UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
Programa de Pós-Graduação em Matemática
Mestrado Profissional - PROFMAT/CCT/UFCG

Maria Eduarda de Oliveira Gomes

Os Números Binários Aplicados à Computação

Campina Grande - PB
Junho/2021

G633n Gomes, Maria Eduarda de Oliveira.
Os números binários aplicados à computação / Maria Eduarda de Oliveira Gomes. – Campina Grande, 2021.
109 f. : il. color.

Dissertação (Mestrado em Matemática) – Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, 2021.
"Orientação: Prof. Dr. José de Arimatéia Fernandes".
Referências.

1. Álgebra de Boole. 2. Números Binários. 3. Unicode. 4. Código ASCII. 5. Computação – Padrão UTf-8. I. Fernandes, José de Arimatéia. II. Título.

CDU 517.987.3(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
Programa de Pós-Graduação em Matemática
Mestrado Profissional - PROFMAT/CCT/UFCG

Maria Eduarda de Oliveira Gomes

Os Números Binários Aplicados à Computação

Trabalho de Conclusão de Curso apresentado ao Corpo Docente do Programa de Pós-Graduação em Matemática - CCT - UFCG, na modalidade Mestrado Profissional, como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. José de Arimatéia Fernandes

Campina Grande - PB
Junho/2021

Maria Eduarda de Oliveira Gomes

Os Números Binários Aplicados à Computação

Trabalho de Conclusão de Curso apresentado ao Corpo Docente do Programa de Pós-Graduação em Matemática - CCT - UFCG, na modalidade Mestrado Profissional, como requisito parcial para obtenção do título de Mestre.

Trabalho aprovado. Campina Grande - PB, 10 de junho de 2021:

Prof. Dr. José de Arimatéia Fernandes
Orientador

Prof. Dr. Rodrigo Cohen Mota Nemer - UFCG
Convidado 1

Prof. Dr. Lenimar Nunes de Andrade - UFPB
Convidado 2

Campina Grande - PB
Junho/2021

Honro o fechamento deste ciclo dedicando este trabalho a todos os professores que tive ao longo dos meus 23 anos de idade, sobretudo aqueles que ultrapassaram as barreiras do conhecimento científico e serviram como fonte de inspiração e motivação para minha evolução pessoal, acadêmica e profissional. De maneira especial, dedico à minha mãe Agueda e minhas tias Luciene e Leoneide, minhas primeiras professoras, que me ensinaram as primeiras letras e os primeiros números, me fazendo, logo cedo, despertar uma paixão pela matemática.

Agradecimentos

Após dois anos de muito estudo e dedicação, chego ao final deste curso de mestrado, e não poderia deixar de agradecer a algumas pessoas que estiveram comigo e foram essenciais no decorrer deste período. Por isso, através de poucas, mas sinceras palavras, destaco e agradeço a importância que elas tiveram nesta consquista e continuam tendo na minha vida.

Antes, porém, elevo meus pensamentos aos céus, e, humildemente, agradeço a Deus, ser onipotente, o mestre dos mestres, pela dádiva da vida e por ter me mantido na trilha correta durante esta etapa acadêmica, sobretudo no decorrer desta pesquisa, me dando saúde, forças e sabedoria para chegar ao fim, me permitindo, assim, realizar este sonho nesta existência.

Agora, um agradecimento todo especial, cheio de amor e gratidão, vai para a minha família, de maneira única e especial para a minha mãe Agueda, pelo amor incondicional, dedicação, cuidado e por tudo que ela sempre fez por mim, principalmente por ter me ofertado uma educação de qualidade; e para a minha amada irmã Maria Eloísa, pelo companheirismo, parceria e por inúmeras vezes ter me compreendido, suprindo a minha ausência em casa. A vocês, e aos demais membros da minha família materna, serei eternamente grata por tudo que sou, por tudo que consegui conquistar e pelo carinho sempre dedicado a mim.

Aos meus amigos, pessoas que não tenho palavras para definir e descrever o quão importantes eles são em minha vida. Aqui destaco Andréa, Francisca e a pequena Ana Júlia, as quais quero agradecer, sobretudo, pela amizade, companheirismo, cumplicidade e incentivo de sempre.

A todos que fazem a Escola Monsenhor, pela compreensão das minhas ausências nas sextas-feiras, onde agradeço na pessoa da gestora e minha amiga Jussara, pessoa que tenho grande respeito, admiração e estima.

Ao meu ex-professor Me. Bruno Lopes pelo incentivo e por ter me presenteado com livros e apostilas muito úteis durante o curso.

Ao orientador da minha dissertação, Prof. Dr. José de Arimatéia Fernandes, pelos importantes ensinamentos, valiosas contribuições e por todo empenho durante este trabalho, para que, juntos, concluíssemos esta pesquisa.

Aos membros da Banca examinadora Prof. Dr. Rodrigo Cohen Mota Nemer e Prof. Dr. Lenimar Nunes de Andrade, pelas observações e contribuições para melhoria deste trabalho.

À UAMat/UFCG, pelo ensino de qualidade oferecido a toda comunidade acadêmica, onde agradeço na pessoa do Coordenador do curso Prof. Dr. Romildo Lima.

Sou grata também a todos os professores que tive ao longo deste curso de mestrado, por todo conhecimento transmitido a mim e a meus colegas colegas de turma, os quais também agradeço pelo companheirismo acadêmico ao longo destes dois últimos anos.

À Sociedade Brasileira de Matemática - SBM, por oferecer o curso em rede nacional.

Por fim, mas não menos importante, os meus agradecimentos vão para aquela instituição que me fez despertar uma maior admiração e amor pela matemática. Minha eterna gratidão ao IFPE - Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco - Campus Pesqueira, por todos os importantes e valiosos conhecimentos a mim ofertados durante a minha graduação.

"A Matemática é a única linguagem
que temos em comum com a natureza."
- Stephen Hawking

Resumo

O presente trabalho de conclusão de curso tem como principal objetivo abordar conceitos pertinentes ao sistema de numeração binário, enfatizando, sobretudo, como eles são aplicados à computação. Através de um levantamento bibliográfico, ficou constatado como esses números são úteis no nosso cotidiano, tendo uma de suas principais aplicações voltada para programação de equipamentos digitais, como computadores e celulares, por exemplo, uma vez que estes dispositivos apenas entendem 0 e 1, isto é, a linguagem binária. No entanto, é importante salientar que, dentro desse contexto, estes números não exprimem quantidades, mas sim estados do sistema, que são compreendidos através da passagem de correntes elétricas ou da ausência dela, representando o 1 e o 0, respectivamente, assim como é sugerido pela Álgebra de Boole. Por fim, será mostrado como tudo isso é feito na prática através de um pequeno chip denominado processador e fazendo uso do código ASCII e do Unicode, com ênfase para o padrão UTF-8, utilizado mundialmente como linguagem padrão entre os computadores.

Palavras-chave: Números Binários. Álgebra de Boole. Unicode.

Abstract

The main objective of this course conclusion work is to address concepts relevant to the binary numbering system, emphasizing, above all, how they are applied to computing. Through a bibliographic survey, it was found how these numbers are useful in our daily lives, having one of its main applications focused on programming digital equipment, such as computers and cell phones, for example, since they only understand 0 and 1, that is, the binary language. However, it is important to note that, within this context, these numbers do not express quantities, but rather states of the system, which are understood through the passage of electrical currents or the absence of it, representing 1 and 0, respectively, as suggested by Boole's Algebra. Finally, it will be shown how all this is done in practice through a small chip called a processor and making use ASCII code and Unicode, with emphasis on the UTF-8 standard, used worldwide as a standard language among computers.

Keywords: Binary Numbers. Boole Algebra. Unicode.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Para que serve a Matemática? Essa é uma pergunta recorrente entre os alunos do Ensino Fundamental ao Superior. Claro que, no mundo tecnológico de hoje em dia, não temos dúvidas da importância que a matemática tem nos cálculos de estruturas da engenharia, nos modelos de propagação da biologia e nos vários recursos computacionais presentes no nosso dia a dia, como os computadores, celulares, impressoras, entre tantos outros.

Mas, se tratando de modelos computacionais, como uma máquina consegue executar operações matemáticas como o cérebro humano e o que, de fato, os computadores leem e executam quando os programas que criam as rotinas e instruções de algumas operações computacionais estão sendo processados? Naturalmente, os computadores necessitam de instruções para realizarem as suas tarefas, mas qual é a linguagem que eles entendem e permitem com que vários programas, aplicativos e rotinas sejam processados com sucesso?

A linguagem que os computadores entendem é a dos números, mais precisamente, a dos números binários 0 e 1. Isso ocorre porque, na verdade, os transistores que formam as placas dos computadores constituem uma chave do tipo liga e desliga, de acordo com a voltagem operada de 0 a 5 volts. Então, esse pulso elétrico, ou a ausência dele, pode ser interpretado como o sim ou o não, o verdadeiro ou o falso, ou seja, a opção entre dois estados pode ser compreendida unicamente por zeros e uns da linguagem binária.

Sendo assim, tudo que precisamos “falar” para o computador em termos de instruções pode ser “lido” pela máquina como zeros e uns. Mas a nossa linguagem de textos do dia a dia e o nosso sistema de numeração que usamos, o decimal, não podem ser entendidos pelo computador. Por isso, se faz necessário converter a nossa linguagem de textos e números decimais para a linguagem binária dos computadores.

Mas como tudo isso é feito? Neste trabalho, responderemos a essa e a outras questões interessantes envolvendo o sistema de numeração na base 2, ou seja, o sistema binário, além de mostrarmos como essa linguagem puramente matemática é convertida e entendida pelos computadores. Antes, porém, é importante destacar que aqui estaremos utilizando o termo "números binários", em alguns casos, para fazer referência ao sistema de numeração binário, dependendo do contexto que se está inserido.

Diante do exposto, este Trabalho de Conclusão de Curso do PROFMAT - UFCG se dará através de uma pesquisa do tipo qualitativa, onde também se fará presente a pesquisa bibliográfica, cujo objeto central de estudo foi motivado a partir da curiosidade em entender como funciona a programação das máquinas, tão úteis no nosso cotidiano.

Outro fator que também motivou a escolha do tema de pesquisa envolvendo os números binários foi o fato de que $1 + 1$, aqui, não é 2, e, sim 10, fato que deixa muitas pessoas curiosas e sem entender o porquê disso acontecer, o que desperta, nelas, o interesse em compreender um pouco mais sobre o assunto e, consequentemente, em querer saber alguma aplicação desses conceitos.

1.1 Objetivos

Este trabalho tem como objetivo geral compreender como funciona a programação dos computadores através do sistema de numeração na base 2, onde buscaremos, através de um levantamento bibliográfico, atingi-lo, destacando os seguintes objetivos específicos:

- Estudar a história, a aritmética e algumas propriedades dos números binários;
- Mostrar que todo número pode ser escrito na base 2;
- Apresentar a Álgebra de Boole, incluindo importantes definições, axiomas, propriedades e teoremas;
- Expor aplicações desta álgebra;
- Compreender o funcionamento dos processadores;
- Tratar do código ASCII e do Unicode, com ênfase no padrão UTF-8;
- Descrever como acontece a comunicação entre humanos e máquinas.

1.2 Organização

Para podermos alcançar os objetivos apresentados anteriormente, este trabalho estará estruturado da seguinte maneira:

De início, a Introdução, que também podemos considerá-la como o Capítulo 1, trará, de forma rápida e resumida, como funciona a linguagem e a programação dos computadores, a metodologia de pesquisa utilizada, a motivação da escolha do objeto de estudo, os objetivos a serem alcançados e a estrutura de todo este Trabalho de Conclusão de Curso.

No Capítulo 2, intitulado “A Aritmética Binária”, vamos nos familiarizar com o sistema de numeração posicional na base 2 e suas propriedades. Além disso, provaremos que todo inteiro positivo pode ser escrito na base 2 e, por fim, mostraremos como os sistemas binário e decimal se relacionam.

O Capítulo 3 apresentará a Álgebra de Boole, que constitui o sistema lógico construído a partir da linguagem binária, enfatizando, sobretudo, o contexto histórico e importantes definições, axiomas, propriedades e teoremas, com suas respectivas demonstrações. Também mostraremos a relação desta álgebra com a teoria dos conjuntos e com a álgebra tradicional, onde são utilizados conceitos já estudados e conhecidos aplicados a essa "nova" álgebra, bem como algumas de suas aplicações, destacando seu uso nos circuitos lógicos, circuito elétricos, no buscador de Google e em jogos de lógica.

O Capítulo de número 4 trata dos circuitos lógicos, que são construídos através de portas lógicas; de outros operadores, além dos básicos; por fim, mostra algumas aplicações da Álgebra de Boole.

No quinto Capítulo, Introdução aos Computadores, trataremos dos conceitos de *bit* e *byte*; falaremos também dos processadores de computadores e como se dá seu funcionamento; em seguida, apresentaremos o código ASCII e o Unicode, destacando o padrão UTF-8 e como eles são utilizados na linguagem computacional, a partir do sistema binário.

Por fim, apresentaremos as Conclusões obtidas ao final desta pesquisa, as referências utilizadas na construção deste trabalho e os anexos, onde encontra-se a tabela ASCII e a tabela do Unicode/Padrão UTF-8.

2 Os Números Binários

Neste capítulo, vamos apresentar um pouco da história dos números binários, onde será mostrado como e por quem esses números foram utilizados no passado; como funciona a expansão relativa a uma base b, $b \in \mathbb{N}$, com a enunciação de propriedades e suas respectivas demonstrações; algumas propriedades existentes entre os números decimais e binários, incluindo a conversão entre números nessas bases; e, por fim, traremos as ideias de soma, subtração, multiplicação e divisão com números binários, com exemplos numéricos de cada uma dessas operações.

2.1 Contexto Histórico

Nos primórdios da existência humana na terra, os homens viviam em cavernas e obtinham o seu próprio alimento através da caça e da pesca. No entanto, com o passar do tempo, foi-se percebendo a importância de produzir sua alimentação através da agricultura e do pastoreio e, com isso, surgiu a necessidade de contar.

Nesse sentido, o homem "desenvolveu a capacidade de comparar conjuntos e estabelecer entre eles uma correspondência de noções quantitativas como pouco e muito, pequeno e grande". (MARTINES, 2019, p. 13)

De acordo com (EVES, 2004, p. 23),

> As pessoas comerciavam entre si e havia a necessidade de anotar a parte de cada família na caçada, ambas as atividades dependiam da ideia de contar, um prelúdio de pensamentos científicos. Alguns povos da Idade da Pedra, como a tribo Sioux, tinham calendários pictográficos que registravam várias décadas de história. Todavia, afora os sistemas de contagem primitivos, tudo o mais teve de esperar o desenvolvimento da agricultura, intensiva em grande escala, que requeria uma aritmética mais sofisticada.

Assim, com a necessidade de contagem começando a se difundir nas sociedades, surgiu também uma espécie de agrupamentos, ou seja, um conjunto de particularidades comuns que iriam classificar grupos e subgrupos. Nesse sentido, temos o que chamamos de Sistema de Numeração, que é um método utilizado para representar quantidades, onde cada número tem representação única. Segundo (MIYASCHITA, 2002), "os primeiros sistemas de escrita numéricas que se conhece são os dos egípcios e os dos sumérios, surgidos por volta de 3.500 a.C. ".

Esses sistemas podem ter diferentes bases, que são, de uma forma geral, a quantidade de algarismos distintos que o compõem. Atualmente, o sistema de numeração que utilizamos tem base 10, também chamado base decimal, que é composto por 10

dígitos, que vão de 0 a 9. Assim, nesse sistema, a partir da quantidade 10 não há novos dígitos, ou seja, para representar números maiores ou iguais a 10, são utilizados os algarismos já existentes. Esse fato é de extrema importância visto que estamos diante de um sistema de numeração posicional.

Com o desenvolvimento tecnológico, sistemas começaram a ser estudados em busca de maior facilidade de representação interna codificada. Dentre os mais comuns, podemos citar os sistemas binários, octal, decimal e hexadecimal, que se adequam às especificidades dos equipamentos a serem utilizados.

Podemos ter um sistema de numeração em qualquer base desejada, digamos, base b, onde os algarismos irão de 0 a $b-1$, se $b \leq 10$. Se $b > 10$, então utilizam-se dígitos alfanuméricos adicionais tais como A, B, C, D. No entanto, neste trabalho, iremos nos deter apenas ao sistema de numeração de base 2, que também é chamado de sistema de numeração binário, que é composto apenas de dois algarismos: 0 e 1. Este sistema de numeração foi proposto pelo matemático alemão Gottfried Leibniz devido à sua simplicidade, pois emprega a menor base possível de numeração (MIYASCHITA, 2002).

Navegando pela história, os primeiros vestígios de uso desse sistema de numeração aconteceu por volta do século III a.C., pelo indiano Pingala, que representou os números de 1 a 8, na nossa base usual, como 001, 010, 011, 100, 101, 110, 111, 1000, sequencialmente e na linguagem dos algarismos indo-arábicos (SANTOS, 2020).

Há registros de que um matemático chinês chamado Shao Yong, no século XI, que usava uma representação sequencial de 0 a 63, tenha utilizado uma aritmética binária com essa representação, mas como existe pouca bibliografia desse fato, há controvérsias nesta história. (SANTOS, 2020).

No entanto, foi com o matemático alemão Gottfried Wilhelm Leibniz, nascido em 1646, que o sistema binário foi aperfeiçoado e efetivamente aplicado. Leibniz escreveu um dos primeiros artigos relacionados ao tema, intitulado *Explication de l'Arithmétique Binaire*.

> Mais do que toda sua filosofia, mais do que sua *Monadologia*, que tanto prezava, seu *Explication de l'Arithmétique Binaire* apresenta, de forma simples e vibrante, as bases da moderna ciência da computação, do funcionamento dos computadores e dos onipresentes telefones celulares. (LOPES, 2011, p. 90)

Ainda segundo (SANTOS, 2020), também é do autor a teoria de que todo raciocínio lógico por ser reduzido a uma combinação ordenada de elementos como números, palavras, sons ou cores, teoria esta que se tornou a base teórica dos computadores modernos.

Portanto, para muitos autores, é atribuído a Leibniz a criação do sistema de numeração na base 2.

Figura 1 – Leibniz (1646 – 1716)

Fonte: http://ecalculo.if.usp.br/historia/leibniz.htm

2.2 Expansão Relativa a uma Base b

Vejamos, agora, um importante teorema da matemática, talvez o mais importante deste capítulo, que diz que qualquer número inteiro pode ser escrito de maneira única em qualquer base b desejada. Aqui nos deteremos, de modo particular, à base 2, tema central deste trabalho.

Teorema 2.1. *Sejam dados os números inteiros $a > 0$ e $b > 1$. Existem números inteiros $n \geq 0$ e $0 \leq r_0, r_1, \cdots, r_n < b$, com $r_n \neq 0$, univocamente determinados, tais que $a = r_0 + r_1 \cdot b + r_2 \cdot b^2 + \cdots + r_n \cdot b^n$.*

Demonstração. Vamos demonstrar o Teorema 2.1 por Indução Completa sobre a.

Existência:

Se $0 < a < b$, basta tomar $n = 0$ e $r_0 = a$.

Suponhamos o resultado válido para todo natural menor do que a, onde $a \geq b$. Vamos prová-lo para a. Pela divisão euclidiana, existem q e r, únicos, tais que

$$a = b \cdot q + r, \text{ com } 0 \leq r < b. \tag{2.1}$$

Como $0 < q < a$, pela hipótese de indução segue-se que existem números inteiros $n' \geq 0$ e $0 \leq r_1, r_2, \cdots, r_{n'+1} < b$, com $r_{n'+1} \neq 0$, univocamente determinados, tais que

$$q = r_1 + r_2 \cdot b + \cdots + r_{n'+1} \cdot b^{n'}. \tag{2.2}$$

Levando em conta as igualdades entre as equações 2.1 e 2.2, temos que

$$a = b \cdot q + r = b \cdot \left(r_1 + r_2 \cdot b + \cdots + r_{n'+1} \cdot b^{n'}\right) + r, \tag{2.3}$$

de onde o resultado segue, pondo $r_0 = r$ e $n' + 1 = n$.

