SISTEMAS DIGITAIS

UNIDADE 1 – SISTEMAS DE NUMERAÇÃO

Introdução

Olá, caro estudante! Seja bem-vindo à unidade 1 de **Sistemas Digitais**.
Aqui, abordaremos como são construídos os sistemas digitais, porém, para isso, vamos passar por alguns pontos conceituais antes. Um desses pontos relaciona-se aos sistemas de numeração. Mas por que devemos passar pelos sistemas de numeração? Bem, lembre-se de que os computadores manipulam somente números; números para representar as instruções e números para representar as informações que serão processadas. Temos, então, que abstrair a forma com que as informações numéricas são manipuladas.
Avançando nossa abstração um pouco mais, para que possamos entender a manipulação, devemos saber como os valores são representados, ou seja, o que são os sistemas de numeração. Será que existem apenas os sistemas decimal e binário? Saiba, desde já, que não: existem vários outros sistemas de numeração, mas os sistemas decimal e binário são, digamos, os principais.
Mas se o computador manipula apenas os sistemas binários, porque existem outros? Os sistemas de numeração servem também para a nossa abstração e para facilitar a manipulação numérica em algumas situações, as quais demandam representar os números de forma particular. Se o computador manipula informações numéricas, existe alguma álgebra por trás disso? Sim, a manipulação de valores binários é regida por uma álgebra denominada "álgebra booleana".
Assim, para conversarmos sobre sistemas numéricos e álgebra booleana, esta unidade abordará, inicialmente, os sistemas de numeração e a conversão entre eles. Veremos, ainda, algumas variações do sistema binário de representação. Depois de averiguarmos os sistemas numéricos, enveredaremos pela álgebra booleana, suas simbologias e teoremas.
Preparado para iniciar o caminho, abordando os sistemas de numeração? Então, vamos lá!

1.1 Conversão entre sistemas de numeração

Antes de iniciarmos nossa conversa sobre conversão entre sistemas de numeração, convém falarmos sobre o que vem a ser um sistema de numeração e quais são os principais, do ponto de vista computacional. De acordo com (TOCCI; WIDMER; MOSS, 2018), os sistemas numéricos mais comuns, sob o ponto de vista computacional, são: decimal; hexadecimal; octal; e binário. E o que eles têm em comum? É o que você saberá a seguir, com a leitura deste tópico.

1.1.1 Sistemas de numeração

Sistemas de numeração representam como um valor numérico é formalizado, ou seja, representa as regras para a representação de um número. Assim, retomando: o que os sistemas decimal, hexadecimal, octal e binário têm em comum é que todos eles podem ser fatorados. Por exemplo, caso tenhamos um número "Q" de quatro dígitos representado por meio de uma base hipotética qualquer denominada B, esse número poderia ser decomposto nos seguintes fatores:

$$\mathbf{Q_3 * B^3 + Q_2 * B^2 + Q_1 * B^1 + Q_0 * B^0}$$

Sendo Q_3 o dígito mais significativo e, consequentemente, Q_0 o dígito menos significativo. Para ficar mais claro, vamos supor um número de quatro dígitos na forma decimal de representação:

$$3546 = \mathbf{3 * 10^3 + 5 * 10^2 + 4 * 10^1 + 6 * 10^0}$$

Citando o exemplo acima de decomposição de um número, acabamos de mencionar a "**base decimal**". Tenha em mente que a base decimal é a base que estamos acostumados a manipular em nosso cotidiano. Iniciando a nossa conversa pela base decimal, fica mais fácil abstrair dois conceitos: os símbolos admissíveis e sua quantidade.

Quantidade de símbolos admissíveis	A quantidade de símbolos que podemos usar em cada dígito é derivado do próprio nome da base. Na base decimal, por exemplo, podemos representar um dígito com um dos 10 símbolos (ou valores) possíveis
Símbolos admissíveis	0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Os computadores não são eficientes para manipular valores na base decimal em função de sua forma de implementação, que é baseada em sistemas lógicos digitais. Em função de sua estrutura eletrônica, em que se admite apenas dois valores possíveis de seus sinais internos, utiliza-se a "**base binária**".
E o que vem a ser sinais internos? Chamamos de "sinais internos", no caso dos circuitos eletrônicos, os valores que trafegam nos fios e componentes eletrônicos que compõem o computador. Esses valores são vinculados às voltagens aplicadas sobre uma determinada parte do circuito. Geralmente, podemos associar a presença de tensão elétrica positiva como "nível lógico 1", e a ausência de tensão elétrica como "nível lógico 0". A figura a seguir ilustra esses dois níveis. Observe!

Figura 1 - Analogia dos níveis lógicos "0" e "1" com a tensão inserida em uma lâmpada. Lâmpada acesa denota que está recebendo o nível "1", ao passo que se estiver apagada, o nível lógico "0".
Fonte: Elaborada pelo autor, 2019.

Na figura, a lâmpada representa um dígito de um valor binário — o chamado "bit" (***B**inary dig**IT***, ou dígito binário). Em tal situação, a lâmpada estará representando o nível "1" quando estiver acesa ou o nível "0" quando apagada. Assim, temos (clique para ler):

quantidade de símbolos admissíveis: 2.
símbolos admissíveis: 0, 1.

