  $T_a \cdot T_b$ .
- **AND\_2:** recebe como entradas o sinal  $T_b$  e o sinal  $T_c$ . Como a porta executa a função AND (E), ela entrega para o circuito a saída  $T_b \cdot T_c$ .
- **OR:** recebe como entradas os sinais oriundos das saídas das portas AND. De AND\_1, recebe o sinal  $T_a \cdot T_b$ . De AND\_2, recebe o sinal  $T_b \cdot T_c$ . Como a porta executa a função OR (OU), ela entrega para o circuito a saída  $T_a \cdot T_b + T_b \cdot T_c$ , que já corresponde à saída  $S$ .

Portanto, podemos escrever a expressão que caracteriza o circuito como:

$$S = T_a \cdot T_b + T_b \cdot T_c$$

Nessa situação, esperamos que o sistema de refrigeração das piscinas de combustível da usina seja acionado nos instantes em que  $T_a$  e  $T_b$  estiverem em nível lógico 1, **ou** nos instantes em que  $T_b$  e  $T_c$  estiverem em nível lógico 1. Note que apenas descrevemos aquilo que a expressão lógica nos disse por meio de seus operadores. Em outras palavras, podemos pensar que, sempre que  $T_b$  e pelo menos um dos outros sensores ( $T_a$  ou  $T_c$ ) estiverem em nível lógico 1, o sistema de refrigeração será acionado.

Caso você se sinta confortável apenas com a análise da expressão, construa e analise a tabela verdade dela. Para isso, seguiremos a seguinte ordem de resolução:

- $T_a \cdot T_b$
- $T_b \cdot T_c$
- $T_a \cdot T_b + T_b \cdot T_c$

Agora, podemos construir a tabela. Optamos por utilizar uma coluna para cada operação lógica. É possível pular etapas, realizar mais de uma operação na mesma coluna, caso se sinta confortável. O objetivo é que tenhamos colunas para as entradas e para a saída. As colunas intermediárias (neste caso,  $T_a \cdot T_b$  e  $T_b \cdot T_c$ ) apenas nos ajudam a atingir nosso objetivo.

**Tabela 35**

$T_a$	$T_b$	$T_c$	$T_a \cdot T_b$	$T_b \cdot T_c$	$S = T_a \cdot T_b + T_b \cdot T_c$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1

Note que a saída  $S$  apenas resulta em 1 quando  $T_b = 1$  e qualquer outro sensor também estiver em nível 1, conforme esperado pela análise da expressão.

Vamos, então, analisar o diagrama de tempo do enunciado, que nos diz o nível lógico de cada sensor para 8 instantes. Acompanhando a linha que representa o sinal de cada sensor ao longo do tempo, podemos dizer se ele estava no estado 0 (quando sua linha estiver baixa) ou no estado 1 (quando sua linha estiver alta) em determinado instante. Temos, então, o que se mostra a seguir.