Unicidade:

Para $0 < a < b$, o resultado é trivial. Seja, então, $a \geq b$ e suponha que o resultado seja válido para todo natural menor do que a, digamos q, onde $0 < q < a$. Queremos provar que o resultado vale para $q = a$.

Para isto, consideremos as seguintes representações para a:

$$a = r_n \cdot b^n + r_{n-1} \cdot b^{n-1} + \ldots r_1 \cdot b + r_0 \tag{2.4}$$

e

$$a = r'_n \cdot b^n + r'_{n-1} \cdot b^{n-1} + \ldots r'_1 \cdot b + r'_0. \tag{2.5}$$

Então, como se trata do mesmo inteiro positivo a, podemos escrever

$$a = b\left(r_n \cdot b^{n-1} + \cdots + r_1\right) + r_0 = b\left(r'_n \cdot b^{n-1} + \cdots + r'_1\right) + r'_0.$$

Agora, pondo $q = r_n \cdot b^{n-1} + \cdots + r_1$ e $q' = r'_n \cdot b^{n-1} + \cdots + r'_1$ é fácil ver que $q < a$ e $q' < a$ e, portanto, a hipótese de indução nos diz que as representações q e q' são univocamente determinadas. Por outro lado, como $0 \leq r_0$, $r'_0 < b$, segue que 2.4 e 2.5 expressam a divisão euclidiana de a, com divisor b, e restos respectivamente iguais a r_0 e a r'_0; daí, a unicidade do quociente e do resto nos garante que

$$q = q' \quad \text{e} \quad r_0 = r'_0$$

e as representações são indistintas. ■

De acordo com a expansão acima, b é a base do sistema de numeração e r_i será cada um dos dígitos do número a, sendo que o índice i indica a posição relativa de cada dígito, ou seja,

$$a = r_n \cdot b^n + r_{n-1} \cdot b^{n-1} + r_{n-2} \cdot b^{n-2} + \cdots + r_1 \cdot b + r_0 = r_n r_{n-1} r_{n-2} \cdots r_2 r_1 r_0.$$

Na verdade, esse teorema nos permite escrever qualquer número inteiro utilizando um sistema posicional, onde "os algarismos têm um valor dependendo de qual posição ocupam no número" (SANTOS; SANTANA, 2013, p. 2). De modo particular, temos que todo número natural pode ser escrito como soma de potências de 2, conforme provado no Teorema 2.1, ou seja,

$$N = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \cdots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

onde cada dígito a_i é igual a 0 ou a 1.

A mesma ideia pode ser estendida aos números racionais, onde utilizaríamos as potências negativas da base. (MARTINES, 2019)

Como consequência do Teorema 2.1, temos:

Corolário 2.2. *Duas expressões escritas numa mesma base b, digamos $a = a_n a_{n-1} \cdots a_1 a_0$ e $a' = a'_n a'_{n-1} \cdots a'_1 a'_0$, representam o mesmo número inteiro se, e somente se, $a_i = a'_i$, para todo i, $0 \leq i \leq n$.*

Demonstração. Decorre imediatamente da unicidade de representação de um inteiro numa base b qualquer. ■

Corolário 2.3. *Se $a = a_n a_{n-1} \cdots a_1 a_0$ e $a' = 0a_n a_{n-1} \cdots a_1 a_0$, então $a = a'$.*

Demonstração. $a' = 0a_n a_{n-1} \cdots a_1 a_0 = 0 \cdot b^{n+1} + a_n \cdot b^n + \cdots + a_1 \cdot b + a_0 = 0 + a_n \cdot b^n + \cdots + a_1 \cdot b + a_0 = a_n \cdot b^n + \cdots + a_1 \cdot b + a_0 = a_n a_{n-1} \cdots a_1 a_0 = a$. ■

Em virtude do Corolário 2.3, costuma-se dizer que o zero, ao ser escrito na extrema esquerda de uma expressão, é um algarismo não significativo, pois sua omissão não altera o valor da expressão considerada. Isto nos diz que, dados a e a' distintos, sempre podemos considerar ambos com o mesmo número de algarismos, bastando, para isso, que completemos com zeros não significativos a expressão decimal com menor número de dígitos. (LUNA, 2013)

Vimos, portanto, que qualquer número pode ser escrito na base que se queira. Logo, fica provado que podemos escrever qualquer número na base 2.

É importante enfatizar a questão dos números negativos nesta base, pois, diferente de como acontece na decimal, que é necessário e suficiente apenas acrescentar o sinal de menos $(-)$ antes de escrever o número, nos binários trabalha-se com o chamado complemento de um número, ou seja, o que falta nesse número para atingir o valor da base, que é 2. Nos racionais, a ideia é trabalhar com a mesma expansão, no entanto, algumas potências receberão expoentes negativos.

Não iremos aprofundar o nosso estudo a respeito dos números negativos e racionais, visto que foge dos nossos objetivos, mas é de extrema importância ressaltar a existência deles. Portanto, aqui nos deteremos apenas aos números inteiros positivos, ou seja, aos números naturais.

Trataremos agora de um algoritmo que permite fazer a expansão de qualquer número relativamente à base b, que consiste em aplicar a divisão euclidiana sucessivas vezes, como segue:

$$a = b \cdot q_0 + r_0 \text{ com } r_0 < b,$$

$$q_0 = b \cdot q_1 + r_1 \text{ com } r_1 < b,$$

$$q_1 = b \cdot q_2 + r_2 \text{ com } r_2 < b$$

e assim sucessivamente.

Como $a > q_0 > q_1 > \cdots$, deveremos, em um certo ponto, ter que $q_{n-1} < b$ e, assim, de

$$q_{n-1} = b \cdot q_n + r_n \text{ com } r_n < b,$$

decorre que $q_n = 0$, o que implica $0 = q_n = q_{n+1} = q_{n+2} = \cdots$, e, portanto, $0 = r_{n+1} = r_{n+2} = \cdots$.

Logo, temos

$$a = r_0 + r_1 \cdot b + \cdots + r_{n-1} \cdot b^{n-1} + r_n \cdot b^n.$$

Nesse sentido, a expansão dada na base b nos fornece um método para representar os números naturais. Quando, por exemplo, a expansão é de acordo com a base $b = 10$, que é a base decimal, os números são representados a partir de uma sequência de números que vão de 0 a 9; já quando esta se dá na base $b = 2$, o número é dito binário e todo número é representado, consequentemente, por uma sequência de 0 e 1.

2.3 Relação entre os Números Binários e os Decimais

O sistema de numeração usual é o decimal, e o tema central deste trabalho são os números binários. Portanto, vamos apresentar como se faz a conversão números de decimais para binários e vice-versa, apresentando, também, exemplos numéricos de cada um.

Para converter de decimais para binários, devemos utilizar as divisões sucessivas por 2 e prestar atenção nos restos que elas deixam, em seguida, pegar o "novo" quociente (resultado da divisão anterior) e dividi-lo novamente por 2, focando sempre nos restos. Este procedimento deve ser repetido até que o quociente seja 0. O número binário procurado será a junção de todos os restos, dispostos lado a lado, em ordem contrária em relação à qual foram encontrados, ou seja, do último para o primeiro.

Exemplo 1. *Vamos transformar o número* 39*, que está na base decimal, para a base binária, através de divisões sucessivas:*

$$39 = 2 \cdot 19 + 1,$$

$$19 = 2 \cdot 9 + 1,$$

$$9 = 2 \cdot 4 + 1,$$

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

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

$$1 = 2 \cdot 0 + 1.$$

Focando nos valores dos restos encontrados, do último para o primeiro, temos

$$(39)_{10} = (100111)_2\,,$$

ou, simplesmente,

$$39 = (100111)_2\,.$$

É importante levar em consideração que, como a base padrão é 10, não há necessidade de destacá-la. Sendo assim, sempre que um número não tiver uma base específica, consideraremos que ela será a decimal.

Para o procedimento inverso, ou seja, para converter de binários para decimais, basta pegar o número e desenvolvê-lo, individualmente, como um produto entre ele e uma potência de 2, que terão expoentes 0, 1, 2, ... , quando acompanhar o valor que corresponder à unidade, à dezena, à centena, ..., respectivamente. Para encontrar o número em decimal, deve-se somar cada resultado obtido.

Exemplo 2. *Agora, iremos converter o número* $(100111)_2$ *para base decimal.*

$$1 \cdot 2^5 = 1 \cdot 32 = 32,$$

$$0 \cdot 2^4 = 0 \cdot 16 = 0,$$

$$0 \cdot 2^3 = 0 \cdot 8 = 0,$$

$$1 \cdot 2^2 = 1 \cdot 4 = 4,$$

$$1 \cdot 2^1 = 1 \cdot 2 = 2,$$

$$1 \cdot 2^0 = 1 \cdot 1 = 1.$$

O número procurado será o resultado da soma $32 + 0 + 0 + 4 + 2 + 1$*, que é* 39*.*
Logo, temos que $(100111)_2 = 39$*.*

Outra importante relação que existe entre o sistema binário e o decimal é a correspondência entre as progressões, onde $[(10)^n]_2 = 2^n$, que pode ser facilmente demonstrado utilizando indução completa sobre n.

Demonstração. Com efeito, para $n = 0$, o resultado é óbvio, isto é, $\left[(10)^0\right]_2 = (1)_2 = 1 = 2^0$.

Supondo que a afirmação seja válida para todo número natural menor que ou igual a n, para $n + 1$ teremos

$$\left[(10)^{n+1}\right]_2 = \left[(10)^n \cdot (10)^1\right]_2 = [(10)^n]_2 \cdot \left[(10)^1\right]_2 = 2^n \cdot 2^1 = 2^n \cdot 2 = 2^{n+1}.$$

Portanto, pelo princípio de indução completa, a propriedade é válida para todo número natural n. ■

Numericamente, esta relação funciona da seguinte maneira:

$$\left[(10)^0\right]_2 = (1)_2 = 1 = 2^0,$$

$$\left[(10)^1\right]_2 = (10)_2 = 2 = 2^1,$$

$$\left[(10)^2\right]_2 = (100)_2 = 4 = 2^2,$$
$$\left[(10)^3\right]_2 = (1000)_2 = 8 = 2^3,$$
$$\left[(10)^4\right]_2 = (10000)_2 = 16 = 2^4,$$
$$\left[(10)^5\right]_2 = (100000)_2 = 32 = 2^5,$$
$$\left[(10)^6\right]_2 = (1000000)_2 = 64 = 2^6,$$
$$\left[(10)^7\right]_2 = (10000000)_2 = 128 = 2^7$$

e assim sucessivamente.

Esta relação também pode ser um mecanismo utilizado para conversão de decimais para binários e vice-versa, por meio da decomposição do número e do valor equivalente entre estas bases.

Assim, por exemplo, para converter o número decimal 39 para base 2, devemos transformá-lo numa soma de potências de 2, isto é,

$$39 = 32 + 4 + 2 + 1 = 2^5 + 2^2 + 2^1 + 2^0,$$

e, pegando os correspondentes binários, temos que

$$39 = 32+4+2+1 = 2^5+2^2+2^1+2^0 = (100000)_2+(100)_2+(10)_2+(1)_2 = (100111)_2\,.$$

De modo análogo, para transformar o número $(100111)_2$ para a base 10, partimos da decomposição de 100111 em termos de potências de 10, ou seja,

$$100111 = 100000 + 100 + 10 + 1,$$

onde, pegando os correspondentes decimais, temos

$$(100111)_2 = (100000)_2+(100)_2+(10)_2+(1)_2 = (32)_{10}+(4)_{10}+(2)_{10}+(32)_1 = (39)_{10}\,.$$

Conforme verifica (MARTINES, 2019, p. 33),

> Uma diferença básica e fundamental entre esse sistema [binário] e o sistema decimal é a quantidade de algarismos utilizados para representar um número [...]. Verificamos que, de maneira geral, quanto menor a base, mais algarismos são necessários, tornando-se menos prático quando se trata da representação de números grandes. Por outro lado, o fato de trabalhar com essa base torna-se mais ágil quando realizamos as operações, pelo fato de não utilizarmos outros dígitos, senão 0 e 1.

2.4 Operações Fundamentais nos Binários

Tendo compreendido um pouco da história dos números binários, de provarmos que todo número pode ser escrito na base 2 e de ter visto como acontece a conversão entre a base decimal e binária e algumas relações importantes, vamos agora ver como acontece as operações elementares (adição, subtração, multiplicação e divisão) nesta base.

De acordo com (MARTINES, 2019, p. 41)

> As operações aritméticas no sistema de numeração posicional de qualquer base b são análogas àquelas que estamos acostumados na base decimal. Assim, as mesmas regras que usamos para adição e multiplicação no sistema decimal são válidas para números escritos em qualquer outro sistema de uma base b qualquer.

Nesta seção, sempre que estivermos considerando números e/ou operações, estaremos sempre nos referindo à base binária, a menos que seja explicitado outra base.

2.4.1 Adição

(FRANZON, 2015, p. 156) afirma que

> O *Explication de l'Arithmétique Binaire* lembra que, no sistema decimal, utilizamos os algarismos de 0 a 9 e diz que, quando chegamos ao dez, iniciamos novamente a contagem escrevendo dez como 10, dez vezes dez, ou cem, como 100, e dez vez cem, ou mil, como 1000. No sistema binário, utilizamos os algarismos 0 e 1 e, quando chegamos ao dois, recomeçamos a contagem, escrevendo dois como 10, e dois vezes dois, ou quatro, como 100, e dois vez quatro, ou oito, como 1000.

Assim, no sistema decimal, não existe um algarismo único para representar a dezena e, nele, escrevemos o algarismo 1(um) seguido do 0(zero) para representá-la. Em outras palavras, o algarismo 1 significa que há um grupo de uma dezena e o 0 que não há unidades. No sistema binário, a ideia é análoga. Para representar o dois, que, nesse sistema, representa a dezena, vamos usar os algarismos 1(um) e 0(zero). Essa será a ideia fundamental para a realização de somas envolvendo números binários.

Nesse sistema, contamos de dois em dois, onde cada 2 unidades de primeira ordem, equivalem a 1 unidade da segunda ordem; cada 2 unidades da segunda ordem equivalem a 1 unidade da terceira ordem e assim sucessivamente (MARTINES, 2019). Em outras palavras, consideramos o "0" como dígito da unidade e transportamos o "1", que, na verdade, representa o 10 (dois), para ordem imediamente superior, no caso, a das dezenas. O mesmo aconteceria se estivéssemos tratando de outras ordens. Este algoritmo também pode ser interpretado como a ideia do "vai um", muito estudado e já conhecido nas operações com decimais.

Temos apenas quatro possibilidades de somas nos binários. São elas:

$$0 + 0 = 0,$$

$$0 + 1 = 1,$$

$$1 + 0 = 1 \text{ e}$$

$$1 + 1 = 10.$$

Neste último caso, o transporte do 1 é imediato para próxima ordem.

Exemplo 3. *Vamos realizar a operação* $1001 + 0110$.

Para facilitar visualmente nosso cálculo, vamos reescrever a operação da seguinte forma, respeitando a posição ordinal de cada algarismo:

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

Em geral, começamos o cálculo pela operação de menor ordem, depois a de ordem imediatamente superior, e assim sucessivamente até o término da soma. Em outras palavras, a operação é efetuada de direita para esquerda.

Assim, somando sem complicações, obtemos:

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

Portanto, temos $1001 + 0110 = 1111$. *No sistema decimal, isso significa que* $9 + 6 = 15$.

Neste exemplo, também poderíamos considerar a soma $1001 + 110$, que chegaríamos ao mesmo resultado, pois, como já foi provado no Corolário 2.3, o 0 da exterma esquerda é não significativo. Colocamos-no-lo, apenas, com o intuito de igualar o número de dígitos de cada parcela.

Exemplo 4. *Somar* 1011 *a* 1010.

Primeiramente, vamos representar cada parcela com os números um abaixo do outro, de modo que os algarismos de mesma ordem fiquem alinhados.

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

Da direita para esquerda, temos $1+0=1$ *e* $1+1=10$*, onde colocar-se-á o* 0 *na mesma posição ordinal e o* 1 *vai para ordem imediatamente superior (ideia do "vai um"), que será acrescido à próxima soma; na terceira adição, temos* $0+0+1$*, sendo que esta última parcela refere-se ao* 1 *da soma anterior, que tem como resultado* 1*; e, por fim, temos* $1+1=10$*, onde ficará o* 0 *na ordem desta soma e o* 1 *irá ocupar a próxima ordem, sendo que não existem mais algarismos a serem somados, o que implica que o* 1 *ocupará, imediatamente, tal posição. Em termos matemáticos, teremos:*

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

Portanto, $1011+1010=10101$*, ou seja,* $11+10=21$*.*

2.4.2 Subtração

Nosso foco agora será a operação de subtração.

A maneira de se realizar uma subtração utilizando números binários é semelhante àquela com os números decimais. O cálculo é feito normalmente através de simples operações e, quando houver necessidade de "empréstimos", esse será feito na casa imediatamente à esquerda (ou na mais próxima à esquerda cujo dígito for 1), respeitando, obviamente, a ideia de que só temos dois algarismos para utilizar.

É importante salientar que aqui estamos considerando os casos onde minuendo é maior do que o subtraendo, o que gera resultados positivos. Não mostramos casos onde o inverso acontece, ou seja, quando o minuendo é menor do que o subtraendo, pois, se assim for, o resultado será um número negativo, que não é o foco do nosso trabalho, tendo em vista que estamos nos detendo apenas aos números naturais.

Analogamente à adição, na subtração também só existem quatro possibilidades de operações. São elas:

$$0-0=0,$$

$$1-0=1,$$

$$1-1=0 \text{ e}$$

$$0-1=1.$$

Neste último caso, a operação será compreendida como $10-1=1$, já considerando um possível empréstimo de 1.

Exemplo 5. *Vejamos, agora, dois exemplos de subtrações com números binários, onde já vamos colocar da maneira visualmente prática:*

$$\begin{array}{rcccc} & 1 & 1 & 0 & 1 \\ - & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad e \quad \begin{array}{rcccc} & 1 & 1 & 0 & 1 \\ - & 1 & 0 & 1 & 0. \\ \hline \end{array}$$

No primeiro caso, com pequenas e simples subtrações, o cálculo é realizado sem dificuldades, tendo como resultado 101*:*

$$\begin{array}{rcccc} & 1 & 1 & 0 & 1 \\ - & 1 & 0 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1. \end{array}$$

Já para o segundo exemplo, devemos proceder com um pouco mais de atenção, visto que nem todas as operações estão óbvias, como aconteceu no exemplo anterior. Assim, vamos efetuar, individualmente, cada uma das subtrações, respeitando as posições ordinais de cada algarismo, e começando sempre da direita para esquerda.