Podemos falar que um número binário é formado por vários bits. O conjunto de bits, chamamos de "palavra". Por exemplo, uma palavra de 8 bits tem o tamanho (ou largura) de 1 byte. Mas, então, podemos realizar a decomposição sobre um valor escrito em binário? A base binária não foge à regra de decomposição em fatores. Por exemplo, supondo o valor binário $1101_{(2)}$, temos:

$$1101_{(2)} = \mathbf{1} * \mathbf{2}^3 + \mathbf{1} * \mathbf{2}^2 + \mathbf{0} * \mathbf{2}^1 + \mathbf{1} * \mathbf{2}^0$$

Entendemos o que é um sistema de numeração, mas como realizar a conversão entre valores representados por bases numéricas distintas? Isso é o que veremos a seguir.

1.1.2 Conversão entre bases numéricas

Acabamos de falar sobre a decomposição de um valor na base binária. Retomando o exemplo:

$1101_{(2)} = \mathbf{1} * \mathbf{2}^3 + \mathbf{1} * \mathbf{2}^2 + \mathbf{0} * \mathbf{2}^1 + \mathbf{1} * \mathbf{2}^0$

O que podemos abstrair dessa decomposição? Inicialmente, podemos usar os valores de $\mathbf{2^n}$ para definir o nome dessa representação binária: BCD8421. O nome "BCD" significa "decimal codificado em binário" (*binary-coded decimal*). O valor 8421 é obtido pelas exponenciações ($\mathbf{2}^3 = 8$; $\mathbf{2}^2 = 4$; $\mathbf{2}^1 = 2$; $\mathbf{2}^0 = 1$).

VOCÊ QUER LER?

Existem outras formas de representação numérica usando sistema binário. Para se adequar a outras aplicações ou circunstâncias, um valor numérico pode ser representado de formas distintas por meio de outros sistemas de representação binários. Para saber mais sobre o assunto, acesse o link: https://wiki.sj.ifsc.edu.br/wiki/index.php/DIG222802_AULA07.

Voltando à decomposição acima, podemos realizar outra abstração: qual é o resultado obtido se realizarmos a soma "$\mathbf{1} * \mathbf{2}^3 + \mathbf{1} * \mathbf{2}^2 + \mathbf{0} * \mathbf{2}^1 + \mathbf{1} * \mathbf{2}^0$"? O resultado da expressão é 13. Então, podemos falar que:
$1101_{(2)} = 13_{(10)}$

Esse processo pode ser aplicado para qualquer base de numeração quando desejamos transformar o número expresso em base decimal. Para demonstrar isso, vamos falar sobre dois outros sistemas de numeração: o sistema hexadecimal e o sistema octal. O sistema hexadecimal é muito utilizado pelos programadores e projetistas de sistemas computacionais por representar um número de forma mais condensada em relação ao sistema decimal ou ao sistema binário de representação numérica.

Quantidade de símbolos admissíveis: 16.
Símbolos admissíveis: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Observando os símbolos admissíveis, você notará a presença das letras "A" a "F". Nesse caso, se fizermos uma correspondência direta com o sistema decimal, temos a faixa de "A", representando o valor 10, até a letra "F", denotando o valor "15". A conversão de hexadecimal para decimal segue o mesmo processo de decomposição. Vamos supor o valor: $3A8C_{(16)}$:

$3A8C_{(16)} = \mathbf{3} * \mathbf{16}^3 + \mathbf{10} * \mathbf{16}^2 + \mathbf{8} * \mathbf{16}^1 + \mathbf{12} * \mathbf{16}^0 = 12288 + 2560 + 128 + 12 = 14988_{(10)}$

Por fim, vamos demonstrar que esse processo de decomposição também serve para realizar a conversão de um valor representado na base octal para binário. O sistema octal foi muito usado pelos programadores e projetistas de sistemas computacionais quando a capacidade dos equipamentos era extremamente menor em relação aos recursos computacionais atualmente disponíveis. Por esse motivo, grave bem, o sistema octal cedeu espaço para o sistema hexadecimal. Em relação às suas características básicas, temos:

- quantidade de símbolos admissíveis: 8.
- símbolos admissíveis: 0, 1, 2, 3, 4, 5, 6, 7.

Para proceder à conversão, vamos supor o valor $5461_{(8)}$:

$5461_{(8)} = \mathbf{5} * \mathbf{8}^3 + \mathbf{4} * \mathbf{8}^2 + \mathbf{6} * \mathbf{8}^1 + \mathbf{1} * \mathbf{8}^0 = 2560 + 256 + 48 + 1 = 2865_{(10)}$

Até o momento, falamos sobre o processo de conversão da base "Q" para a base decimal. Mas como realizar o processo contrário? Como converter um valor expresso na base decimal para a base Q? Basta realizar divisões sucessivas do valor decimal pela base em questão.

A figura a seguir ilustra o processo de conversão dos valores acima calculados para as suas bases de origem. Confira!