De início, $1 - 0 = 1$*; depois temos* $0 - 1$*, que não é imediata. É aí que vamos recorrer ao que usualmente chamamos de "pegar emprestado", e esse empréstimo é feito ao algarismo de ordem imediatamente superior, que, nesse caso, é o* 1*, e, cedendo este valor, passará a ser* 0*. Esse valor cedido, que é uma ordem superior a que o* 0 *está, se juntará com o* 0*, tornando-se* 10*. Assim, o* $0 - 1$ *passará a ser visto como* $10 - 1 = 1$*, pois, nos binários, o* 10 *equivale ao* 2*. Prosseguindo, tínhamos* $1 - 0$*, mas esse* 1 *foi cedido à ordem anterior, logo, restou apenas* $0 - 0$*, que é* 0*; por fim, temos* $1 - 1 = 0$*. Ou seja,*

$$\begin{array}{rcccc} & 1 & {}^{0}\not{1} & {}^{1}0 & 1 \\ - & 1 & 0 & 1 & 0 \\ \hline & 0 & 0 & 1 & 1. \end{array}$$

Portanto, $1101 - 1010 = 11$*. Isso significa que* $1310 = 3$*.*

2.4.3 Multiplicação

Na multiplicação, procedemos de forma análoga à que realizamos com os números decimais, começando sempre da direita para esquerda e multiplicando todos os fatores, individualmente, entre si, seguindo os casos mais simples:

$$0 \cdot 0 = 0,$$

$$0 \cdot 1 = 0,$$

$$1 \cdot 0 = 0 \text{ e}$$

$$1 \cdot 1 = 1.$$

Exemplo 6. *Vejamos os seguintes exemplos, já com seus respectivos resultados e escritos na forma conveniente.*

$$\begin{array}{cccc} & & & 1 \\ 1 & 0 & 1 & 0 \\ & & \times & 1 \\ \hline 1 & 0 & 1 & 0 \end{array} \quad e \quad \begin{array}{ccccccc} & & & 1 & 0 & 1 & 0 & 1 \\ & & & & & \times & 1 & 0 \\ \hline & & & 0 & 0 & 0 & 0 & 0 \\ + & 1 & 0 & 1 & 0 & 1 & & \\ \hline & 1 & 0 & 1 & 0 & 1 & 0. \end{array}$$

Notemos que não há necessidade de usar o "vai um", visto que a maior possibilidade de resultado é 1*, proveniente do* $1 \cdot 1$*.*

2.4.4 Divisão

A divisão também terá o mesmo algoritmo que é utilizado na divisão com decimais, onde, de início, no dividendo, escolhe-se, de forma conveniente, a "parte" que seja maior ou igual ao divisor e, em seguida, realiza-se a divisão, sendo que o resultado desta operação irá para o quociente, que deverá ser multiplicado pelo divisor e o resultado subtraído do número "escolhido" na primeira parte da operação, resultando num valor que, posteriormente, será unido ao próximo número que ficou no dividendo sem ter sido utilizado. Este procedimento deverá ser repetido até que o resto seja 0, resultando, assim, uma divisão exata.

Vale salientar que não vamos tratar de casos onde as divisões não são exatas, uma vez que estamos nos detendo somente aos números inteiros positivos.

Exemplo 7. *Vejamos um exemplo prático de como acontece a divisão nos números binários:*

$$\begin{array}{rcccccc|cccc} & 1 & 1 & 1 & 1' & 0' & 0' & 1 & 1 & 0 & 0 \\ \cline{8-11} - & 1 & 1 & 0 & 0 & & & & 1 & 0 & 1 \\ \cline{1-5} & 0 & 0 & 1 & 1 & 0 & & & & & \\ & - & 0 & 0 & 0 & 0 & & & & & \\ \cline{1-5} & 0 & 0 & 1 & 1 & 0 & 0 & & & & \\ & & - & 1 & 1 & 0 & 0 & & & & \\ \cline{1-6} & 0 & 0 & 0 & 0 & 0 & 0 & & & & \end{array}$$

Portanto, o resultado da divisão de 111100 *por* 1100 *é* 101*. No sistema decimal, isso equivale a afirmar que o resultado da divisão de* 60 *por* 12 *é igual a* 5*.*

3 A Álgebra de Boole

Neste capítulo, iremos discursar sobre uma parte da matemática aplicada que é utilizada na programação de computadores e de outros equipamentos eletrônicos, como celulares e videogames. Para isto, será abordado o contexto histórico desta álgebra; em seguida, será apresentada sua formalização matemática, com ênfase nos operadores básicos "ou", "e" e "não", seus axiomas e propriedades; também tratar-se-á comentado sobre expressões e funções booleanas, incluindo o processo de derivação e simplificação de expressões; a questão dos circuitos lógicos; outros operadores lógicos, além dos básicos, também serão abordados; e, por fim, serão mostradas algumas aplicações da Álgebra de Boole.

3.1 Contexto Histórico

Autodidata, George Boole, nascido na Inglaterra no dia 02 de novembro de 1815, foi o responsável pela invenção desse tipo de álgebra.

Mesmo quase sem formação acadêmica, Boole, incentivado pelo seu pai, dedicou-se a estudar línguas e Matemática, tornou-se professor aos 16 anos numa escola em Lincoln e chegou até a abrir sua própria no ano de 1835. Boole dedicou-se também a escrever trabalhos na área, baseando-se em obras de Laplace e de Lagrange para escrita do seu primeiro trabalho, publicado em 1840.

Em 1844, foi condecorado com a medalha de ouro *Royal Society* por seu trabalho sobre métodos algébricos para a solução de equações diferenciais, publicado no *Journal of Mathematics of Cambrige*. Em 1847, publicou uma obra curta chamada *The Mathematical Analysis of Logic*, onde defendia que a lógica deveria estar associada à Matemática numa visão mais ampla, visto que, até então, essa era tida apenas como a ciência das grandezas e dos números. Ainda nesta obra, Boole introduz os conceitos de lógca simbólica, apresentando-os como traduções para equações algébricas.

Foi nomeado professor de Matemática no *Queens College* em 1849, na Irlanda, onde permaneceu até o final da sua vida.

Sua principal obra foi publicada no ano de 1854, intitulada *Investigation of the Laws of thought*. Nela, Boole amplia e esclarece as ideias apresentadas em 1847, se fundamentando, simultaneamente, numa lógica formal e na nova álgebra, denominada Álgebra de Boole, Álgebra Booleana, Álgebra Proposicional ou, ainda, Álgebra da Lógica. Nesta obra,

> Boole usou as letras x, y e z para representar subconjuntos de coisas, números, pontos, ideias, ou outras entidades escolhidas de um con-

> junto universal ou universo de discurso, cuja totalidade ele designava pelo símbolo ou número; e mostrou que sua álgebra fornecia um algoritmo simples para raciocínios silogísticos. (MELLO, 2010, p. 181–182)

Nesse sentido, Boole defendia a ideia de que os processos de raciocínio utilizados na vida cotidiana poderiam ser representados através de termos de lógica matemática, ou seja, ele acreditava que era possível exprimir o raciocínio humano em termos matemáticos. Por este motivo, a álgebra de Boole também é conhecida como a Álgebra do Pensamento. Com isso, Boole, mais tarde, tornar-se-ia um dos gênios que dariam sua contribuição para a invenção dos computadores.

Em 1855, casou-se com Mary Everest, com quem teve cinco filhas. Em 1857, tornou-se membro da *Royal Society*. Boole também recebeu títulos das Universidades de Oxford e de Dublin.

Além disso, o matemático publicou outros trabalhos, tais como um tratado sobre equações diferenciais e um tratado em cálculo de diferenças finitas. Também estudou métodos gerais de probabilidades e foi um dos pioneiros na investigação das propriedades básicas dos números.

Sendo considerado um gênio, sua ideia central com a álgebra booleana era formular a linguagem do pensamento através de símbolos. O próprio autor, em uma das suas obras, escreveu que pretendia estabelecer o cálculo da lógica através do uso de símbolos, cujas leis de combinação são conhecidas e gerais, onde os resultados admitem uma interpretação consistente. (BOOLE, 1847) Assim, "Boole viu a lógica de um modo novo e chegou a uma álgebra mais simples". (RIZZATO; RINALDI, 2005)

Segundo (MELLO, 2010, p. 182),

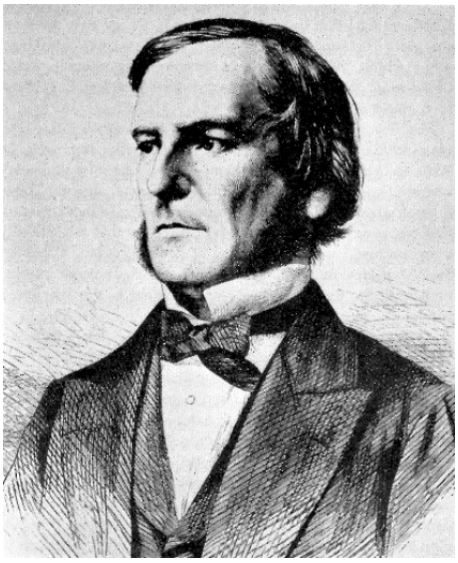
> a álgebra booleana é largamente usada não só por matemáticos puros, mas também por outros que a aplicam a problemas de teoria da informação. As notações mudaram um pouco desde os dias de Boole, mas os princípios fundamentais são os estabelecidos por ele há mais de um século.

George Boole teve sua carreira encerrada muito cedo, quando em 1864, aos 49 anos de idade, veio a falecer na Irlanda, vítima de pneumonia. Antes disso, ele disse a um amigo que a lógica booleana poderia ser a contribuição mais valiosa, senão a única, que ele havia feito ou que, provavelmente, faria à ciência, e era o motivo pelo qual desejaria ser lembrado postumamente (NEWS, 2015). E assim aconteceu, pois, sem suas ideias, os computadores modernos não teriam as características que tem hoje.

3.2 A Álgebra Formal

Diferentemente da álgebra tradicional, onde as variáveis podem assumir um infinidade de valores, na álgebra de Boole há apenas dois possíveis valores para as vari-

Figura 2 – George Boole (1815 – 1864)



Fonte: https://www.techtudo.com.br/artigos/noticia/2012/08/uma-algebra-diferente.html

áveis: 0 e 1. Logo, a base utilizada neste tipo de álgebra é a binária.

A álgebra booleana pode ser definida como uma estrutura matemática que incorpora as propriedades básicas do cálculo proposicional e da teoria dos conjuntos, ou seja, é um outro modelo de uma mesma estrutura matemática. (ABAR, 2004)

A linguagem de programação utilizada pelos computadores é composta unicamente por números, mais especificamente, pelos número 0 e 1, que são os números binários. No entanto, estes valores não exprimem quantidades, mas apenas, e só, estados do sistema. Nesse sentido, o que esses números transmitem para as placas dos computadores são pulsos elétricos (1) ou a ausência deles (0), o que pode ser interpretado como uma chave do tipo liga/desliga, ou ideias como o sim ou o não, o verdadeiro ou o falso, o tudo ou o nada. Teoricamente, o 1 é como se fosse o universo e o 0, o nada.

Assim, a álgebra de Boole, ou álgebra booleana,

> é uma estrutura algébrica que esquematiza as operações lógicas, e está presente em todas as partes, desde a programação por trás dos videogames até o código dos aplicativos que usamos. Pode-se dizer que os tijolos que formam a programação, que são os comandos ou instruções dadas a um sistema informático, são todos baseados na lógica de Boole. (NEWS, 2015, p.01)

(BARANAUSKAS, 2012) nos traz algumas importantes definições quando diz que existem apenas duas constantes booleanas 0(zero) e 1(um) e que as variáveis booleanas são representadas por letras e podem assumir apenas dois valores lógicos, também 0 ou 1. O autor ainda acrescenta que uma expressão booleana é uma expressão matemática envolvendo constantes e/ou variáveis booleanas e seu resultado só pode assumir dois valores: 0 ou 1.

(GÜNTZEL; NASCIMENTO, 2001, p. 01) afirmam que,

> como o número de valores que cada variável pode assumir é finito (e pequeno), o número de estados que uma função booleana pode assumir também será finito, o que significa que podemos descrever completamente as funções booleanas utilizando tabelas. Devido a este fato, uma tabela que descreva uma função booleana recebe o nome de tabela verdade, e nela são listadas todas as combinações de valores que as variáveis de entrada podem assumir e os correspondentes valores da função (saídas).

(VIEIRA, 2000) ressalta que, como uma função de n variáveis possui apenas 2 conjuntos possíveis de valores de entrada, a função pode ser descrita completamente através de uma tabela de 2^n linhas, onde cada linha mostra o valor da função para uma combinação diferente dos valores de entrada.

Nesse sentido, a partir do uso desta tabela, é possível definir o valor lógico de uma proposição booleana, isto é, saber se esta é verdadeira ou falsa, quando esta vale 1 ou 0, respectivamente. No decorrer deste trabalho, iremos utilizá-la em várias situações para encontrar valores lógicos.

Em sua teoria, Boole utilizou o conceito de operadores, ou perguntas, que traduzem um enunciado. Há três operadores básicos, que, quando combinados, podem traduzir todos os enunciados booleanos, inclusivem os mais complexos. São eles: "ou", "e" e "não", que são equivalentes a "or", "and" e "not", respectivamente, no inglês, língua materna de Boole.

Veremos, agora, como funcionam estes operadores básicos.

3.2.1 Operador *OR* (OU)

O operador "ou", cuja escrita original é "*or*", também pode ser chamada de adição lógica. (GÜNTZEL; NASCIMENTO, 2001) afirmam que, como uma expressão booleana ou vale 0 ou vale 1, basta que esta seja definida quando a operação valer 1, pois, automaticamente, a operação resultará 0 nos demais casos. Nesse contexto, por definição, esta operação resulta 1 se pelo menos uma das constantes de entrada for 1, isto é, a operação resulta 0 somente quando todas elas forem 0.

O símbolo mais utilizado para representar esta operação é $+$, o mesmo que é empregado nas adições algébricas. Vale destacar, por outro lado, que a adição em questão não é a algébrica, mas a lógica. Em outras biografias também é muito encontrado o símbolo $\vee$ para representá-la.

Listando todas as possíveis combinações para soma lógica boolenana, tem-se:

$$0 + 0 = 0,$$

$$0 + 1 = 1,$$

$$1 + 0 = 1 \text{ e}$$

$$1 + 1 = 1.$$

Como o operador "ou" é binário, é necessário que haja pelo menos duas variáveis envolvidas, não sendo possível realizar uma adição se houver apenas uma variável.

Em geral, nas expressões não se faz uso de todas as possibilidades de valores, e, sim, faz-se uso de letras para designar as variáveis. Então, supondo a análise do comportamento da expressão $A + B$ (lê-se "A ou B"), fazendo uso de uma tabela verdade, temos:

Tabela 1 – Soma lógica com duas variáveis

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Fonte: Autora

Analogamente aconteceria se houvesse mais de duas variáveis. Vejamos, por exemplo, uma tabela verdade para 3 variáveis, digamos $A + B + C$, pois $A + B + C = (A + B) + C$:

Tabela 2 – Soma lógica com três variáveis

A	B	C	A+B+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

Fonte: Autora

Assim, para 2, 3 ou, generalizando, para n variáveis, poderia ser aplicada diretamente a definição do operador "ou", que diz que o valor de saída será 1 se pelo menos um dos valores de entrada for 1, sendo 0 apenas no caso em que todos eles forem iguais a 0.

3.2.2 Operador *AND* (E)

O operador originalmente escrito *and*, cuja tradução é "e", também pode ser chamado de multiplicação lógica. (GÜNTZEL; NASCIMENTO, 2001, p.3) afirmam que a operação "e" resulta 0 se pelo menos uma das constantes de entrada for 0. Assim, pela definição dada, o resultado de uma operação "e", ou seja, de uma multiplicação lógica, será 1 se, e somente se, todas as entradas forem 1.

O símbolo $\cdot$, muito utilizado na multiplicação algébrica, também é usado na multiplicação lógica para representar o operador "e". Também é muito comum encontrar o símbolo $\wedge$ como outra notação.

Listanto as possíveis combinações para a multiplicação lógica envolvendo valores booleanos, temos:

$$0 \cdot 0 = 0,$$

$$0 \cdot 1 = 0,$$

$$1 \cdot 0 = 0 \text{ e}$$

$$1 \cdot 1 = 1.$$

Análogo ao operador "ou", o operador "e" também é binário, sendo necessário, portanto, a existência de pelo menos duas variáveis de entrada para que a operação possa ser realizada.

A tabela verdade que mostra o comportamento de $A \cdot B$ (lê-se "A e B"), é a seguinte:

Tabela 3 – Multiplicação lógica com duas variáveis

A	B	A $\cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Fonte: Autora

Para três variáveis de entrada, a ideia seria semelhante, uma vez que $A \cdot B \cdot C = (A \cdot B) \cdot C$:

Tabela 4 – Multiplicação lógica com três variáveis

A	B	C	A $\cdot B \cdot C$
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

Fonte: Autora

Portanto, com duas ou três variáveis, vimos que o resultado é 1 unicamente quando todas as constantes de entrada são 1. Generalizando para n variáveis, o resultado será 1 se, e somente se, todas as constantes forem 1, ou seja, se pelo menos uma das constantes for 0, o resultado de saída também será 0.

3.2.3 Operador Complemento

O operador complemento, que também pode ser chamado de negação ou inversão, tem a função de valor complementar ou valor inverso ao que a variável original apresenta (GÜNTZEL; NASCIMENTO, 2001). Então, como as constantes booleanas podem apenas assumir os valores 0 e 1, a operação complementar aplicada a estes valores resultam em 1 e 0, respectivamente.

Os símbolos mais utilizados para representar esta operação em relação, por exemplo, a uma variável A são $\sim A$, A' ou ainda uma barra horizontal acima da variável (lê-se "A negado" ou "não A").

Listanto os possíveis resultados desta operação, temos:

$$0' = 1 \text{ e}$$

$$1' = 0.$$

Diferentemente do que acontece nas operações "ou" e "e", a operação complementação, para estar definida, há necessidade de apenas uma variável de entrada, sendo, portanto, uma operação unitária.

A tabela verdade, nesse caso, é:

Tabela 5 – Negação lógica com uma variável

A	A'
0	1
1	0

Fonte: Autora

3.3 Definições Básicas e Propriedades

Diante do exposto, por álgebra booleana entende-se um conjunto $A = \{a, b, c, \ldots\}$ junto com duas operações binárias, $+$ (também denotada por $\vee$ e chamada "ou") e $\cdot$ (também denotada $\wedge$ e conhecida por "e"), uma operação unitária $'$ (também escrita como $\backsim$ ou uma barra superior acima da variável, e chamada de operação complemento, inversão ou negação) e dois únicos elementos 0 e 1, também chamados de zero ou falso ou de um ou verdadeiro, respectivamente, se, e somente se, satisfazem os seguintes postulados (Postulados de *Huntington*): (OLIVEIRA, 2017)

Axioma 3.1. *As operações $+$ e $\cdot$ são comutativas. Isto é, para todo a e b em A,*

$$a + b = b + a \quad e \quad a \cdot b = b \cdot a$$

Axioma 3.2. *Cada operação é distributiva sobre a outra. Ou seja, para todo a, b e c em A,*

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad e \quad a + (b \cdot c) = (a + b) \cdot (a + c)$$

Axioma 3.3. *Existem em A elementos identidades 0 e 1, distintos, com relação as operações $+$ e $\cdot$, respectivamente. Isto é, para todo a em A,*

$$a + 0 = a \quad e \quad a \cdot 1 = a$$

Axioma 3.4. *Há um elemento a', denominado complemento de a, tal que, para cada elemento a de A, existe um elemento a', também em A, tal que*

$$a + a' = 1 \quad e \quad a \cdot a' = 0$$

Assim, a álgebra booleana pode ser definida como uma sextupla ordenada $\langle A, +, \cdot, ', 0, 1\rangle$, onde são satisfeitos os axiomas referidos anteriormente.

> Alguns autores incorporam outros axiomas como parte da definição de uma álgebra booleana. Vale registrar que os postulados de *Huntington* correspondem a um conjunto minimal de postulados, isto é, nenhum deles pode ser derivado a partir dos demais. Mais ainda, é um conjunto completo no sentido de que qualquer propriedade de uma álgebra Booleana pode ser derivada/provada a partir desses postulados. (HIRATA, 2005, p. 30)

Os axiomas 3.1, 3.3 e 3.4 são aceitos por definição. Já o 3.2 não é tão óbvio, por isso vamos provar a veracidade dele através de tabelas-verdade. Iremos fazer o mesmo com os demais axiomas.

Antes, porém, é importante acrescentar o que diz (LIPSCHUTZ; LIPSON, 1976), quando afirmam que devemos, em geral, adotar que a negação tem precedência sob a multiplicação, e que a multiplicação tem precedência em relação a adição, a menos que as operações estejam entre parenteses, que tem prioridade sobre os demais operadores.

Por exemplo, $a + b \cdot c$ significa $a + (b \cdot c)$ e não $(a + b) \cdot c$, e $a \cdot b'$ representa $a \cdot (b')$ e não $(a \cdot b)'$.

Analogamente à álgebra nos reais, quando $a + b \cdot c$ é escrito como $a + bc$, há uma equivalência entre as expressões.

Agora, vejamos, através de tabelas-verdade, a validade dos axiomas apresentados, onde será destacado com um $*$ as colunas que deverão ter os mesmos valores de saída, representando a igualdade das expressões.

- Axioma 3.1 :

Tabela 6 – Tabelas verdades do axioma da comutatividade

a + b = b + a

a	b	a+b	b+a
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1
		*	*

e

a · b = b · a

a	b	a ·*b*	b ·*a*
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1
		*	*

Fonte: Autora

- Axioma 3.2 :

Tabela 7 – Tabelas verdades do axioma da distributividade

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

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
				*			*

e

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

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
				*			*

Fonte: Autora

- Axioma 3.3 :

Tabela 8 – Tabelas verdades dos elementos identidade

a + 0 = a

a	0	a+0
0	0	0
1	0	1
*		*

e

a $\cdot 1 = a$

a	1	a $\cdot 1$
0	1	0
1	1	1
*		*

Fonte: Autora

- Axioma 3.4 :

Tabela 9 – Tabelas verdades do complemento

$a + a' = 1$

a	a'	a + a'	1
0	1	1	1
1	0	1	1
		*	*

e

a · a' = 0

a	a'	a $\cdot a'$	0
0	1	0	0
1	0	0	0
		*	*

Fonte: Autora

Vejamos alguns exemplos de como essa álgebra é utilizada.

Exemplo 8. *O conjunto dos bits (dígitos binários)* $A = \{0, 1\}$ *com as operações binárias* $+$ *e* $\cdot$ *e a operação unitária* $'$ *definidas da maneira usual é uma álgebra booleana, isto é,*

$$\begin{array}{lll} 0+0=0 & 0\cdot 0=0 & \\ 0+1=1 & 0\cdot 1=0 & 0'=1 \\ 1+0=1 & 1\cdot 0=0 & 1'=0 \\ 1+1=1 & 1\cdot 1=1 & \end{array}$$

Demonstração. A demonstração da validade dos axiomas neste exemplo encontra-se nas tabelas verdades já apresentadas. ■

Exemplo 9. *Seja* $A = \{1, 2, 5, 7, 10, 14, 35, 70\}$*, ou seja, o conjunto de divisores de* 70*. As operações binárias* $+$ *e* $\cdot$ *e a operação unitária* a' *definidas como segue, para quaisquer* $a_1, a_2 \in A$*,*

- $a_1 + a_2$ *: o mínimo múltiplo comum entre* a_1 *e* a_2*;*
- $a_1 \cdot a_2$*: o maior divisor comum entre* a_1 *e* a_2*; e*
- $a' = \dfrac{70}{a}$

Temos que A *é uma álgebra booleana, com* 1 *o elemento zero e* 70 *o elemento unitário.*

Demonstração. O Axioma 3.1 segue da propriedade já conhecida que diz que $\text{mmc} = (a, b) = \text{mmc}\,(b, a)$ e $\text{mdc} = (a, b) = \text{mdc}\,(b, a)$.

Para o Axioma 3.2, faremos uso de um importante resultado, que também será demonstrado.

Afirmação 3.1. $\text{mdc}\,(a, \text{mmc}\,(b, c)) = \text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c))$.

Demonstração. Sejam $a, b, c \in \mathbb{N}$ e $p_1, p_2, \ldots, p_n$ números primos tais que

$$a = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k} \ldots p_n^{\alpha_n},$$

$$b = p_1^{\beta_1} p_2^{\beta_2} \ldots p_k^{\beta_k} \ldots p_n^{\beta_n},$$

$$c = p_1^{\gamma_1} p_2^{\gamma_2} \ldots p_k^{\gamma_k} \ldots p_n^{\gamma_n},$$

onde $\alpha_i, \beta_i, \gamma_i \in \mathbb{N}$ para todo i. Não há perda de generalidade em supor os mesmos primos em cada decomposição, uma vez que podemos ter expoentes nulos. Lembrando agora que

$$\text{mdc}\,(a, b) = p_1^{min(\alpha_1,\beta_1)} p_2^{min(\alpha_2,\beta_2)} \ldots p_k^{min(\alpha_k,\beta_k)} \ldots p_n^{min(\alpha_n,\beta_n)}$$

e

$$\text{mmc}\,(a, b) = p_1^{max(\alpha_1,\beta_1)} p_2^{max(\alpha_2,\beta_2)} \ldots p_k^{max(\alpha_k,\beta_k)} \ldots p_n^{max(\alpha_n,\beta_n)}$$

vamos determinar a potência de p_k em $\text{mdc}\,(a, \text{mmc}\,(b, c))$ e também em $\text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c))$.

Temos 6 casos a considerar:

- Caso 1: $\alpha_k \leq \beta_k \leq \gamma_k$
- Caso 2: $\alpha_k \leq \gamma_k \leq \beta_k$
- Caso 3: $\beta_k \leq \alpha_k \leq \gamma_k$
- Caso 4: $\beta_k \leq \gamma_k \leq \alpha_k$
- Caso 5: $\gamma_k \leq \alpha_k \leq \beta_k$
- Caso 6: $\gamma_k \leq \beta_k \leq \alpha_k$

No caso 1, $\alpha_k \leq \beta_k \leq \gamma_k$ e daí a potência de p_k em $\text{mmc}\,(b, c)$ é $p_k^{max(\beta_k,\gamma_k)} = p_k^{\gamma_k}$. Portanto, a potência de p_k em $\text{mdc}\,(a, \text{mmc}\,(b, c))$ é igual a $p_k^{min(\alpha_k,\gamma_k)} = p_k^{\alpha_k}$.

Considerando ainda o caso 1, as potências de p_k em $\text{mdc}\,(a, b)$ e em $\text{mdc}\,(a, c)$ são ambas iguais a p^{α_k}. Logo, a potência de p_k em $\text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c))$ é dada por $\text{mmc}\,(p_k^{\alpha_k}, p_k^{\alpha_k}) = p_k^{\alpha_k}$.

O mesmo vale para os outros casos $2, 3, \cdots, 6$: a potência de p_k em $\text{mdc}\,(a, \text{mmc}\,(b, c))$ coincide com a de $\text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c))$.

Portanto, as potências de p_k em $\text{mdc}\,(a, \text{mmc}\,(b, c))$ e em $\text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c))$ coincidem para todo $k = 1, 2, \cdots, n$. Concluímos, dessa forma, que

$$\text{mdc}\,(a, \text{mmc}\,(b, c)) = \text{mmc}\,(\text{mdc}\,(a, b), \text{mdc}\,(a, c)).$$

■

Assim, temos

$$a\cdot(b+c) = \text{mdc}\,(a, b+c) = \text{mdc}\,(a, \text{mmc}\,(b,c)) = \text{mmc}\,(\text{mdc}\,(a,b)\,, \text{mdc}\,(a,c)) = a\cdot b + a\cdot c$$

e

$$a + (b\cdot c) = \text{mmc}\,(a, b\cdot c) = \text{mdc}\,(\text{mmc}\,(a,b)\,, \text{mmc}\,(a,c)) = (a+b)\cdot(a+c)\,.$$

O Axioma 3.3 segue de que 1 é o elemento zero e 70, o elemento unitário. Assim,

$$a + 0 = \text{mmc}\,(a,0) = \text{mmc}\,(a,1) = a, \text{ pois } a = 1\cdot a = a\cdot 1$$

e

$$a\cdot 1 = \text{mdc}\,(a,1) = \text{mdc}\,(a,70) = a \text{ pois } a \in A, A \text{ conjunto dos divisores de } 70.$$

O Axioma 3.4 segue do fato que $a' = \dfrac{70}{a}$, logo:

$$a + a' = \text{mmc}\,(a,a') = \text{mmc}\left(a, \frac{70}{a}\right) = 70 = 1$$

e

$$a\cdot a' = \text{mdc}\,(a,a') = \text{mdc}\left(a, \frac{70}{a}\right) = 1 = 0.$$

■

3.3.1 Propriedades Fundamentais da Álgebra de Boole

1. **Princípio de dualidade**

   Cada expressão ou identidade algébrica dedutível a partir dos postulados de uma álgebra booleana continua válida se todas as ocorrências dos operadores + e · e os elementos identidade 0 e 1 são trocados um pelo outro. (HIRATA, 2005, p. 32)

   O termo utilizado para designar a nova expressão booleana após as trocas correspondentes é "dual". Por exemplo, o dual de $(1+a)\cdot(b+0) = b$ é $(0\cdot a)+(b\cdot 1) = b$.

   Nos axiomas já apresentados, o dual de cada um deles também é um axioma, tendo em vista a simetria existente entre eles, ou seja, no axioma 3.2, temos $a\cdot(b+c) = (a\cdot b)+(a\cdot c)$, cujo dual é $a+(b\cdot c) = (a+b)\cdot(a+c)$, onde este último continua válido, como visto, anteriomente, na tabela 7.

   Assim, de uma forma geral,

> O dual de qualquer teorema em uma álgebra booleana também é um teorema. Em outras palavras, se qualquer afirmação é uma consequência dos axiomas de uma Álgebra Booleana, então o dual também é uma consequência desses axiomas, uma vez que a afirmação dual pode ser provada usando o dual de cada etapa da prova da expressão original. (LIPSCHUTZ; LIPSON, 1976, p. 369)

2. **Unicidade do 0 e do 1**

   Os elementos 0 (zero) e 1 (um) são únicos.

   Demonstração. Sejam a_1 e a_2 elementos quaisquer em A. Supondo, por absurdo, que existam dois elementos zeros, distintos, digamos 0_1 e 0_2, pelo axioma 3.3,

   $$a_1 + 0_1 = a_1 \qquad \text{e} \qquad a_2 + 0_2 = a_2.$$

   Sem perda de generalidade, podemos tomar $a_1 = 0_2$ e $a_2 = 0_1$. Assim,

   $$0_2 + 0_1 = 0_2 \qquad \text{e} \qquad 0_1 + 0_2 = 0_1.$$

   Pelo Axioma 3.1 e a transtividade de igualdades, resulta que $0_1 = 0_2$, o que mostra a unicidade do 0.

   Para a unicidade do 1, podemos considerá-la como a dual do 0, ou seja, a demonstração desta decorre imediatamente do princípio da dualidade ■

3. **Idempotência**

   Para todo elemento $a \in A$, tem-se que $a + a = a$ e $a \cdot a = a$.

   Demonstração. Para $a + a = a$, temos

   $$\begin{aligned} a + a &= (a + a) \cdot 1 && \text{(Axioma 3.3)} \\ &= (a + a) \cdot (a + a') && \text{(Axioma 3.4)} \\ &= a + (a \cdot a') && \text{(Axioma 3.2)} \\ &= a + 0 && \text{(Axioma 3.4)} \\ &= a && \text{(Axioma 3.3).} \end{aligned}$$

   Para $a \cdot a = a$, basta tomar o dual do item anterior. ■

4. **Identidade**

   Para todo $a \in A$, tem-se que $a + 1 = 1$ e $a \cdot 0 = 0$.

   Demonstração. Para $a + 1 = 1$, temos

$$\begin{aligned} a+1 &= 1\cdot(a+1) && \text{(Axioma 3.3)}\\ &= (a+a')\,(a+1) && \text{(Axioma 3.4)}\\ &= a+(a'\cdot 1) && \text{(Axioma 3.2)}\\ &= a+a' && \text{(Axioma 3.3)}\\ &= 1 && \text{(Axioma 3.4).} \end{aligned}$$

Para $a\cdot 0=0$, basta tomar o dual do item anterior. ■

5. **Absorção**

   Para quaisquer a, $b\in A$, tem-se que $a+(a\cdot b)=a$ e $a\cdot(a+b)=a$.

   Demonstração. Para $a+(a\cdot b)=a$, temos

$$\begin{aligned} a+(a\cdot b) &= a\cdot 1+a\cdot b && \text{(Axioma 3.3)}\\ &= a\,(1+b) && \text{(Axioma 3.2)}\\ &= a\cdot 1 && \text{(Propriedade 4)}\\ &= a && \text{(Axioma 3.3).} \end{aligned}$$

   Para $a\,(a+b)=a$, basta tomar o dual do item anterior. ■

6. **Associatividade**

   Para quaisquer a, b, $c\in A$, tem-se que $a+(b+c)=(a+b)+c$ e $a\cdot(b\cdot c)=(a\cdot b)\cdot c$.

   Demonstração. Antes de provar as propriedades associativas, é necessário enunciar a seguinte afirmação (que também será demonstrada):

   Afirmação 3.2. *Para quaisquer a, b, $c\in A$, $a\,[(a+b)+c]=[(a+b)+c]\,a=a$.*

   Demonstração. Com efeito,

$$\begin{aligned} a\,[(a+b)+c] &= [(a+b)+c]\,a && \text{(Axioma 3.1)}\\ &= a\,(a+b)+ac && \text{(Axioma 3.2)}\\ &= a+ac && \text{(Propriedade 5)}\\ &= a && \text{(Propriedade 5).} \end{aligned}$$

   ■

   Agora, utilizando a afirmação acima, provaremos a associatividade na álgebra booleana. De fato,

$$\begin{aligned}
Z &= [(a+b)+c]\,[a+(b+c)] \\
&= [(a+b)+c]\,a + [(a+b)+c]\,(b+c) && \text{(Axioma 3.2)} \\
&= a + [(a+b)+c]\,(b+c) && \text{(Afirmação 3.2)} \\
&= a + [(a+b)+c]\,b + [(a+b)+c]\,c && \text{(Axioma 3.2)} \\
&= a + [(b+a)+c]\,b + [(a+b)+c]\,c && \text{(Axioma 3.1)} \\
&= a + b + [(a+b)+c]\,c && \text{(Afirmação 3.2 e Propriedade 5)} \\
&= a + (b+c)\,.
\end{aligned}$$

De maneira análoga,

$$\begin{aligned}
Z &= [(a+b)+c]\,[a+(b+c)] \\
Z &= (a+b)\,[a+(b+c)] + c\,[a+(b+c)] && \text{(Axioma 3.2)} \\
&= (a+b)\,[a+(b+c)] + c && \text{(Afirmação 3.2)} \\
&= a\,[a+(b+c)] + b\,[a+(b+c)] + c && \text{(Axioma 3.2)} \\
&= a\,[a+(b+c)] + b + c && \text{(Afirmação 3.2 e Propriedade 5)} \\
&= (a+b) + c.
\end{aligned}$$

Portanto, $a + (b + c) = (a + b) + c$.

A demonstração de $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ segue pelo dual da anterior. ■

7. **Complemento do 0 e do 1**

   Tem-se $0' = 1$ e $1' = 0$.

   Demonstração. Temos

$$\begin{aligned}
0' &= 0' + 0 && \text{(Axioma 3.3)} \\
&= 1 && \text{(Axioma 3.4).}
\end{aligned}$$

   Para $1' = 0$, basta tomar o dual do item anterior. ■

8. **Unicidade do complemento**

   O inverso de qualquer elemento $a \in A$ é único, isto é, se $a + x = 1$ e $a \cdot x = 0$ para algum $x \in A$, então $x = a'$.

   Demonstração. Por absurdo, suponhamos que há dois elementos a'_1 e a'_2 em A, distintos, tais que

$$a + a'_1 = 1; \qquad a + a'_2 = 1; \qquad a \cdot a'_1 = 0 \qquad \text{e} \qquad a \cdot a'_2 = 0.$$

Assim, temos

$$\begin{aligned} a'_1 &= 1 \cdot a'_1 && \text{(Axioma 3.3)} \\ &= (a + a'_2) \cdot a'_1 && \text{(Hipótese)} \\ &= a \cdot a'_1 + a'_2 \cdot a'_1 && \text{(Axioma 3.3)} \\ &= 0 + a'_2 \cdot a'_1 && \text{(Axioma 3.4)} \\ &= a'_2 \cdot a'_1 && \text{(Axioma 3.3).} \end{aligned}$$

Analogamente,

$$\begin{aligned} a'_2 &= 1 \cdot a'_2 && \text{(Axioma 3.3)} \\ &= (a + a'_1) \cdot a'_2 && \text{(Hipótese)} \\ &= a \cdot a'_2 + a'_1 \cdot a'_2 && \text{(Axioma 3.3)} \\ &= 0 + a'_1 \cdot a'_2 && \text{(Axioma 3.4)} \\ &= a'_1 \cdot a'_2 && \text{(Axioma 3.3).} \end{aligned}$$

Portanto, pelo Axioma 3.1, segue que $a'_1 = a'_2$.

■

9. **Involução**

Para todo $a \in A$, $(a')' = a$.

Demonstração. Seja $(a')' = b$. Temos, pelo axioma 3.4,

$$a' \cdot b = 0 \quad \text{e} \quad \text{e } a' + b = 1.$$

Ainda, por 3.4,

$$a' \cdot a = 0 \quad \text{e} \quad \text{e } a' + a = 1.$$

Devido a unicidade do complemento, $(a')' = b = a$. ■

10. **Teorema de DeMorgan**

(GÜNTZEL; NASCIMENTO, 2001, p. 2) afirmam que "o primeiro teorema de DeMorgan diz que a complementação de um produto (lógico) equivale a soma (lógica) das negações de cada variável do referido produto". Em termos matemáticos, para quaisquer elementos $a_1, a_2, a_3, \ldots \in A$,

$$(a_1 \cdot a_2 \cdot a_3 \cdot \cdots)' = a'_1 + a'_2 + a'_3 + \cdots$$

Analogamente, "o segundo teorema é o dual (i.e., o espelho) do primeiro, ou seja, a complementação de uma soma (lógica) equivale ao produto das negações individuais das variáveis" (GÜNTZEL; NASCIMENTO, 2001, p. 2). Isto é,

$$(a_1 + a_2 + a_3 + \cdots)' = a_1' \cdot a_2' \cdot a_3' \cdot \cdots$$

O resultado geral segue por indução, para isto, basta motrarmos que o resultado vale para a_1 e a_2. Particularizando para duas incógnitas, temos que, para quaisquer $a, b \in A$,

$$(a+b)' = a' \cdot b' \qquad \text{e} \qquad (a \cdot b)' = a' + b'$$

Demonstração. Antes de provar o Teorema, vamos anunciar e demonstrar duas afirmações:

Afirmação 3.3. $(a+b) + a' \cdot b' = 1$

Demonstração. Com efeito,

$$\begin{array}{rcll}
(a+b) + a' \cdot b' & = & [(a+b) + a'] \cdot [(a+b) + b'] & \text{(Axioma 3.2)} \\
& = & [a' + (a+b)] \cdot [b' + (a+b)] & \text{(Axioma 3.1)} \\
& = & [(a'+a) + b] \cdot [a + (b'+b)] & \text{(Propriedade 6 e Axioma 3.1)} \\
& = & 1 \cdot 1 & \text{(Axioma 3.4 e Propriedade 4)} \\
& = & 1 & \text{(Axioma 3.3).}
\end{array}$$

■

Afirmação 3.4. $(a+b) \cdot a' \cdot b' = 0$

Demonstração. De fato,

$$\begin{array}{rcll}
(a+b) \cdot a' \cdot b' & = & a \cdot (a' \cdot b') + b \cdot (b' \cdot a') & \text{(Axiomas 3.1 e 3.2)} \\
& = & (a \cdot a') \cdot b' + (b \cdot b') \cdot a' & \text{(Propriedade 6)} \\
& = & 0 + 0 & \text{(Axioma 3.4 e Propriedade 4)} \\
& = & 0 & \text{(Axioma 3.3).}
\end{array}$$

■

Logo, pela unicidade do complemento, conclui-se que $(a+b)' = a' \cdot b'$.