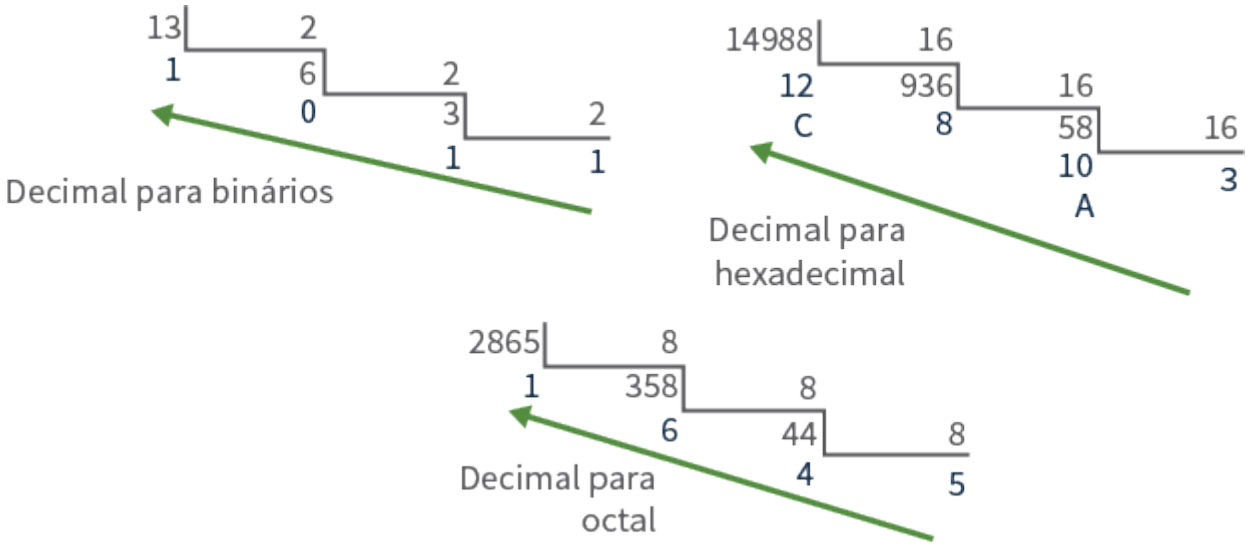


Figura 2 - Processo de divisão de números representados na base decimal para as bases binária, hexadecimal e octal, utilizando-se divisões sucessivas.
Fonte: Elaborada pelo autor, 2019.

Na figura anterior, podemos notar que o valor resultante da conversão é construído coletando-se o último quociente e todos os restos da divisão na ordem inversa de aparição.

Mencionamos que as bases octal e hexadecimal são usadas principalmente por programadores e projetistas de sistemas computacionais, certo? Como elas são mais próximas do sistema binário, será que existe uma forma mais fácil para realizar a conversão entre binário para hexadecimal, octal e hexadecimal ou de octal para binário? Para respondermos essa questão, vamos analisar a quantidade de bits necessária para representarmos um valor em hexadecimal ou octal:

Hexadecimal	Quantidade de símbolos admissíveis: 16 Quantidade de bits necessários para cada dígito: log2(16) = 4
Octal	Quantidade de símbolos admissíveis: 8 Quantidade de bits necessários para cada dígito: log2(8) = 3

Assim, cada dígito hexadecimal gerará uma sequência de quatro bits, e cada dígito octal será convertido em três bits. A figura a seguir exemplifica o processo de conversão entre essas bases de representação numérica, tomando como exemplo o valor binário de 16 bits: "1011011010101101".

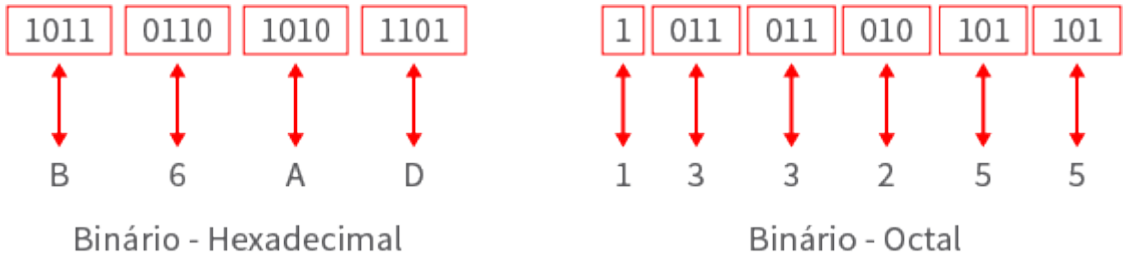


Figura 3 - Conversão entre a base binária e as bases hexadecimal e octal. O processo de agrupamento de 4 ou 3 bits é realizado nos dois sentidos da conversão.
Fonte: Elaborada pelo autor, 2019.

Na figura, perceba que há relação entre os agrupamentos de quatro ou três bits para cada dígito hexadecimal ou octal, respectivamente. O valor binário é representado de acordo com o padrão BCD8421. Mas o computador manipula números binários representando valores positivos e negativos. Como ele consegue diferenciar essas faixas de valores? Para responder essa questão, chegou a hora de mencionarmos que dentro da representação binária um sistema conhecido como "complemento 2" merece destaque.
Para uma melhor abstração, vamos começar a falar sobre complemento 2 exemplificando com o tipo "**char**" da linguagem "C" de programação. Esse tipo comumente é representado por uma palavra de 8 bits (1 byte). Assim, temos 2^8 valores possíveis, ou seja, 256 valores que podem ser representados em uma variável do tipo "**char**".
Mas quais são esses valores? Por que essa faixa vai de -127 a 128? Porque que se utilizássemos na criação da variável o modificador de tipo "**unsigned**" (ficando "**unsigned char**"), a faixa iria de 0 a 255? Tudo isso ocorre devido ao fato de que o bit mais significativo (MSB – *Most Significant Bit*), nos casos em que não há a utilização do "**unsigned**", representa o sinal.
Dessa forma, dos oito bits do "**char**", temos um bit de sinal e sete contendo a informação propriamente dita. Consequentemente, 2^7 denota que temos 128 valores possíveis: 128 valores negativos (de -128 a -1) e 128 valores positivos (de 0 a 127).
E como representar um valor negativo? Bem, vamos tomar como exemplo prático, para facilitar a apresentação e os cálculos, um valor de quatro bits "$\mathbf{1110_{(2)}}$". De acordo com o que foi apresentado, temos:

- Bit MSB (bit sinal): "**1**" → indicando que o valor é negativo. Caso o MSB fosse 0, o número representado seria positivo.
- Valor propriamente dito: "$\mathbf{110_{(2)}}$". Caso o bit sinal fosse 0 (indicando um número positivo), poderíamos aplicar o BCD8421 diretamente sobre "$\mathbf{110_{(2)}}$". Porém, como o bit sinal vale "**1**", não podemos falar que o valor representado é "$\mathbf{-6_{(10)}}$".

Nesse caso, com o bit sinal indicando um valor negativo, para se conhecer o valor, temos que aplicar dois passos:

- Passo 1: inverte-se todos os bits do valor representado: "$\mathbf{1110_{(2)}}$" → "$\mathbf{0001_{(2)}}$".
- Passo 2: soma-se ao valor obtido da inversão dos bits: "$\mathbf{0001_{(2)}}$" + "$\mathbf{0001_{(2)}}$" = "$\mathbf{0010_{(2)}}$" (1 + 1 = 2).

Assim, entendemos que o valor "$\mathbf{1110_{(2)}}$" equivale a "$\mathbf{-2_{(10)}}$".
Para achar o correspondente negativo a partir de um valor positivo ou para achar o correspondente positivo a partir de um valor negativo, devemos seguir os mesmos dois passos apresentados. Porém, tenha em mente que o computador não precisa realizar tais operações. O valor em complemento 2 já é diretamente obtido em suas operações de subtração, por exemplo.
Como mencionamos, um computador manipula palavras expressas no sistema binário de representação numérica. Assim, como temos uma álgebra matemática para manipularmos o nosso sistema decimal, deve haver uma álgebra adequada para a manipulação de valores binários. A seguir, conversaremos sobre a **álgebra booleana**.

VAMOS PRATICAR?

Agora é a sua vez de converter os valores a seguir. Os números sobrescritos entre parênteses representam as bases numéricas:

a) $1011_{(2)} = ?_{(10)}$
b) $1011\ 1110\ 1010\ 1111_{(2)} = ?_{(16)}$
c) $1C7_{(16)} = ?_{(10)}$
d) $3B4E_{(16)} = ?_{(2)}$
e) $3561_{(8)} = ?_{(16)}$

1.2 Álgebra booleana: simbologia e teoremas

Conhecer a álgebra booleana não somente é importante para abstrair o funcionamento dos sistemas computacionais, mas também para quando estivermos modelando os sistemas digitais. A princípio, um sistema digital pode ser representado por meio de uma expressão booleana, além de podermos utilizar outros elementos que serão vistos oportunamente.

1.2.1 Variáveis e expressões booleanas

De acordo com Vahid e Laschuk (2008), uma álgebra objetiva a representação de valores por meio de letras e símbolos. Mas o que são tais letras e símbolos? Para facilitar nossa abstração, vamos supor um exemplo prático: imagine que a iluminação de um cômodo de uma casa tenha acendimento automático. Isso se dá de acordo com a seguinte condição:

Se (alguém_presente_no_cômodo E já_anoiteceu) então luz = acesa
Senão luz = apagada

No nosso trecho apresentado de pseudocódigo, podemos identificar variáveis de entrada e saída.

Variáveis de entrada	As variáveis de entrada poderão, nesse caso, ser relacionadas a um sensor de presença (P) e um sensor de luminosidade (L). Ambos os sensores entregarão, ao sistema digital, valores binários.
Variável de saída	Essa variável é associada a uma saída (S) conectada a uma lâmpada, que será acesa ou permanecerá apagada.

Note que todos os itens envolvidos manipulam valores binários. Então, poderemos reescrever o pseudocódigo usando uma expressão booleana:

S = P "AND" L

Dessa forma, podemos identificar os primeiros elementos da álgebra booleana:

Expressão booleana: **S = P "AND" L**

- Variáveis booleanas: **S ; P ; L**
- Operador lógico: **AND**

VOCÊ O CONHECE?

Sistemas lógicos digitais, e a computação como um todo, são baseados na manipulação de valores binários. Valores binários remetem à lógica booleana e seria impossível falar de álgebra booleana sem falar sobre George Boole, o idealizador da álgebra booleana. Para saber um pouco sobre a vida dele, você poderá ler a dissertação de Sousa (2005), disponível em: https://repositorio.ufrn.br/jspui/bitstream/123456789/14224/1/GiselleCS.pdf.