A demonstração da igualdade $(a \cdot b)' = a' + b'$, dual de $(a+b)' = a' \cdot b'$, pode ser demonstrada através do princípio da dualidade. Portanto, tem-se $(a \cdot b)' = a' + b'$. ■

11. **Expressões elementares**

    Além das propriedades, leis e teoremas apresentados até aqui, ainda existem outras expressões que, por vezes, são utilizadas. São elas:

    a) $a' + a \cdot b' = a' + b'$

    b) $a' + a \cdot b = a' + b$

    c) $a + a' \cdot b = a + b$

    d) $a + a' \cdot b' = a + b'$

    e) $(a + b)(a + c) = a + b \cdot c$

    A validade dessas expressões pode ser comprovada a partir de manipulações algébricas com as propriedades já anunciadas e demonstradas. No entanto, aqui omitiremos tais demonstrações.

    É importante salientar que as demonstrações, aqui realizadas na forma algébrica, também poderiam ter sido realizadas através de tabelas verdades e/ou de Diagramas de Venn.

3.4 Expressões e Funções Booleanas

3.4.1 Expressões Booleanas

(LIPSCHUTZ; LIPSON, 1976) dizem que, ao considerar um conjunto de variáveis (letras e/ou símbolos), digamos $a_1, a_2, \cdots, a_n$, uma expressão booleana nestas variáveis é definida como qualquer variável ou qualquer expressão construída a partir dessas, utilizando as operações $+$, $\cdot$ ou $'$. Obviamente, as expressões booleanas devem ser bem formadas, isto é, as operações $+$ e $\cdot$ são binárias, enquando a operação $'$ é unitária.

Por exemplo,

$$(a + b' \cdot c)' + (a \cdot b \cdot c' + a' \cdot b)' \qquad \text{e} \qquad \Big[(a \cdot b' \cdot c + b)' + a' \cdot c\Big]'$$

são expressões booleanas nas variáveis a, b e c.

(GÜNTZEL; NASCIMENTO, 2001, p. 13) afirmam que "um literal é uma variável negada ou um variável não negada". Assim, por exemplo, o literal pode ser a variável booleana a ou o seu complemento a'.

(LIPSCHUTZ; LIPSON, 1976) definem um produto fundamental como um produto de dois ou mais literais, em que dois dos quais não envolvem a mesma variável. Assim,

$$a \cdot b', \quad a \cdot b'c, \quad a, \quad b' \quad \text{e} \quad a \cdot' b \cdot c$$

são exemplos de produtos fundamentais, enquanto $a \cdot b \cdot a' \cdot c$ e $a \cdot b \cdot c \cdot b$ não são.

No entanto, qualquer produto que envolva literais pode ser reduzido a 0 ou a um produto fundamental, utilizando as propriedades da álgebra booleana. Isso justifica-se com base no Axioma 3.4 e na Propriedade 3, pois, caso exista um produto da mesma variável na forma negada e não negada, recorremos ao Axioma 3.4, que resultará aquele produto em 0, e, se existir um produto de uma variável na mesma forma, recorremos a propriedade de Idempotência (propriedade 3), que nos resultará apenas em uma variável, resultando, assim, num produto fundamental.

Vejamos os exemplos a seguir:

Exemplo 10. $a \cdot b \cdot a' \cdot c$

$$\begin{aligned} a \cdot b \cdot a' \cdot c &= a \cdot a' \cdot b \cdot c && \textit{(Axioma 3.1)} \\ &= (a \cdot a') \cdot b \cdot c && \textit{(Propriedade 6)} \\ &= 0 \cdot b \cdot c && \textit{(Axioma 3.4)} \\ &= 0 && \textit{(Propriedade 4)} \end{aligned}$$

Exemplo 11. $a \cdot b \cdot c \cdot b$

$$\begin{aligned} a \cdot b \cdot c \cdot b &= a \cdot b \cdot b \cdot c && \textit{(Axioma 3.1)} \\ &= a \cdot (b \cdot b) \cdot c && \textit{(Propriedade 6)} \\ &= a \cdot b \cdot c && \textit{(Propriedade 3)} \end{aligned}$$

Portanto, vimos, nestes exemplos, que podemos reduzir as expressões a 0 ou a um produto fundamental. Raciocínio análogo seria utilizado para reduzir qualquer expressão a 0 ou a um produto fundamental que envolva variáveis booleanas. O caso onde a expressão é 0 ou é um produto que já encontra-se na forma reduzida, o resultado é óbvio.

3.4.2 Funções booleanas

Dada uma álgebra booleana $\langle A, +, \cdot, ', 0, 1 \rangle$ e uma expressão booleana em n variáveis $a_1, a_2, \cdots, a_n$, tem-se que uma função booleana é definida como $f : A^n \to A$, onde o valor da função f para um elemento $b = (b_1, b_2, \cdots, b_n) \in A^n$ é calculado substituindo cada ocorrência de a_i na expressão por b_i, para $i = 1, 2, \cdots, n$. (HIRATA, 2005)

Exemplo 12. *A função $f : A^2 \to A$, definida pela expressão $f(a_1, a_2) = a_1 + a_2$ pode ser representada pela tabela verdade a seguir:*

a_1	a_2	$a_1 + a_2$
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*1*
		$*$

Exemplo 13. *Outro exemplo de função booleana é* $g : A^2 \to A$*, onde* $g(a_1, a_2) = a_1 + a_1' \cdot a_2$*, cuja tabela verdade é a seguinte:*

a_1	a_2	a'_1	$a'_1 \cdot a_2$	$a_1 + a_1' \cdot a_2$
0	*0*	*1*	*0*	*0*
0	*1*	*1*	*1*	*1*
1	*0*	*0*	*0*	*1*
1	*1*	*0*	*0*	*1*
				$*$

Notemos que as tabelas-verdade dos Exemplos 12 e 13 são iguais (destaque para $*$, isto implica que as expressões $a_1 + a_2$ e $a_1 + a_1' \cdot a_2$ são equivalentes, isto é, definem uma mesma função).

Algumas formas para descrever/representar funções booleanas são:

- expressões booleanas;
- tabelas verdades;
- diagramas;
- circuitos.

3.5 Derivação de Expressões Booleanas

(GÜNTZEL; NASCIMENTO, 2001, p. 9) afirmam que, frente a uma tabela verdade que descreva uma função booleana, "derivar uma expressão booleana para esta função é encontrar uma equação que a descreva".

> Há basicamente duas maneiras de se definir (ou descrever) uma função booleana: descrevendo-se todas as situações das variáveis de entrada para as quais a função vale 1 ou, alternativamente, todas as situações em que a função vale 0. O primeiro método é conhecido por Soma de Produtos (SdP), enquanto o segundo é chamado Produto de Somas (PdS). Qualquer função booleana pode ser descrita por meio de soma de produtos ou por meio de produtos de somas. Como as funções booleanas só podem assumir um dentro dois valores (0 ou 1), basta usar-se um dos dois métodos para se encontrar uma equação para uma função. (GÜNTZEL; NASCIMENTO, 2001, p. 9)

Vejamos como eles acontecem.

3.5.1 Soma de Produtos (SdP)

A derivação de uma expressão de termos usando soma de produtos acontece por meio do que, na Álgebra Booleana, é chamado de mintermos.

(HIRATA, 2005) define mintermos (ou produto canônico) em n variáveis, $a_1, a_2, \cdots, a_n$, como uma expressão booleana formada pelo produto de cada uma das n variáveis ou dos respectivos complementos, mas não de ambas. A notação para o mintermo é m_n, $n \in \{0, 2^n - 1\}$, onde n é a linha na tabela-verdade que descreve a função correspondente.

A lógica utilizada no minterno é a do 1, ou seja, na tabela verdade, os valores de saída analisados são os que forem 1. Para as variáveis de entrada, aquelas que valerem 1, terão sua representação na forma normal, caso contrário, ou seja, se for 0, a variável será negada.

Como estamos derivando uma expressão por meio de somas de produtos, cada linha cuja variável de saída for 1, terá aqueles literais sendo multiplicados (operação "e" : $\cdot$); no final, os mintermos serão somados (operação "ou" : $+$) a fim de encontrar a expressão de soma de produtos correspondente.

A tabela verdade seguinte apresenta todos os mintermos com 3 variáveis.

Tabela 10 – Tabela verdade de mintermos com 3 variáveis

a	b	c	$\mathrm{m}_{a,b,c} = m_n$
0	0	0	a' $\cdot b' \cdot c' = m_0$
0	0	1	a' $\cdot b' \cdot c = m_1$
0	1	0	a' $\cdot b \cdot c' = m_2$
0	1	1	a' $\cdot b \cdot c = m_3$
1	0	0	a $\cdot b' \cdot c' = m_4$
1	0	1	a $\cdot b' \cdot c = m_5$
1	1	0	a $\cdot b \cdot c' = m_6$
1	1	1	a $\cdot b \cdot c = m_7$

Fonte: Autora

Vejamos um exemplo de como derivar uma expressão booleana através de soma de produtos:

Exemplo 14. *Encontrar uma expressão em soma de produtos (SdP) para função f que está descrita na seguinte tabela verdade:*

a	b	c	f
0	*0*	*0*	*0*
0	*0*	*1*	*0*
0	*1*	*0*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

Os valores de saída que tem valor 1 *são correspondentes aos mintermos* m_2, m_3, m_5 *e* m_6, *sendo que, pela tabela 10,* $m_2 = a' \cdot b \cdot c'$, $m_3 = a' \cdot b \cdot c$, $m_5 = a \cdot b' \cdot c$ *e* $m_6 = a \cdot b \cdot c'$. *Logo, a expressão de soma de produtos equivalente a esta função será a soma destes mintermos, isto é* $f = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$.

3.5.2 Produdo de Somas (PdS)

A derivação de uma expressão usando produto de somas é o método inverso ao de soma de produtos, ou seja, o dual, e ela acontece por meio do chamado maxtermo.

(HIRATA, 2005) define maxtermo, ou soma canônica, em n variáveis, $a_1, a_2, \cdots, a_n$, como uma soma de n literais, cada um correspondendo a uma variável. A notação para o maxtermo é $M_n = n \in \{0, 2^n - 1\}$, onde n é a linha na tabela verdade que descreve a função correspondente.

Mintermos e maxtermos diferem em um se referir a produtos e o outro a somas, nesta ordem.

A lógica utilizada no maxtermo é a do 0, ou seja, na tabela verdade, os valores de saída analisados serão os que forem 0. Para as variáveis de entrada, aquelas que valerem 0 terão sua representação na forma original, caso contrário, isto é, se for 1, a variável será negada.

Como estamos derivando uma expressão por meio de produtos de somas, cada linha, cuja variável de saída for 0, terá aqueles literais sendo somados (operação "ou": $+$) e, no final, os maxtermos serão multiplicados (operação "e" : $\cdot$), para, assim, encontrar a expressão de produto de somas correspondente.

É importante chamar a atenção para o fato de que a soma, neste caso, terá precedência sob o produto. Isto significa que o uso dos parenteses em torno de cada maxtermo se torna obrigatório.

A tabela seguinte traz todos os maxtermos possíveis com 3 variáveis.

Tabela 11 – Tabela verdade de maxtermos com 3 variáveis

a	b	c	$\mathrm{M}_{a,b,c} = M_n$
0	0	0	a + b + c = M_0
0	0	1	a + b + c' = M_1
0	1	0	a + b' + c = M_2
0	1	1	a + b' + c' = M_3
1	0	0	a' + b + c = M_4
1	0	1	a' + b + c' = M_5
1	1	0	a' + b' + c = M_6
1	1	1	a' + b' + c' = M_7

Fonte: Autora

Vejamos um exemplo de como derivar uma expressão booleana através de produto de somas.

Exemplo 15. *Encontrar uma expressão em produto de somas (PdS) para a função que está descrita na seguinte tabela verdade:*

a	b	c	f
0	*0*	*0*	*0*
0	*0*	*1*	*0*
0	*1*	*0*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

Os valores de saída que tem valor 0 *correspondem aos maxtermos* M_0, M_1, M_4 *e* M_7, *sendo que, pela tabela 11,* $M_0 = a + b + c$, $M_1 = a + b + c'$, $M_4 = a' + b + c$ *e* $M_7 = a' + b' + c'$. *Logo, a expressão de produto de somas equivalente a esta função será o produto destes maxtermos, isto é,* $f = (a + b + c)(a + b + c')(a' + b + c)(a' + b' + c')$.

3.6 Simplificação de Expressões Booleanas

Há algumas expressões que trazem consigo muitos elementos (produtos, somas e/ou negações envolvendo as variáveis), então, é normal que se deseje reduzir, ao máximo, o número de operações contidas na expressão. Assim, este "processo de redução de literais (ou de redução de operações, equivalentemente) é denominado simplificação". (GÜNTZEL; NASCIMENTO, 2001, p. 13)

(HIRATA, 2005, p. 45) ressalta que

> se uma expressão pode ser derivada a partir de outra aplicando-se um número finito de vezes as regras (leis/propriedades) da álgebra booleana, então elas são ditas equivalentes. O valor de expressões equivalentes, para cada atribuição de valores às variáveis booleanas, é o mesmo".

(BARANAUSKAS, 2012, p. 14) afirma que "há duas formas para simplificar expressões booleanas: fatoração e mapas de Veitch-Karnaugh". Vejamos como funciona cada uma delas.

3.6.1 Fatoração

Este método consiste em manipulações algébricas, com sucessivas aplicações dos axiomas e propriedades da álgebra booleana, com o objetivo de simplificar, ao máximo, a expressão.

Exemplo 16. *Vamos simplificar a expressão booleana* $S = a \cdot b \cdot c + a \cdot c' + a \cdot b'$.

$$
\begin{aligned}
S &= a \cdot b \cdot c + a \cdot c' + a \cdot b' && \\
&= a\,(b \cdot c + c' + b') && \textit{(Axioma 3.2)} \\
&= a\,(b \cdot c + (c' + b')) && \textit{(Propriedade 6)} \\
&= a\left(b \cdot c + \left((c' + b')'\right)'\right) && \textit{(Propriedade 9)} \\
&= a\left(b \cdot c + \left(\left((c \cdot b)'\right)'\right)'\right) && \textit{(Teorema 10)} \\
&= a\left(b \cdot c + (c \cdot b)'\right) && \textit{(Propriedade 9)} \\
&= a\left(b \cdot c + (b \cdot c)'\right) && \textit{(Axioma 3.1)} \\
&= a \cdot 1 && \textit{(Axioma 3.4)} \\
&= a && \textit{(Axioma 3.3)}
\end{aligned}
$$

Vimos, neste exemplo, como uma expressão pode ser simplificada através de aplicações sucessivas das propriedades da álgebra booleana. Nele, percebemos que, para simplificar um expressão, o primeiro passo é identificar os termos de produtos que se diferenciam apenas por um literal, a fim de aplicar a propriedade distributiva. Na verdade, este é sempre a primeira etapa em qualquer simplificação que envolva variáveis booleanas.

Executada esta etapa, as próximas consistem em aplicar sucessivas vezes axiomas e propriedades já conhecidas da álgebra booleana até que a expressão seja reduzida o máximo possível.

Vejamos outro exemplo.

Exemplo 17. *Simplificar a expressão* $S = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$.

Identificando os produtos que diferem apenas por um literal, temos $a' \cdot b \cdot c'$ *e* $a' \cdot b \cdot c$. *Seguindo os passos já descritos para simplificação, teremos:*

$$
\begin{aligned}
S &= a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c' \\
&= a' \cdot b\,(c + c') + a \cdot b' \cdot c + a \cdot b \cdot c' && \textit{(Axioma 3.2)} \\
&= a' \cdot b \cdot 1 + a \cdot b' \cdot c + a \cdot b \cdot c' && \textit{(Axioma 3.4)} \\
&= a' \cdot b + a \cdot b' \cdot c + a \cdot b \cdot c' && \textit{(Axioma 3.3)} \\
&= a' \cdot b + a \cdot c' \cdot b + a \cdot b' \cdot c && \textit{(Axioma 3.1)} \\
&= b\,(a' + a \cdot c') + a \cdot b' \cdot c && \textit{(Axioma 3.2)} \\
&= b\,(a' + c') + a \cdot b' \cdot c && \textit{(Axiomas 3.2, 3.3 e 3.4)} \\
&= a' \cdot b + b \cdot c' + a \cdot b' \cdot c && \textit{(Axioma 3.2)}
\end{aligned}
$$

Notemos que na expresão $a' \cdot b + a \cdot b' \cdot c + a \cdot b \cdot c'$ já obtemos uma simplificação em relação à expressão original, uma vez que houve redução no número de operações e de literais. Assim, essa expressão pode ser dita como uma soma de produtos simplificada, porém não é mínima, visto que ainda seria possível realizar manipuções algébricas, a fim de reduzir ainda mais a expressão dada.

Em $a' \cdot b + b \cdot c' + a \cdot b' \cdot c$, foram feitas todas as simplificações possíveis, pois foram agrupados e simplificados todos os termos que diferiam apenas por uma variável. Quando isso acontece, a expressão chega a sua máxima simplificação. As expressão $b\,(a' + c') + a \cdot b' \cdot c$, por sua vez, é dita expressão na forma fatorada.

3.6.2 Mapas de Veitch-Karnaugh

Conforme acrescenta (BARANAUSKAS, 2012, p. 28)

> alternativamente ao método de simplificação algébrico por fatoração, há outro método de simplificação baseado na identificação visual de grupos de mintermos que podem ser simplificados. Para tanto, é necessário que os mintermos seja dispostos, de maneira conveniente, em tabelas conhecidas como diagramas ou mapas de Veitc-Karnaugh.

Nesse sentido, o mapa de Karnaugh, idealizado em 1952 por Edward Veitch e aperfeiçoado por Maurice Karnaugh, é um método de simplificação de expressões booleanas, utilizando representações gráficas. (ABAR, 2004)

O modo mais prático para a aplicação deste método é utilizando a tabela verdade determinada pela expressão que se deseja simplificar. Feito isso, devemos distribuir estes valores, de modo conveniente, no mapa de Karnaugh.

Para isto, consideremos S_n, $n \in \{0, 2^n - 1\}$, onde n é o número de variáveis da expressão, o valor final da n-ésima linha na tabela verdade. Por exemplo, para 2 e 3 variáveis temos, respectivamente, as seguintes tabelas verdades:

Tabela 12 – Tabela verdade apresentando o valor final S_n com 2 variáveis

a	b	S_n
0	0	S_0
0	1	S_1
1	0	S_2
1	1	S_3

Fonte: Autora

Tabela 13 – Tabela verdade apresentando o valor final S_n com 3 variáveis

a	b	c	S_n
0	0	0	S_0
0	0	1	S_1
0	1	0	S_2
0	1	1	S_3
1	0	0	S_4
1	0	1	S_5
1	1	0	S_6
1	1	1	S_7

Fonte: Autora

Esses valores S_n, $n \in \{0, 2^n - 1\}$ devem ser aplicados no mapa da seguinte forma:

Figura 3 – Distribuição no mapa para 2 variáveis

a \ b	0	1
0	S_0	S_1
1	S_2	S_3

Fonte: Autora

Figura 4 – Distribuição no mapa para 3 variáveis

a \ bc	00	01	11	10
0	S_0	S_1	S_3	S_2
1	S_4	S_5	S_7	S_6

Fonte: Autora

É importante enfatizar a questão de todos os valores S_n sejam dispostos exatamente como está no mapa, onde, de um quadro para o outro, há a menor mudança possível das constantes.