Mencionamos, há pouco, o termo "**operador lógico**". Mas o que vem a ser um operador lógico? Conversaremos sobre isso a seguir.

1.2.2 Operadores lógicos

O que vem a ser operador lógico? Um operador lógico realiza a conexão entre duas variáveis para que seja estabelecido um resultado. Os operadores lógicos básicos são: NOT, AND, OR, e, em sistemas digitais, são representados pelas "**portas lógicas**". Assim, podemos falar que as portas lógicas são os elementos de abstração de mais baixo nível em sistemas digitais.

CASO

Um desenvolvedor de sistemas computacionais recebeu um projeto para automatizar a irrigação de uma pequena área. Tal irrigação levava em conta apenas a umidade do solo, umidade medida em quatro pontos por intermédio de sensores. O primeiro pensamento que lhe veio à mente consistiu em implementar a solução em microcontroladores (tais como PIC ou Arduino). Logo em seguida, ele refletiu: o projeto é extremamente simples e, caso fosse utilizado algum microcontrolador, este ficaria subaproveitado. Além disso, os componentes utilizados para atuar na abertura da água e a coleta do nível de umidade seriam os mesmos caso implementasse uma solução baseada em microcontroladores, ou baseada simplesmente na eletrônica digital. Dessa forma, ele resolveu, então, desenvolver a solução baseada na eletrônica digital. Para tanto, desenvolveu seu sistema lógico digital a partir das expressões booleanas que mapeavam o comportamento desejado para o sistema, de modo que fossem atendidas as funcionalidades da irrigação.

Antes de falarmos sobre as propriedades e teoremas da álgebra booleana, vamos conversar sobre as **portas lógicas**. Convém mencionarmos que toda porta lógica estará associada a uma representação em diagrama esquemático (a notação utilizada no desenho do circuito), uma simbologia na álgebra booleana (para representar a porta nas expressões booleanas) e uma tabela-verdade (que contempla todas as combinações possíveis dos valores que poderão ser assumidos pelas variáveis de entrada). Assim, para antecipar, podemos, também, representar o exemplo dado anteriormente (do acendimento automático da luz) por meio da expressão booleana, diagrama esquemático e tabela-verdade. Tais elementos são contemplados na figura a seguir. Observe!

	P	L	S = P . L
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Figura 4 - Tabela-verdade, expressão booleana e diagrama esquemático do sistema lógico digital para o acendimento automático da luz.
Fonte: Elaborada pelo autor, 2019.

Na figura, perceba, temos a tabela-verdade, a expressão booleana e o diagrama esquemático do sistema lógico digital para o acendimento automático da luz em função das variáveis "P" (presença de alguma pessoa no recinto) e "L" (intensidade da luminosidade do dia). Para que a luz se acenda, deve haver alguma pessoa no recinto "E" e a luminosidade externa ao recinto deve estar baixa.
Note, ainda, que a tabela-verdade (à esquerda da figura) representa todas as combinações possíveis das variáveis de entrada — combinação 0 até combinação 3. O número de linhas da tabela-verdade é representado por 2^n, em que "*n*" é o número de variáveis de entrada envolvidas. Os valores inseridos na tabela representam o próprio BCD8421.

VOCÊ QUER VER?

Podemos dizer que a base para a implementação de sistemas lógicos digitais é a construção da tabela-verdade. A tabela-verdade reflete os resultados da expressão booleana, levando-se em conta todas as combinações possíveis das variáveis de entrada. Para saber um pouco mais sobre como preencher uma tabela-verdade, assista ao vídeo: https://www.youtube.com/watch?v=mVJXZit3msA.

Mas o que realmente representam tais símbolos? Descubra a seguir.

Porta "NOT"

A porta NOT realiza a inversão (ou complemento) da variável apresentada à sua entrada. Trata-se, portanto, de uma porta unária, ou seja, é representada por uma função que envolve apenas uma variável de entrada.

- Simbologias: $S = \sim A$; $S = \bar{A}$; $S = A$; S

A figura a seguir ilustra o diagrama esquemático e a tabela-verdade referente à porta NOT.

	A	S = ~A
0	0	1
1	1	0

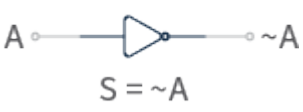


Figura 5 - Diagrama esquemático e tabela-verdade de uma porta NOT.
Fonte: Elaborada pelo autor, 2019.

Saiba que o diagrama esquemático da porta NOT também pode ser denotado apenas com a circunferência quando associado a alguma outra estrutura ou porta lógica.

Portas "AND" e "OR"