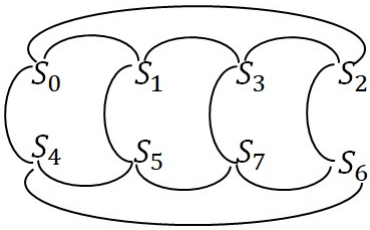
Na distribuição da figura 3 está claro os valores de entrada em relação as suas variáveis, já na da 4, onde na primeira linha estão dispostos as possíveis entradas para bc, os valores das demais linhas correspondem aos valores de b e c, respectivamente.

Estamos mostrando exemplos para duas e três variáveis. No entanto, para mais variáveis a ideia seria a mesma.

Assim, como os valores possíveis para S_n são 0 ou 1, esse método de simplificação consiste em fazer agrupamentos, quando possível, de todos os "1" existentes no mapa. Esses agrupamentos, por sua vez, devem ser feitos traçando grupos de múltiplos de 2 "1" adjacentes, onde nenhum "1" pode ficar de fora dos grupos formados, sendo que, se necessário, ele pode ser agrupado mais de uma vez, mas nunca sem necessidade. (ABAR, 2004) No entanto, "é importante ressaltar que o conceito de adjacência é aplicável na horizontal e na vertical, mas nunca na diagonal". (GÜNTZEL; NASCIMENTO, 2001, p. 19)

Vejamos, na figura seguinte, os possíveis agrupamentos para uma expressão de três variáveis, visto que, na de duas, todos são agrupáveis, exceto na diagonal.

Figura 5 – Possíveis agrupamentos para 3 variáveis



Fonte: Autora

Após a identificação de todos os grupos de "1" adjacentes entre si, cada um deles originará um produto, onde as variáveis deste serão as que correspondem aos valores de entrada comuns a todos os componentes do grupo. É importante ressaltar que, se o valor comum for 1, a variável ficará na sua forma original, no entanto, se for 0, estará na forma complementar ou negada.

O resultado final será a soma de todos os produtos obtidos na junção dos grupos.

Em termos operacionais da álgebra booleana, os grupos formados estão munidos através da operação $\cdot$ ("e") e, os diferentes grupos, serão somados ($+$, operação "ou").

Vejamos como essa forma de simplificação funciona com as expressões já simplificadas anteriormente.

Exemplo 18. *Simplificar a função* $S = a \cdot b \cdot c + a \cdot c' + a \cdot b'$.

Tabela verdade:

Tabela 14 – Tabela verdade da função $S = a \cdot b \cdot c + a \cdot c' + a \cdot b'$

a	b	c	b'	c'	a $\cdot b \cdot c$	a $\cdot c'$	a $\cdot b'$	S = a $\cdot b \cdot c + a \cdot c' + a \cdot b'$
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	1	1	1
1	0	1	1	0	0	0	1	1
1	1	0	0	1	0	1	0	1
1	1	1	0	0	1	0	0	1

Fonte: Autora

Figura 6 – Mapa de Karnaugh para função $S = a \cdot b \cdot c + a \cdot c' + a \cdot b'$

a \ bc	00	01	11	10
0	0	0	0	0
1	1	1	1	1

Fonte: Autora

*Portanto, extraindo os valores de saída da tabela 14 e colocando-os no mapa da figura 6, temos que o conjunto de "*1*" encontrado são os quatro que encontram-se na última linha. Nela, o único valor comum a todos ele é o* 1*, que está fazendo referência a variável* a*. Portanto, a simplificação resulta em* $S = a$*.*

Exemplo 19. *Simplificar a função* $S = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$*.*

Tabela 15 – Tabela verdade da função $S = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$

a	b	c	a'	b'	c'	a' $\cdot b \cdot c'$	a' $\cdot b \cdot c$	a $\cdot b' \cdot c$	a $\cdot b \cdot c'$	S
0	0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0
0	1	0	1	0	1	1	0	0	0	1
0	1	1	1	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	0	0	0
1	0	1	0	1	0	0	0	1	0	1
1	1	0	0	0	1	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0	0

Fonte: Autora

Figura 7 – Mapa de Karnaugh para função $S = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$

a \ bc	00	01	11	10
0	0	0	1	1
1	0	1	0	1

Fonte: Autora

*Extraindo os valores de saída da tabela 15 e colocando-os no mapa da figura 7, temos que os conjuntos de "*1*" encontrados são os que encontram-se na última coluna, onde as constantes comuns são o* 1 *e* 0*, representando b e c, respectivamente, que resulta no produto entre b e c negado, visto que a constante do c é* 0*; encontramos também a junção de dois "*1*" na quarta e quinta coluna da segunda linha, onde as constantes comuns são* 0 *e* 1*, que representam o a e o b, respectivamente, que resultará em* $a' \cdot b$*; e, por fim, há um único* 1 *no valor correspondente à terceira linha, terceira coluna, que tem como constantes* 1*,* 0 *e* 1*, que implica no produto* $a \cdot b' \cdot c$*. Portanto, fazendo a soma de cada produto encontrado, temos que a simplificação para função* $S = a' \cdot b \cdot c' + a' \cdot b \cdot c + a \cdot b' \cdot c + a \cdot b \cdot c'$ *é* $S = a' \cdot b + b \cdot c' + a \cdot b' \cdot c$*.*

Vimos nesta seção, portanto, duas formas distintas de simplificar expressões booleanas: a fatoração, onde a simplificação é feita algebricamente por meio da manipulação e/ou aplicações de axiomas e propriedades da álgebra booleana, e o processo de simplificação fazendo uso do mapa de Karnaugh, que é uma técnica onde, através de uma tabela verdade e de um mapa que irá utilizar, convenientemente, valores da tabela verdade, obtém-se, ao final do processo, a menor expressão possível.

4 Circuitos Lógicos e Aplicações

Neste capítulo, iremos comentar sobre os circuitos lógicos, que são construídos através de portas lógicas; outros operadores lógicos, além dos básicos, também serão abordados; e, por fim, serão mostradas algumas aplicações da Álgebra de Boole.

4.1 Circuitos Lógicos

> Já vimos que uma função booleana pode ser representada por uma equação ou detalhada por sua tabela verdade. Mas uma função booleana também pode ser representada de forma gráfica, onde cada operador está associado a um símbolo específico, permitindo o imediato reconhecimento visual. Tais símbolos são reconhecidos por portas lógicas. (GÜNTZEL; NASCIMENTO, 2001, p. 6)

Como também já visto, um conjunto de operadores, juntamente com variáveis, dão origem a uma expressão booleana. Voltado para um contexto de símbolos, quando os operadores algébricos são substituídos por portas lógicas, um conjunto de portas lógicas dão origem aos circuitos lógicos. As portas lógicas, por sua vez, podem ser vistas como circuitos elementares.

Cada circuito lógico pode ser visto como uma máquina L que contém um ou mais dispositivos de entrada e exatamente um dispositivo de saída. Cada dispositivo de entrada em L envia um sinal, especificamente, um *bit* (dígito binário), 0 ou 1, para o circuito L e este processa o conjunto de bits para gerar um bit de saída. Por conseguinte, uma sequência de n bits pode ser atribuída a cada dispositivo de entrada e L processa as sequências de entrada, um bit de cada vez, para produzir uma sequência de saída de n bits.

A introduçao da álgebra booleana no estudo dos circuitos lógicos ocorreu somente em 1938, com o matemático americano Claude Elwood Shannon ($1916 - 2001$), quando ele percebeu o paralelo entre lógica poposicional e a lógica de circuitos, e compreendeu que esta álgebra poderia ter um papel fundamental na sistematização de um novo ramo da eletrônica. (SANTOS; OLIVEIRA, 2016)

Portanto, "a álgebra booleana descreve os circuitos que podem ser construídos pela combinação de portas lógicas em que as variáveis e funções pode ter apenas valores 0 e 1". (VIEIRA, 2000, p. 11)

Veremos como funcionam as portas lógicas para, em seguida, investigar os circuitos lógicos.

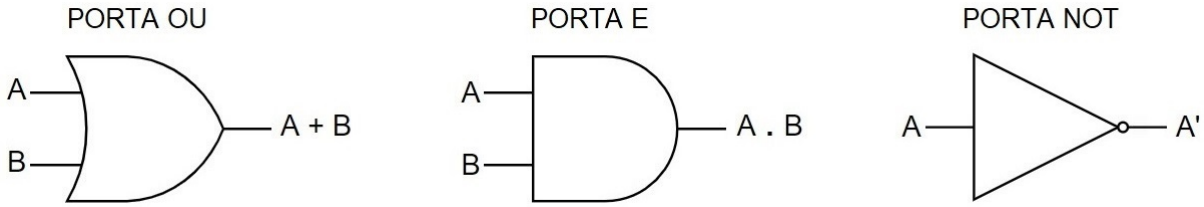
4.1.1 Portas Lógicas

"As portas lógicas representam mais do que símbolos de operadores, elas representam circuitos físicos, como, por exemplo, os circuitos eletrônicos, sendo 0 a ausência de tensão (0 volt) e 1 a presença de tensão (normalmente 5 volts)." (OLIVEIRA, 2017, p. 26)

Há três portas lógicas fundamentais, cada uma para representar os operadores lógicos "ou", "e" e "não". Neste trabalho, vamos adotar a convenção de que as linhas que entram à esquerda da porta são as de entrada e, à direita, será a de saída, que é única.

Vejamos cada uma dessas portas lógicas:

Figura 8 – Portas lógicas

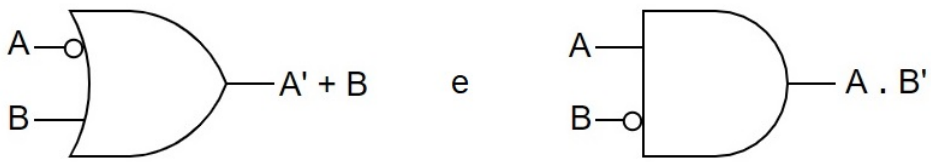


Fonte: Autora

As portas "ou" e "e" necessitam de pelo menos duas entradas, pois representam a adição e a multiplicação lógica, respectivamente, e têm exatamente uma saída. Já a porta "not", também chamada de porta inversora, que representa a negação ou complemento, só pode ser realizada com uma variável por vez, tendo, portanto, apenas uma única entrada e uma única saída.

É importante acrescentar que a porta "not" também pode ser representada por meio de uma aplicação direta da variável a ser negada na operação seguinte, como na figura 9.

Figura 9 – Portas lógicas com o inversor aplicados diretamente na operação



Fonte: Autora

4.1.2 Exemplos de Circuitos Lógicos

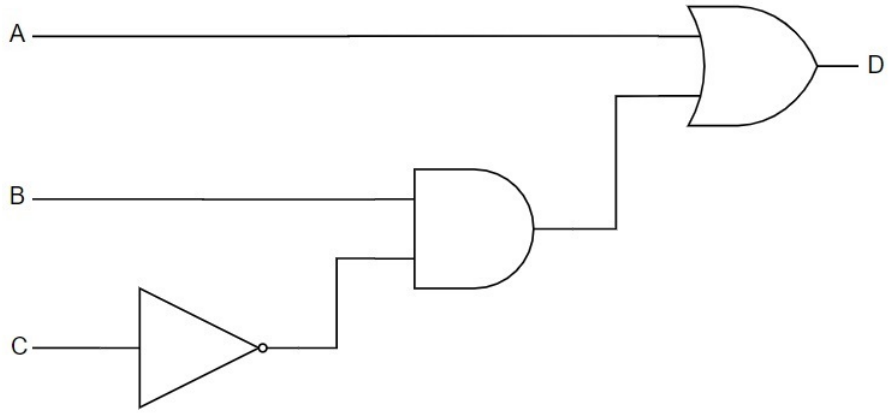
Dada uma função booleana qualquer, é possível desenhar o circuito lógico que a representa, sendo que este circuito é composto das portas lógicas relacionadas às ope-

rações que são realizadas sobre as variáveis de entrada. Os resultados das operações são conduzidos por fios, os quais, no desenho, são representados por linhas simples. (GÜNTZEL; NASCIMENTO, 2001)

Para construir um circuito lógico, a ordem é similar à seguida no processo de simplificação de uma função booleana: complemento, parenteses, produto e soma. Para cada variável de entrada, traça-se uma linha saindo delas até as portas necessárias para representar cada uma das subexpressões.

Exemplo 20. *Vejamos como fica o circuito lógico da função* $D = A + B \cdot C'$.

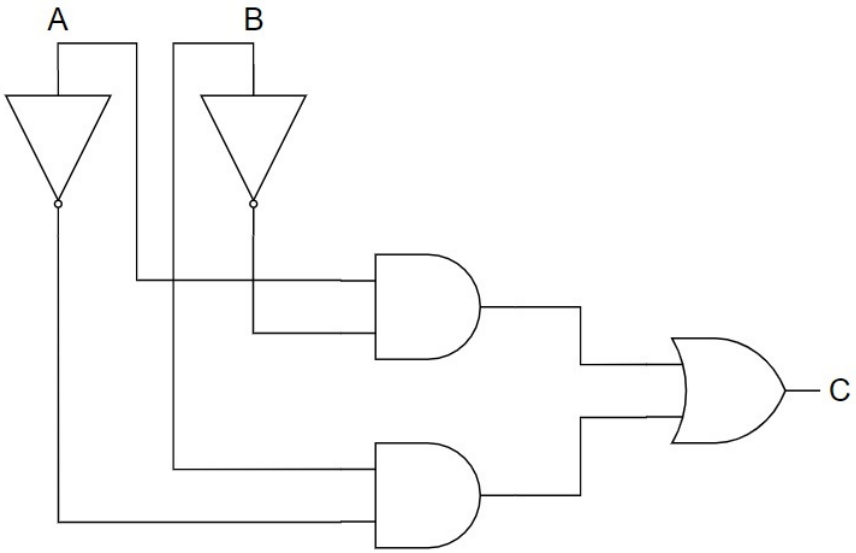
Figura 10 – Circuito lógico $D = A + B \cdot C'$



Fonte: Autora

Exemplo 21. *Fazendo o procedimento inverso, porém na mesma linha de raciocínio, vejamos como, frente a um circuito lógico, podemos encontrar a função que o determinou.*

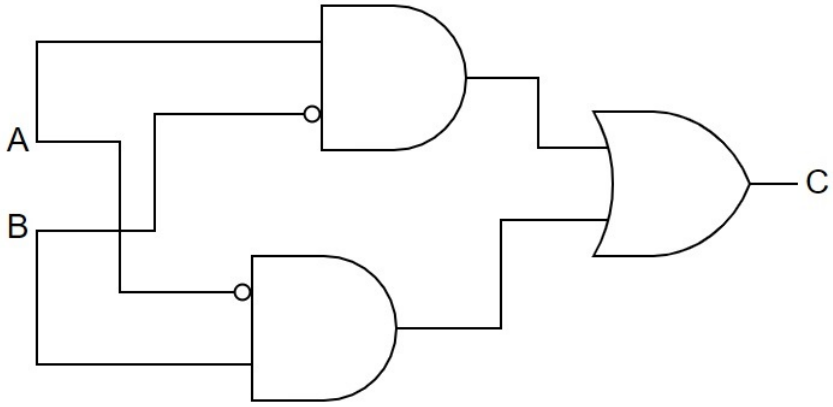
Figura 11 – Circuito lógico



Fonte: Autora

Esse circuito também poderia ser repesentado como segue abaixo, que nos levaria à mesma função booleana.

Figura 12 – Circuito lógico



Fonte: Autora

Nesse caso, o método mais prático é começar analisando as portas da direita para esquerda, acompanhando as linhas que as geraram, até chegar às variáveis. Assim, a primeira porta encontrada é a "ou", o que significa que teremos uma soma, que foi gerada por duas portas "e", ou seja, esta será uma soma de dois produtos. O primeiro produto é entre a variável A, que passou pela porta inversora, e a variável B, ou seja, $A' \cdot B$; já o segundo é o produto é entre as variáveis A e B, sendo que esta última também

passou pela porta inversora, logo $A \cdot B'$*. Portanto, este circuito pode ser expresso pela função* $C = A' \cdot B + A \cdot B'$*.*

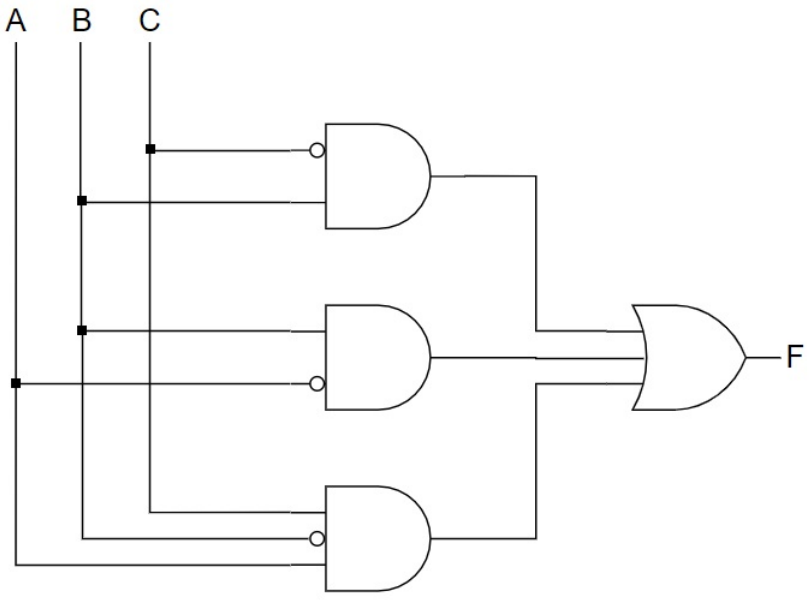
Conforme afirmam (GÜNTZEL; NASCIMENTO, 2001, p. 13),

> o número de elementos (portas lógicas e conexões) de um circuito lógico depende diretamente do número de operações booleanas (inversão, "e" e "ou") contidas na expressão associada. Desta forma, é normal que se deseje reduzir o número de operações contidas numa função de modo a poder-se implementá-la com circuitos lógicos mais simples e, portanto, de menos custo.

Assim, "no contexto de circuitos, uma das questões mais importantes é saber determinar o melhor circuito (em termos de custo, eficiência, etc) que realiza uma dada função" (HIRATA, 2005, p. 52). Esses circuitos mais simples podem ser obtidos através da simplificação da expressão da função original, conforme visto na seção 3.6.