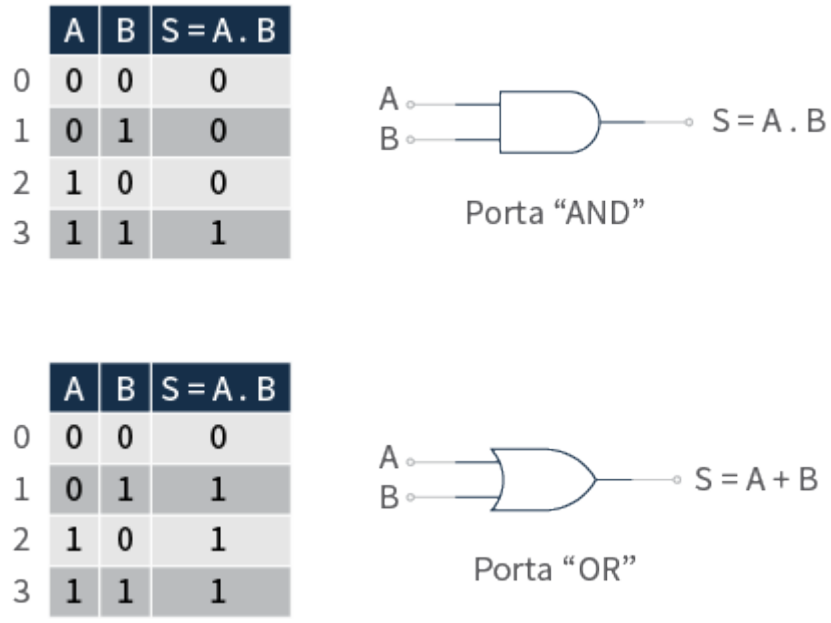
A porta AND é uma porta binária, ou seja, para externar o seu valor de saída, são necessárias duas variáveis de entrada. A saída será "1" apenas se as duas entradas também forem "1" simultaneamente.

- Simbologias da porta AND: $S = A . B$; $S = AB$; $S = A \wedge B$

Na simbologia da porta AND, o símbolo de "." pode ser omitido. Assim, por exemplo, a expressão S = A . B também poderá ser escrita na forma S = AB. Por sua vez, a porta OR é, assim como a porta AND, uma porta binária. A saída será "0" apenas se as duas entradas também forem "0" simultaneamente.

- Simbologias da porta OR: $S = A + B$; $S = A \vee B$

A figura a seguir ilustra os diagramas esquemáticos e as tabelas-verdade referentes às portas AND e OR.



	A	B	S = A . B
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

	A	B	S = A . B
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

Figura 6 - Tabelas-verdade e diagramas esquemáticos relativos às portas AND e OR.
Fonte: Elaborada pelo autor, 2019.

As portas AND e OR poderão ser conectadas a uma porta NOT em suas saídas, gerando as portas universais NAND e NOR, respectivamente. Saiba mais sobre elas a seguir!

Portas "NAND" e "NOR"

Como mencionado anteriormente, as portas NAND e NOR são portas ditas "universais", ou seja, podemos usar somente portas NAND ou somente portas NOR para implementar nosso sistema lógico digital. Os valores de saída das suas tabelas-verdade são, portanto, os valores complementados das saídas das portas AND e OR, respectivamente.

A figura a seguir ilustra os diagramas esquemáticos e as tabelas-verdade referentes às portas NAND e NOR.

	A	B	S = ~(A . B)
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0

Porta “NAND”

	A	B	S = ~(A + B)
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

Porta “NOR”

Como você pode notar nos diagramas esquemáticos da figura anterior, o sinal da negação anexado às saídas das portas AND e OR consiste apenas na circunferência. Como mencionado anteriormente, caso a negação esteja atrelada a alguma outra estrutura, para se representar a negação, basta utilizar a circunferência sem a seta indicativa da direção do fluxo do sinal (entrada → saída).

Portas “XOR” e “XNOR”

Por fim, temos as portas EXCLUSIVE-OR (OU-EXCLUSIVO ou XOR) e a porta COINCIDÊNCIA (ou XNOR). A porta XOR apresenta a saída valendo 1 se, e somente se, tivermos um único valor “1” representado pelas suas entradas. Por outro lado, a porta XNOR retrata o inverso da porta XOR, ou seja, a saída será “1” caso os dois valores de entrada sejam iguais.

A figura a seguir ilustra os diagramas esquemáticos, as simbologias e as tabelas-verdade referentes às portas XOR e XNOR.

	A	B	S = A XOR B
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

Porta “XOR”

	A	B	S = A XBIR B
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1

Porta “Coincidência”

Na figura anterior, podemos observar o diagrama esquemático da porta XNOR como sendo a porta XOR complementada. Porém, podemos encontrar também a porta XNOR sendo representada como uma porta AND com linha dupla junto às suas entradas.

VOCÊ SABIA?

Você sabia que a maioria dos operadores vistos aqui podem ser aplicados na programação para realizar a manipulação "bit-a-bit"? Essa técnica, denominada "bitwise" permite que, por exemplo, bits sejam alterados dentro de uma variável. Para saber mais, você poderá acessar a apostila escrita por Roberto Renna (2014), disponível em: https://www.telecom.uff.br/pet/petws/downloads/apostilas/arduino/apostila_de_programacao_arduino.pdf.

As portas aqui mencionadas farão parte das expressões booleanas (representando os operadores lógicos ou operadores booleanos). Para que possamos manipular as expressões, teremos que aplicar os teoremas da álgebra booleana, os quais veremos a seguir.

VAMOS PRATICAR?

Para uma melhor memorização do comportamento dos operadores lógicos e de suas respectivas simbologias frente aos diagramas esquemáticos, construa a tabela-verdade e indique a simbologia dos operadores a seguir:

a) NAND

b) XOR

c) OR

d) Coincidência

e) AND

1.2.3 Teoremas da álgebra booleana

Os teoremas da álgebra booleana nortearão a manipulação das expressões booleanas para, por exemplo, desenhar o diagrama esquemático ou calcular o valor resultante a partir da combinação dos valores das variáveis de entrada. Para começarmos, podemos mencionar a precedência dos operadores:

- 1º: parênteses
- 2º negação
- 3º operador AND
- 4 º operador OU, XOR, XNOR

Assim, ao nos depararmos com, por exemplo, a expressão booleana "**S = (~A + B) . C**", teremos o diagrama esquemático e a tabela-verdade ilustrados a seguir. Confira!