Logo, ao invés de construir o circuito lógico da função booleana $F = A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$, poderíamos construir o circuito equivalente da expressão reduzida $F = A' \cdot B + B \cdot C' + A \cdot B' \cdot C$, que seria o seguinte:

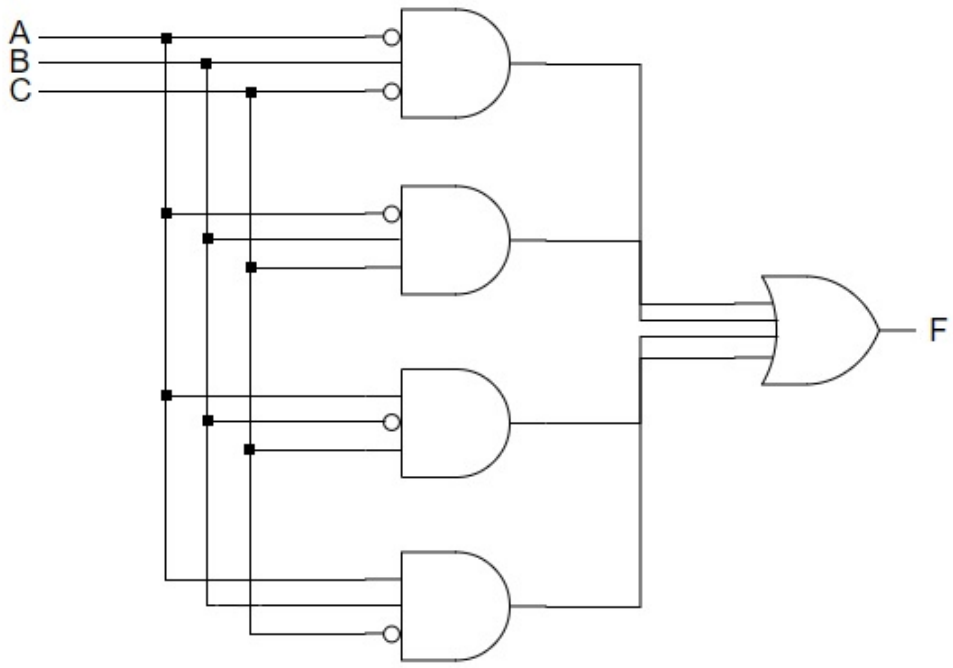
Figura 13 – Circuito lógico $F = A' \cdot B + B \cdot C' + A \cdot B' \cdot C$



Fonte: Autora

É importante chamar atenção para o fato de que, na imagem, fios que se cruzam, onde aqueles que não tem o pontinho preto não possuem conexão entre si. Caso fossémos fazer o circuito da função $F = A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$, teríamos

Figura 14 – Circuito lógico $F = A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$



Fonte: Autora

onde é notório a redução do tamanho do circuito da figura 13 em relação ao da 14.

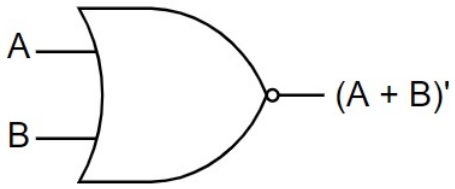
4.2 Outros Operadores Lógicos

Além dos operadores já vistos até aqui, existem outros não muito utilizados, que são formados pela combinação dos operadores básicos (OU, E e NOT). Estes são NOR $(a + b)'$, que é o complemento da operação OU; NAND $(a \cdot b)'$, ou seja, o complementar da operação E; XOR $(a \oplus b)$, que significa a ou b, mas não ambos, ou seja, o OU exclusivo; e, por fim, XNOR $(a \oplus b)'$, que é o NOR exclusivo. (GÜNTZEL; NASCIMENTO, 2001)

Conforme ressalta (LIPSCHUTZ; LIPSON, 1976), as operações NOR, NAND, XOR e XNOR, analogamente às básicas OU e AND, podem ter duas ou mais entradas.

Vejamos a tabela verdade de cada um desses operadores para duas variáveis, e suas respectivas portas lógicas.

Figura 15 – Porta lógica do operador NOR



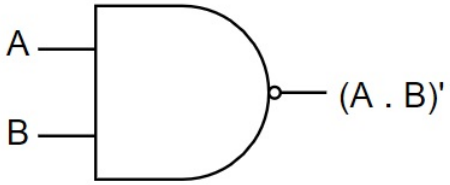
Fonte: Autora

Tabela 16 – Tabela verdade do operador NOR com 2 variáveis

A	B	(A + B)'
0	0	1
0	1	0
1	0	0
1	1	0

Fonte: Autora

Figura 16 – Porta lógica do operador NAND



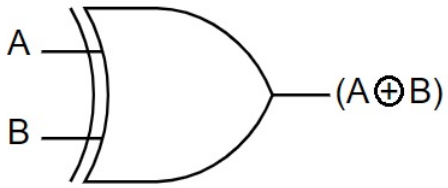
Fonte: Autora

Tabela 17 – Tabela verdade do operador NAND com 2 variáveis

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

Fonte: Autora

Figura 17 – Porta lógica do operador XOR



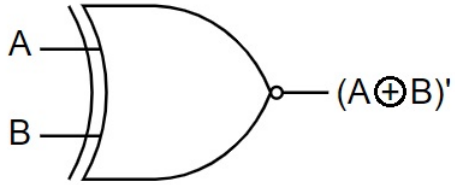
Fonte: Autora

Tabela 18 – Tabela verdade do operador XOR com 2 variáveis

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

Fonte: Autora

Figura 18 – Porta lógica do operador XNOR



Fonte: Autora

Tabela 19 – Tabela verdade do operador XNOR com 2 variáveis

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

Fonte: Autora

Observemos que a única diferença entre as portas OU E NOR, E e NAND e XOR e XNOR é que NOR, NAND e XNOR são seguidas por um círculo logo após o símbolo original.

É importante ressaltar que alguns textos também utilizam esse círculo após a porta para indicar a inversão, que também poderia ser feita através da porta exclusiva para negação da Figura 8, pois, como visto nesta seção, estes operadores são formadas através de combinações dos operadores básicos OU, E e NOT.

4.3 Aplicações

O sistema algébrico apresentado neste capítulo dá origem, dentre outras aplicações, ao que se convencionou chamar-se de Lógica Digital, que toma por base o sistema conceitual da álgebra de Boole, sendo que a simbologia e as características dos elementos com que opera modificam-se no sentido de operacionalizar máquinas elétricas, onde os impulsos elétricos representam os valores lógicos (DIAS, 1994). Nesse sentido, iremos mostrar que a álgebra booleana, muito utilizada para projetos de circuitos elétricos, também se faz presente na programação do buscador Google, além de ter sido estímulo para desenvolver jogos que despertam o desenvolvimento do raciocínio lógico.

4.3.1 Circuitos Elétricos

Analogamente ao já visto até aqui, na lógica digital também vamos contar com a presença de duas constantes (0 e 1), que aqui vão representar os estados lógicos, isto é, o estado lógico 0 e o estado lógico 1. Segundo (DIAS, 1994, p. 51),

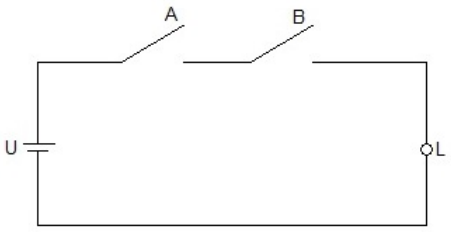
> fisicamente, entretanto, tais estados estão associados à posição de um interruptor ligado a um ponto de um dado circuito elétrico. Ou seja, o estado lógico 1 passa a indicar o correspondente ao interruptor fechado, isto é, quando o interruptor encontra-se fechado, o mesmo permite que a corrente flua através do ponto onde este se encontra. Por outro lado, o estado lógico 0 relaciona-se ao interruptor aberto e, consequentemente, nenhuma corrente pode passar pelo ponto considerado.

Estabelecida essa relação, vamos ver como as funções booleanas, até aqui escritas na forma algébrica, através de tabelas verdades ou de circuitos lógicos, podem ser associadas, fisicamente, aos circuitos elétricos.

4.3.1.1 Circuitos em Série

Os circuitos em série são os mais simples dos circuitos elétricos (OLIVEIRA, 2017) e é aquele onde há um único percurso para a corrente de energia passar. Para efeitos de ilustração, consideremos o seguinte circuito, onde U representa uma fonte de energia, L uma lâmpada que se queira acender e A e B dois interruptores.

Figura 19 – Circuito em Série



Fonte: Autora

Nela, L se acenderá se, e somente se, A e B estiverem ambos ligados, levando L ao estado lógico 1. No entanto, se ao mesmo um dos interruptores estiver desligado L não acenderá, o que implica o estado lógico 0.

Essa ideia pode ser transcrita pela seguinte tabela:

Tabela 20 – Tabela verdade de um circuito em série

A	B	L
0	0	0
0	1	0
1	0	0
1	1	1

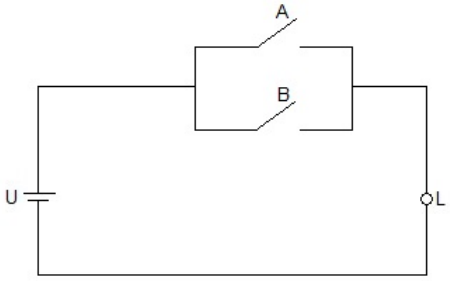
Fonte: Autora

Neste exemplo, temos uma clara aplicações do operador AND, ou seja, de uma multiplicação lógica. Assim, este circuito, que também pode ser chamado de circuito AND, está associado a função booleana $L = A \cdot B$.

4.3.1.2 Circuito em Paralelo

Vamos agora observar um exemplo de circuito que os interruptores estão conectados em paralelo, onde a corrente elétrica tem dois caminhos a seguir.

Figura 20 – Circuito em Paralelo



Fonte: Autora

Nesse caso, para que a lâmpada esteja acesa (estado lógico 1), é necessário e suficiente que pelo menos um dos interruptores esteja ligado, ou seja, a lâmpada só não acenderá (estado lógico 0) se ambos os interruptores estiverem desligados. Podemos traduzir esses fatos, na seguinte tabela:

Tabela 21 – Tabela verdade de um circuito em série

A	B	L
0	0	0
0	1	1
1	0	1
1	1	1

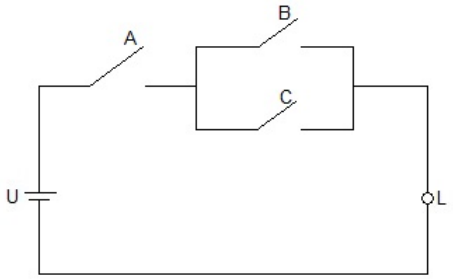
Fonte: Autora

Agora, é fácil ver que se trata de uma aplicação do operador OU, que resulta no circuito OU, e L pode ser escrito algebricamente como $L = A + B$.

4.3.1.3 Circuito Misto

Neste caso, teremos aplicações dos operadores OU e E simultaneamente, como será o caso do circuito a seguir:

Figura 21 – Circuito Misto



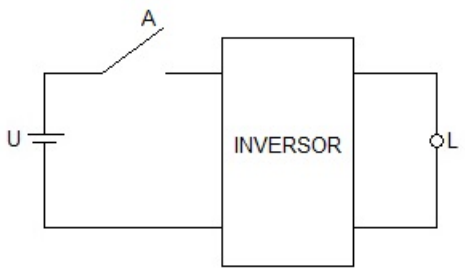
Fonte: Autora

Aqui, a lâmpada L se acenderá se o interruptor A estiver ligado e se um dos interruptores B e C também estiver. Algebricamente, poderíamos traduzir este circuito para $L = A \cdot (B + C)$.

4.3.1.4 Circuito Inversor

Teremos, agora, um aplicação do operador NOT. Para isto, consideremos o seguinte circuito.

Figura 22 – Circuito Inversor



Fonte: Autora

Nesse caso, quando A está ligado, L está apagado, pois o circuito inversor muda a passagem da corrente elétrica. Já quando A está desligado, L se encontrará acesa, o que dá origem a seguinte tabela verdade:

Tabela 22 – Tabela verdade de um circuito inversor

A	L
0	1
1	0

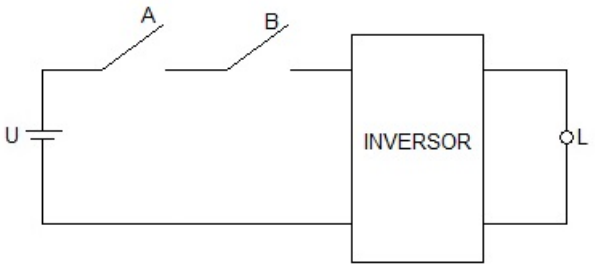
Fonte: Autora

Em termos de álgebra booleana, temos uma aplicação do operador NOT, e, algebricamente, este circuito pode ser traduzido por $L = A'$.

4.3.1.5 Circuitos NAND e NOR

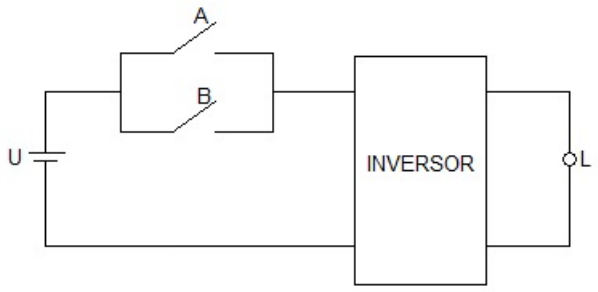
Este circuito podem ser criados a partir da combinação dos circuitos OU, AND e NOT, similar ao que aconteceu com os operadores. Vejamos exemplos de cada um deles:

Figura 23 – Circuito NAND



Fonte: Autora

Figura 24 – Circuito NOR



Fonte: Autora

Estes circuitos podem também ser expressos pelas tabelas verdades a seguir:

Tabela 23 – Tabela verdade de um circuito NAND

A	B	A $\cdot B$	L
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Fonte: Autora

Tabela 24 – Tabela verdade de um circuito NOR

A	B	A + B	L
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Fonte: Autora

Ou, de forma algébrica, $L = (A \cdot B)'$ e $L = (A + B)'$, respectivamente.

4.3.2 Busca do Google

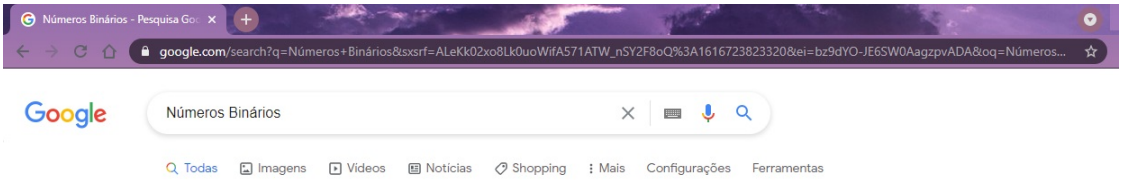
O Google, também chamado de Gigante das Buscas, é um site utilizado por milhares de pessoas no mundo para buscas e pesquisas nos diversos segmentos. No entanto, apesar dessa grande quantidade de usuários utilizá-lo, poucos sabem que, por trás de sua base de programação, encontra-se a álgebra de Boole.

A busca do Google funciona com a utilização implícita do operador E. Por exemplo, se é feito uma busca direta como “Números Binários”, o Google usa o comando E para combinar as palavras “Números” e “Binários”. Em outros buscadores antigamente, ou até mesmo no início do próprio Google as últimas páginas não traziam exatamente

o resultado esperado, isso acontecia devido a presença do operador OU. No exemplo "Números Binários", as últimas páginas trariam resultados "Números" ou "Binários", mas o Google já inovou ao atualizar sua programação e hoje em dia só se faz uso do operador E". (OLIVEIRA, 2017)

É importante chamar atenção para o sinal de adição na barra de endereço, onde fica o link da pesquisa, que está separando as palavras chaves introduzidas no buscador.

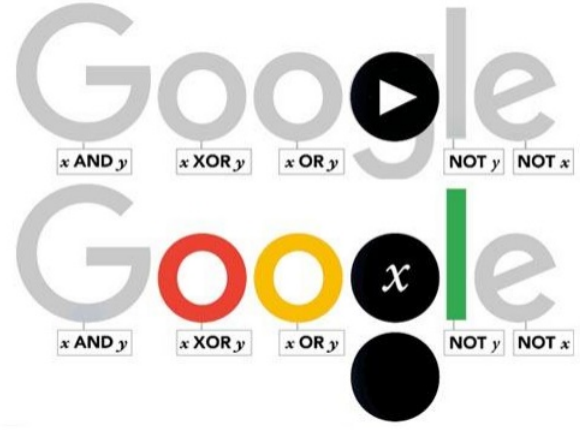
Figura 25 – Barra de endereço de uma busca no Google



Fonte: Autora

"Vale ressaltar que o nome Google é um amálgama com o nome George Boole. Em linguística amálgama é a mistura de duas palavras com o objetivo de formar um nova" (OLIVEIRA, 2017, p. 35), inclusive o Google homenageou o 200° aniversário de seu precursor George Boole com um Google Doodle, que simulava as portas lógicas, já estudadas na subseção 4.1.1.

Figura 26 – Barra de endereço de uma busca no Google



Fonte: (OLIVEIRA, 2017)

4.3.3 Jogos Boole

"Os jogos de Boole se constituem em um elemento motivador de aprendizagem, principalmente entre alunos que demonstram dificuldades em realizar cálculos abstratos por não terem o raciocínio lógico bem desenvolvido" (GIRELLI; BEZERRA, 2014, p.

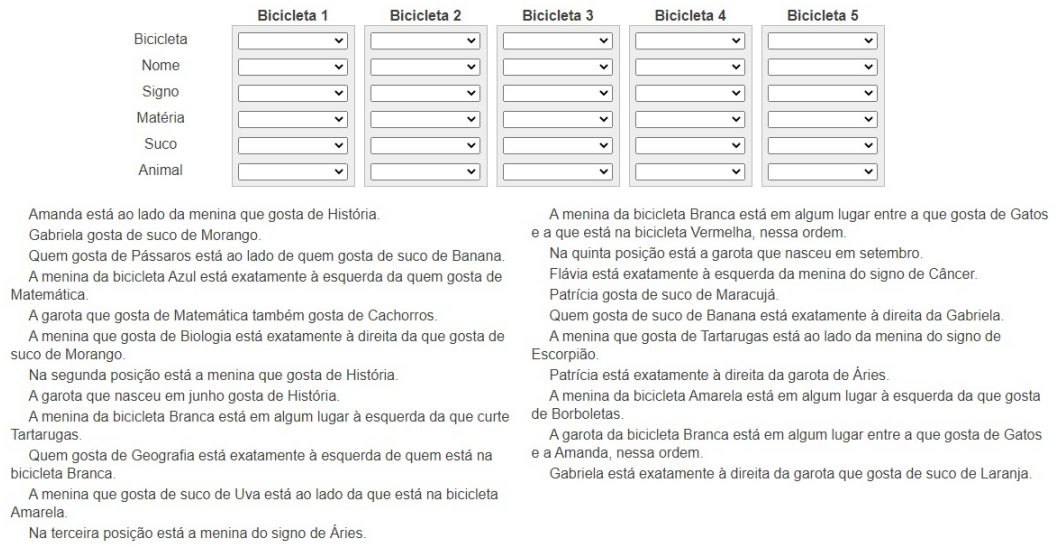
01) e é resultado de um projeto realizado há mais de 20 anos por Procópio Mendonça Mello, com o objetivo de desenvolver formas de raciocínio matemático.

> As histórias apresentadas nos jogos Boole têm como solução verdadeiras matrizes, no sentido matemático do termo, isto é, quadros de fileiras e colunas conexas. As colunas correspondendo aos critérios característicos da estrutura predeterminada e as linhas correspondendo aos objetos de mesmas categorias (ou significações). Os jogos consistem na aplicação das operações booleanas aos enunciados verbais, transformando estes em equações lógicas por intermédio da simbologia da álgebra booleana. (MELLO, 2010, p. 183)

Estes jogos também são chamados de Problemas de Lógica.

A imagem que segue é um exemplo deste jogo.