	A	B	C	(~A	+	B)	.	C
0	0	0	0	1	1	0	0	0
1	0	0	1	1	1	0	1	1
2	0	1	0	1	1	1	0	0
3	0	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	0
5	1	0	1	0	0	0	0	1
6	1	1	0	0	1	1	0	0
7	1	1	1	0	1	1	1	1

Figura 7 - Tabela-verdade e diagrama esquemático relativos à expressão S = (~A + B) . C.
Fonte: Elaborada pelo autor, 2019.

A figura anterior ilustra a tabela-verdade e o diagrama esquemático relativo à expressão "**S = (~A + B) . C**". Perceba que a ordem de manipulação dos operadores segue a precedência da álgebra booleana. A presença de uma inversão ligada a uma variável faz com que, no caso, a coluna relativa a "~A" receba o complemento de "A". Em função dos parênteses, a primeira operação a ser realizada consiste no operando OR (marcado em azul e denotado por "1º"). Após o preenchimento da coluna referente ao resultado do operador OR, efetua-se o operador AND — marcado na cor vermelha e referenciado como o 2º passo a ser tomado. A coluna referente ao resultado do operador AND (delimitada pelo retângulo vermelho) representa os resultados da expressão booleana, tendo em vista todas as combinações possíveis envolvendo as variáveis de entrada "A", "B" e "C".
Além das precedências, temos que nos atentar para o fato de que, assim como a álgebra matemática, a álgebra booleana também apresenta propriedades comutativa, associativa e distributiva, as quais você entenderá melhor a seguir!

• **Propriedade Comutativa**: a ordem das variáveis não interfere no resultado produzido. Exemplos:

A . B = B . A

A + B = B + A

A ⊕ B = B ⊕ A

• **Propriedade Associativa**: caso tenhamos variáveis conectadas por meio do mesmo tipo de operador, poderemos criar grupos (associar) para que possamos definir a ordem de manipulação. Exemplos:

(A . B) . C = A . (B . C) = A . B . C

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

(A ⊕ B) ⊕ C = A ⊕ (B ⊕ C) = A ⊕ B ⊕ C

• **Propriedade Distributiva**: na propriedade distributiva, procedemos à distribuição não somente da variável, mas também do operador associado à variável. Exemplos:

A . (B + C) = A . B + A . C

A + B . C = (A + B) . (A + C)

Além das propriedades citadas, uma quarta propriedade merece destaque na manipulação de expressões booleanas. Trata-se do "Teorema de De Morgan". Este teorema pode ser aplicado quando um operador "NOT" incidir, além das variáveis, sobre o próprio operador "AND" ou "OR". Nesse caso, temos que proceder, além da inversão das variáveis, à alteração do próprio operador, trocando o "AND" pelo "OR" e vice-versa:

a) ~(A + B) = ~A . ~B

b) ~(A . B) = ~A + ~B

Nesses casos, temos, em (a), a incidência da negação sobre o operador "OR". Nota-se que, após a aplicação do Teorema de De Morgan, o operador OR foi alterado para AND e as variáveis foram complementadas. Em (b), a negação abrange o operador AND, razão pela qual apareceu, após a aplicação do teorema, o operador OR.

Além dos teoremas e das propriedades, a álgebra booleana também contempla os postulados (IDOETA; CAPUANO, 2012): complemento; idempotência; identidade; elemento neutro; elemento de absorção. Confira!

- Complemento

  Pode ser aplicado sobre um valor booleano ou envolvendo operadores. No caso de valores lógicos e das portas AND e OR, temos:

  ~0 = 1

  ~1 = 0

  A . ~A = 0

  A + ~A = 1

- Idempotência

  Envolve a aplicação de um número par de negações, resultando na própria variável envolvida:

  ~(~ A) = A

- Identidade

  Ao aplicarmos um operador "AND" ou "OR" sobre apenas uma variável, temos, como resultado, a própria variável:

  A + A = A

  A . A = A

- Elemento neutro

  O elemento neutro representa o valor lógico (0 ou 1) que, quando aplicado a um operador, não modifica o valor resultante:

  A + 0 = A

  A . 1 = A

- Elemento de absorção

  O elemento de absorção, ao ser aplicado sobre um operador, faz com que, independentemente do valor da variável envolvida, o operador resulte no próprio valor aplicado:

  A + 1 = 1

  A . 0 = 0

Se aplicarmos as propriedades e postulados citados, podemos abstrair as seguintes propriedades:

A + A . B = A

A + ~A . B = A + B

(A + ~B) . B = A . B

A . B + A . ~B = A

(A + B) . (A + ~B) = A

A . (A + B) = A

A . (~A + B) = A . B

A .B + ~A . C = (A + C) . (~A + B)

Juntando o Teorema de De Morgan com a idempotência, conseguimos uma aplicabilidade prática: reescrever a expressão de forma mais simples. Por exemplo, imagine que temos a seguinte expressão lógica:

S = ~A . ~B

Nessa expressão, temos operador NOT sendo aplicado a cada variável. Assim, podemos aplicar os seguintes passos:

- **~A . ~B = ~(~(~A . ~B))** → nega-se a expressão duas vezes para que, de acordo com a idempotência, não se altere seu valor.
- **~(~(~A . ~B)) = ~(~~A + ~~B)** → aplica-se o Teorema de De Morgan na inversão mais interna.
- **~(~~A + ~~B)= ~(A + B)** → aplica-se a idempotência nas inversões que precedem as variáveis "A" e "B".
- **S = ~(A + B)** → expressão reescrita usando apenas um operador NOR.

Foi mencionado, anteriormente, que operadores NAND e NOR são ditos como universais. Dessa forma, aplicando os princípios vistos aqui, vamos representar os operadores básicos utilizando somente NAND e NOR:

- Operador NOT:

S = ~A → S = ~(A . A)

S = ~A → S = ~(A + A)

- Operador AND:

S = A . B → ~(~(A . B) . ~(A . B))

S = A . B → ~(~(A + A) + ~(B + B))

- Operador OR:

S = A + B → ~(~(A . A) . ~(B . B))

S = A + B → ~(~(A + B) + ~(A + B))

Para finalizar, convém mencionar um fato interessante na álgebra booleana: o princípio da dualidade. Esse princípio menciona que a partir de uma expressão booleana, ao efetuarmos a troca dos operadores lógicos (de AND para OR e vice-versa), além de trocarmos os valores de "0" para "1" e vice-versa, mantemos a equivalência entre as duas expressões booleanas.

Como exemplos, podemos citar as expressões booleanas abaixo:

A + 0 = A ↔ A . 1 = A

A + 1 = 1 ↔ A . 0 = 0

A + A = A ↔ A . A = A

A + ~A = 1 ↔ A . ~A = 0

Com a aplicação das propriedades citadas, já começamos a enveredar pelo campo da otimização, simplificação ou otimização de expressões booleanas. Simplificar uma expressão resulta em várias melhorias ao realizarmos a implementação física dos sistemas lógicos digitais.

VAMOS PRATICAR?

Conversamos sobre como preencher uma tabela-verdade produzindo, inclusive, o resultado da expressão booleana para cada combinação das variáveis de entrada. Construa a tabela-verdade das expressões a seguir:

a) S = A + ~B.C

b) S = A.(B + C) + ~A(C + ~D)

c) S = A + ~A.B

d) S = (A ⊕ B).(A + ~B)

Síntese

Chegamos ao fim desta unidade. Nesse nosso diálogo, você teve o primeiro contato com o mundo dos sistemas lógicos digitais. Inicialmente, abordamos os sistemas de numeração e como fazemos para realizar a conversão dentre os sistemas mais importantes do ponto de visto computacional: decimal; hexadecimal; octal; e binário. Como você pôde ver, essa área é baseada na manipulação de palavras cujos elementos (dígitos) são representados por bits. Por ser baseado na manipulação de bits, temos que conhecer, entender e manipular a álgebra booleana, seus operadores, propriedades e postulados. Na parte prática, os operadores lógicos são representados pelas portas lógicas, componentes básicos da eletrônica digital.

Nesta unidade, você teve a oportunidade de:

- ter um contato inicial com sistemas de numeração;
- conhecer, compreender e manipular informações utilizando a álgebra booleana;
- construir tabelas-verdade para representar o comportamento de uma expressão booleana em função de todas as combinações possíveis de suas variáveis de entrada;
- identificar e saber utilizar as propriedades, operadores e postulados.

Bibliografia

CANAL CEFOR. **Criação de Tabela-verdade**, 12 maio 2014. Disponível em: https://www.youtube.com/watch?v=mVJXZit3msA. Acesso em 16/09/2019.

IDOETA, I. V.; CAPUANO, F. G. **Elementos de Eletrônica Digital**. 41. ed. São Paulo: Érica, 2012.

INSTITUTO FEDERAL DE SANTA CATARINA. **DIG222802 AULA07**, 01 jun. 2016. Disponível em: https://wiki.sj.ifsc.edu.br/wiki/index.php/DIG222802_AULA07. Acesso em: 16/09/2019.

RENNA, R. B. **Apostila de Tópicos Especiais em Eletrônica II**: Introdução ao microcontrolador Arduino. Niterói: Universidade Federal Fluminense, 2014. Disponível em: https://www.telecom.uff.br/pet/petws/downloads/apostilas/arduino/apostila_de_programacao_arduino.pdf. Acesso em: 16/09/2015.

SOUSA, G. S. **Uma reavaliação do pensamento lógico de George Boole à luz da história da Matemática**. Dissertação (Mestrado) – Departamento de Pós-graduação em Educação, Universidade Federal do Rio Grande do Norte, Natal, 2005. Disponível em https://repositorio.ufrn.br/jspui/bitstream/123456789/14224/1/GiselleCS.pdf. Acesso em: 16/09/2019.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais: Princípios e Aplicações**. 12. ed. São Paulo: Pearson Education do Brasil, 2018.

VAHID, F.; LASCHUK, A. **Sistemas Digitais**: Projeto, otimização e HDLs. Porto Alegre: Bookman, 2008.