Figura 27 – Jogo Boole - Dificuldade Normal

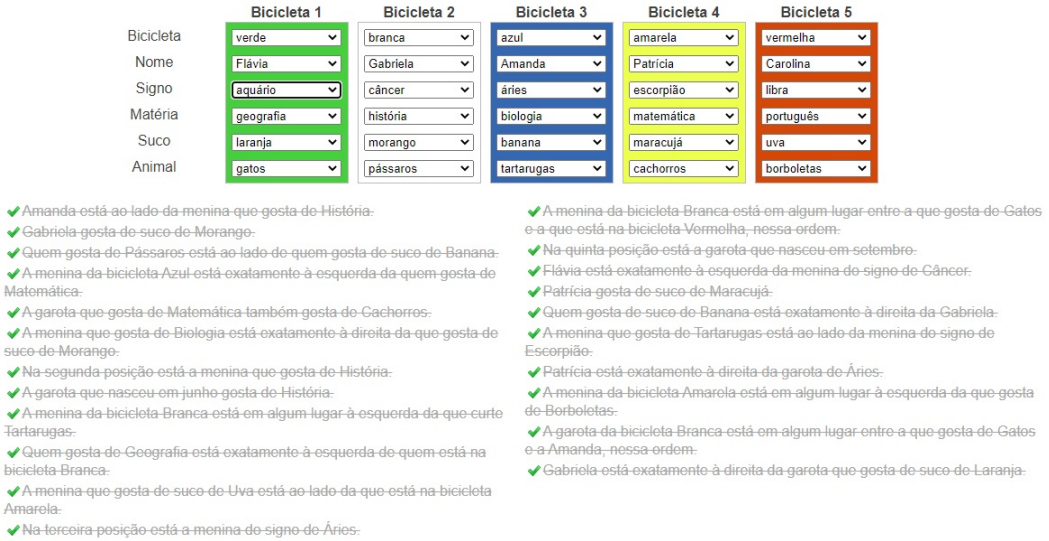


Fonte: Autora

Nele, as linhas e as colunas têm que ser correlacionadas entre si de modo que todas as dicas fiquem verdadeiras.

Vejamos o jogo resolvido:

Figura 28 – Jogo Boole Resolvido - Dificuldade Normal



Fonte: Autora

> Veja que, quando o jogo está resolvido, as dicas estão riscadas, isso significa que o jogo está dizendo que aquela dica foi preenchida corretamente, ou seja, uma dica verdadeira. Se olharmos a resolução do jogo e resolvêssemos mudar algo de lugar, uma dessas dicas não ficaria mais riscada, pois ela se tornaria uma dica falsa. Com isso, percebe-se que cada jogo desse tipo possui apenas uma resolução correta. É aí que está a álgebra proposicional, as dicas podem ser interpretadas como proposições, e devem ser todas as verdadeiras quando a operação de E for realizada entre elas. (OLIVEIRA, 2017, p. 37)

Num contexto voltado para sala de aula, estes jogos são frequentemente utilizados com cartas, tendo em vista que

> ao considerar a ludicidade como um processo de ensino aprendizagem, o educador favorece seu foco de atuação, pois o trabalho em grupo enriquece as relações grupais caracterizando um contexto propício para a utilização de atividades lúdicas. Essas atividades são meios de comunicações que permitem ressignificar o real, permitindo que o educando se envolva interagindo em situações lógicas, afetivas e sociais. (DIDáTICO-PEDAGóGICAS, 2013, p. 02)

5 Introdução aos Computadores

Neste capítulo, apresentaremos alguns conceitos elementares pertinentes ao tema em questão, e, em seguida falaremos dos processadores ou CPU's dos computadores, enfatizando, sobretudo, como se dá seu funcionamento através da linguagem binária. Falaremos também do código ASCII, que é conhecido e utilizado em todo mundo, bem como do Unicode, com ênfase do Padrão UTF-8 e, por fim, mostraremos, de fato, como essas ferramentas se interligam para facilitar nossa comunicação com os computadores.

5.1 Conceitos Elementares

Já vimos que, numa perspectiva voltada para o mundo digital, existem apenas dois algarismos (0 ou 1), que indicam, na verdade, estados do sistema, tais como desligado ou ligado, fechado ou aberto, não ou sim, entre outros. Estes algarismos, são, na verdade, a linguagem que os computadores utilizam para armazenar e processar os dados, onde toda informação é convertida para linguagem binária, isto é, zeros e uns.

5.1.1 Bit

Essa ideia de 0 e 1 nos remete a um termo denominado Bit, cujo símbolo é *b*, que, por definição, é a menor unidade utilizada na computação para medir a transferência ou armazenamento de informações digitais. O termo é uma abreviação da expressão dígito binário ou *binary digit*, em inglês. (GOGONI, 2019)

Sendo binário, eles só podem assumir os valores 1 e 0, que representam, respectivamente, a passagem, ou não, de corrente elétrica.

> Fisicamente, um bit pode ser representado de várias formas: via eletricidade (dois valores de voltagem aplicados num fio), via luz (em fibras ópticas), via ondas eletromagnéticas (redes sem fio), enfim, em tudo que seja possível identificar dois estados diferentes. (TEDESCO, 2020)

Assim, um computador, ao processar dados, ele estará, na verdade, processando bit, ou seja, dizer esse bit é 0 ou 1, onde, ao final do processamento, teremos uma informação mais completa, isto é, agrupamentos de bits zeros e uns.

É importante chamar atenção para o fato de que esses agrupamentos de bits representam todo e qualquer dado que estiver armazenado no computador, tais como imagens, textos, músicas, áudios, vídeos, entre outros.

Para realizar o cálculo das possibilidades do agrupamento dos bits, iremos utilizar sempre a base 2, visto que estamos tratando da base binária, elevado ao número de bits agrupados.

Por exemplo,

Tabela 25 – Possibilidades de agrupamentos de bits

1 bit	2 bits	8 bits	n bits
$2^1 = 2$	$2^2 = 4$	$2^8 = 256$	2^n
2 possibilidades	4 possibilidades	256 possibilidades	2^n possibilidades

Fonte: Autora

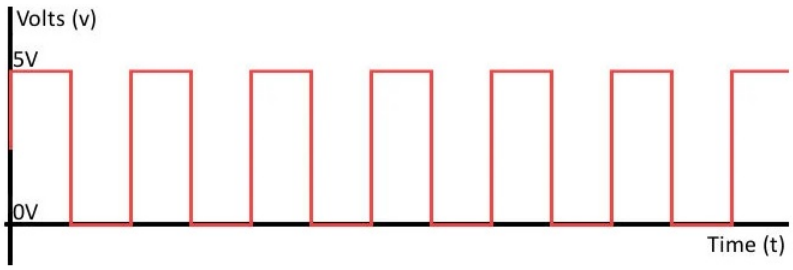
Neste sentido, para 3 bits, há 8 possibilidades de agrupamentos, o que significa que o computador pode armazenar até 8 informações por vez.

(CANDIDO, 2013, p. 3) ressalta que

> Em nossa linguagem, a menor unidade de informação é o caractere, que, isoladamente, não serveria como uma informação útil. Em computação, o bit seria da mesma forma que o caractere. Por essa razão, as informações manipuladas por um computador são codificadas em grupos ordenados de bits, para poder terem um significado útil.

A ilustração seguinte exibe um sinal digital, onde 1 indica a presença de tensão $(5V)$, enquanto sua ausência $(0V)$ é representada por 0.

Figura 29 – Imagem Bit



Fonte: https://tecnoblog.net/303263/bit-ou-byte/

É assim que um computador "fala". O bit é geralmente utilizado para medir velocidades de conexão e transferência de dados, embora, no passado, também tenha sido usado como unidade básica para armazenamento. (GOGONI, 2019)

No entanto, não podemos armazenar menos de 8 bits. Nesse sentido, dá-se origem a um novo termo, como veremos a seguir.

5.1.2 Byte

O "byte é a menor unidade de armazenamento utilizada pelos computadores" (TEDESCO, 2020). (GOGONI, 2019) afirma que "um byte (símbolo B) ou octeto, é um pacote que agrupa oito bits". Assim, o byte é o termo binário, enquanto bit é o dígito binário. Nesse sentido, temos a seguinte equivalência que nos diz que 1 byte equivale a 8 bits agrupados.

$$1 \text{ byte} = 8 \text{ bits}$$

(CANDIDO, 2013, p.5) acrescenta que, na utilização do termo byte como capacidade de armazenamento em uma memória de computadores, verificamos a inclusão de caracteres, tais como K(Kilo), M(Mega), G(Giga), etc.

Nos computadores, onde todas as indicações numéricas referem-se à potências de 2, podemos resumir algumas dessas conversões de armazenamento na seguinte tabela:

Figura 30 – Grandezas usadas para abreviar valores em computação

GRANDEZAS USADAS PARA ABREVIAR VALORES EM COMPUTAÇÃO			
VALORES			
UNIDADE	**POTÊNCIA DE 2**	**Bytes**	**bits**
1 Byte	8 bits = 2^0	1	8 * 2^0
1 KB (1 Kilo bit)	1024 Bytes = 2^{10}	1024	8 * 2^{10}
1 MB (1 Mega bit)	1024 KB = 2^{20}	1.048.576	8 * 2^{20}
1 GB (1 Giga bit)	1024 MB = 2^{30}	1.073.741.824	8 * 2^{30}
1 TB (1 Tera bit)	1024 GB = 2^{40}	1.099.511.627.776	8 * 2^{40}
1 PB (1 Peta bit)	1024 TB = 2^{50}	1.125.899.906.843.624	8 * 2^{50}
1 EB (1 Exa bit)	1024 PB = 2^{60}	1.152921.504.607.870.976	8 * 2^{60}
1 ZB (1 Zeta bit)	1024 ZB = 2^{70}	1.180.591.620.718.458.879.424	8 * 2^{70}
1 YB (1 Yota bit)	1024 YB = 2^{80}	1.208.925.819.615.701.892.530.176	8 * 2^{80}

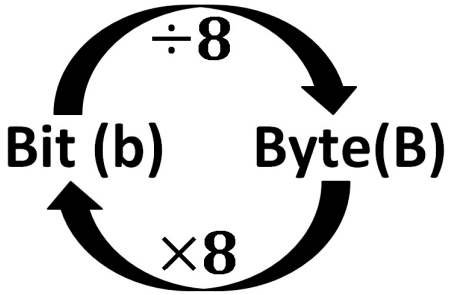
Fonte: (CANDIDO, 2013)

Ainda segundo (CANDIDO, 2013, p.4),

> como os principais códigos de representação de caracteres utilizam grupos de 8 bits por caractere, os conceitos de byte e caractere tornam-se semelhantes e, as palavras, quase sinônimas. O byte, pode representar um caractere no computador, não tendo a finalidade de representar qualquer tipo de informação, sendo tão somente uma unidade de armazenamento e transferência.

A conversão entre bits e bytes é realizada através de multiplicações e divisões por 8, utlizando o seguinte esquema:

Figura 31 – Conversão de Bit para Byte



Fonte: (CANDIDO, 2013)

Assim, a figura 31 nos diz que para converter um determinado valor de bit para byte, divide-se por 8, e reciprocamente, isto é, de byte para bit, multiplica-se pelo mesmo número, como é feito no exemplo a seguir:

Exemplo 22.

$$800b = 100B \quad pois \quad 800 \div 8 = 100.$$

$$15B = 120b \quad pois \quad 15 \times 8 = 120.$$

Na prática, encontramos o uso de bytes, entre outras aplicações, num *Pen Drive*, onde tem-se uma determinada capacidade de armazenamento seguida de GB, ($32GB$, por exemplo), onde o G significa giga e o B, byte.

Já na conexão de internet, encontramos uma utilização do conceito de bit, onde tempos, por exemplo, uma conexão de $100Mbps$, que significa 100 mega bits por segundo.

(TEDESCO, 2020) afirma que

> Existe uma confusão bastante comum entre os consumidores de internet em relação a unidade de medida da velocidade de transferência. Quando você contrata aquela internet de fibra óptica de $100MB$, que você lê que poderá baixar a "100 mega por segundo", pode parecer que será possível baixar um arquivo de 100 mega bytes em 1 segundo, mas isso não é verdade. Velocidade de transferência é medida em bits por segundo e não em bytes. Portanto, um download a $100Mbps$ significa 100 mega bits por segundo e não 100 mega bytes. Para se chegar no valor em mega bytes, é preciso dividir por 8. Ou seja, quando você baixa a $100Mbps$, você está na prática baixando $\frac{100}{8} = 12,5$ mega bytes por segundo.

5.2 Os Processadores

Também chamado de Unidade Central de Processamentos, do inglês *Central Processing Unit* (CPU ou UPC), o processador é uma poderosa máquina de calcular com

números binários e vale destacar que seu uso não se restringe apenas a computadores. Equipamentos como celulares, videogames, smartphones, tablets, etc, também precisam de processadores para trabalhar, sendo que nenhum dispositivo eletrônico consegue funcionar sem a famosa CPU. (ARRUDA, 2011)

Nos computadores, ele é o chip responsável pela execução de cálculos, aceleração, endereçamento e também por tomar todas as decisões lógicas que basicamente resultam em todas as tarefas do computador, por este motivo, o processador é conhecido como o cérebro do computador.

Em relação a este equipamento, (SOUZA, 2012, p. 1) afirma que

> esse chip sofreu transformações tecnológicas ao longo dos anos, proporcionando aos computadores um aumento considerável em seu poder computacional e na sua flexibilidade de uso. Paralelamente à evolução das CPU's, os computadores passaram a ser utilizados por um número cada vez maior de pessoas, pois, à medida que as máquinas passaram a ter uma alta demanda, o preço sofreu considerável redução, sendo essa uma tendência seguida até os dias atuais. Neste processo evolutivo dos processadores, nada mais é que um pedaço de silício com inúmeros condutores complexamente ligados a dimensões microscópicas.

Essa evolução ao longo do tempo fez com que estes dispositivos deixassem de ser programados com números decimais e passassem a usar a base binária como linguagem padrão, bem como evoluíssem no sentido de deixarem de ser mecânicos e lentos e tornassem rápidos e de fácil acesso, com a utilização de transistores chaveados no lugar de relés e válvulas, que eram complicadas e consumiam uma grande quantidade de energia. (SOUZA, 2012)

Figura 32 – Processador moderno

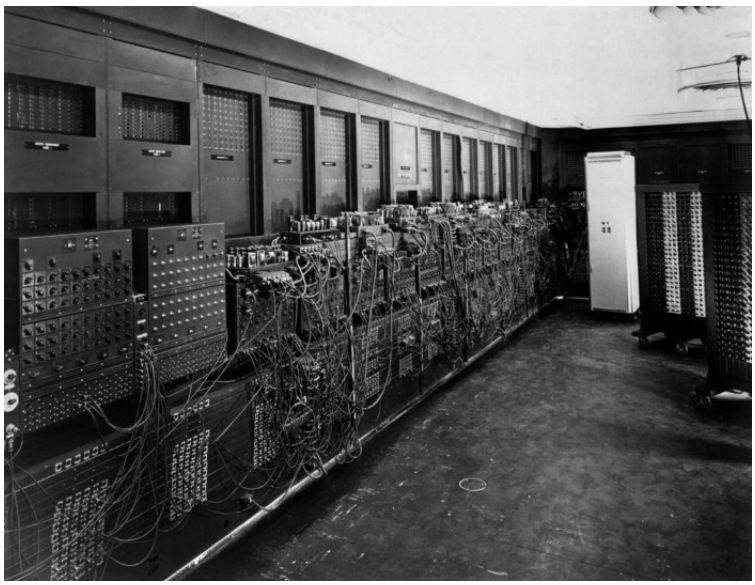
Fonte: https://www.reibatuta.com.br/processador-intel-core-i5-2400-3-10-ghz-lga-1155-oem

5.2.1 Contexto Histórico

A principal diferença entre os primeiros computadores e os atuais é que os antigos não eram capazes de armazenar dados. Podemos destacar o computador ENIAC (*Electronic Numerical Integrator Analyzer and Computer*), que teve seu desenvolvimento iniciado em 1943 pelos cientistas norte americando John Eckert e John Mauchly, entrando em funcionamento em 1946, sendo que, toda vez que era necessário realizar uma tarefa diferente, ele tinha sua estrutura física modificada, onde cabos eram repostos e chaves ligadas ou desligadas para que uma nova função pudesse ser executada.

Vale destacar que, no projeto inicial, tinha-se um plano de armazenamento de softwares em seu interior. No entanto, para agilizar o trabalho da construção da máquina, a ideia acabou ficando para trás.

Figura 33 – Computador ENIAC



Fonte: (SALES, 2018)

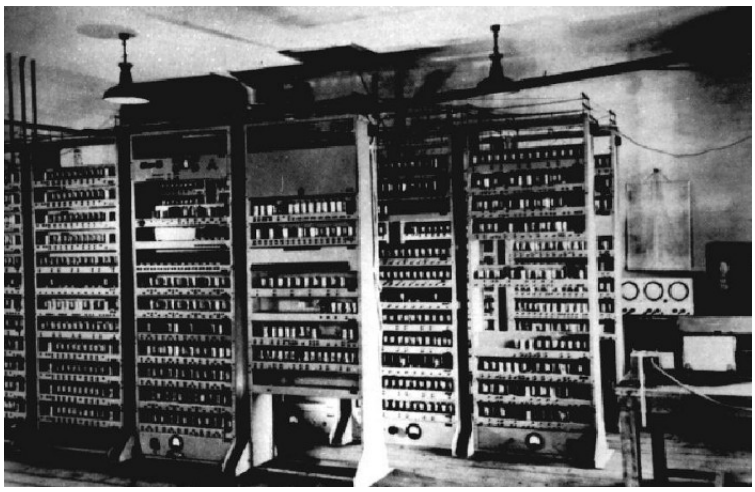
Este computador tinha capacidade de processamento de 5000 operações por segundo, sendo todas decimais, e possuia cerca de 1800 válvulas, que baseavam-se no princípio termiônico, utilizando o fluxo de elétrons no vácuo. O ENIAC foi utilizado para propósitos militares em cálculos de trajetórias de mísseis e codificação de mensagens secretas; era extremamente caro, consumia muita energia e ocupava muito espaço. Na verdade, suas funções assimilavam-se as que uma calculadora simples hoje realiza.

Figura 34 – Válvula termiônica

Fonte: https://pt.wikipedia.org/wiki/VC3A1lvula_termiC3B4nica

Outro computador dos primórdios que marcou foi o *Electronic Discrete Variable Automatic Computer* - EDVAC, que foi projetado para usar códigos binários e manter programas armazenados na memória. Sendo construído pela *Ballistic Research Laboratory U.S. Universidade da Pensilvânia* com base na arquitetura dos matemáticos John Eckert, John Mauchly e Jonh von Neumann entre os anos de 1946 e 1949, o EDVAC foi pensado buscando resolver problemas encontrados no ENIAC.

Figura 35 – Computador EDVAC



Fonte: (SALES, 2018)

De acordo com (SALES, 2018)

> o computador foi concebido para ser a adição binária, subtração e multiplicação, divisão programada e automática. Também possuia um verificador automático para até mil palavras. [...] Fisicamente, o computador foi construído pelos seguintes componentes: um leitor gravador, uma unidade de controle com o osciloscópio, uma unidade para receber instruções de controle e de memória e encaminhá-los para outras unidades, uma unidade computacional para realizar operações aritméticas um par de números em um tempo e mantê-los na memória após confirmação com outra unidade idêntica, um cronômetro e uma unidade de memória dual.

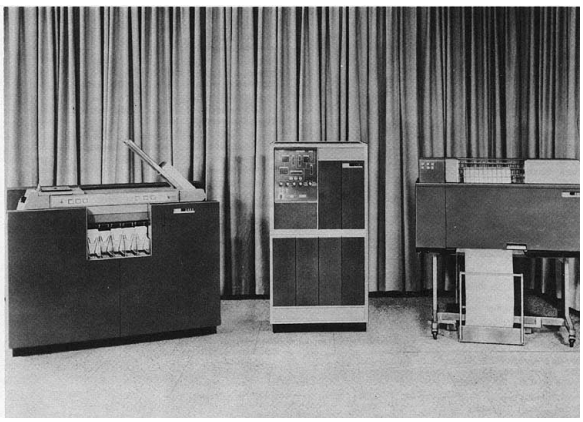
No EDVAC, o programa ou software que estava sendo executado poderia ser modificado sem que fosse necessário alterações físicas no computador. Essa foi a ideia que impulsionou a criação de uma unidade central de processamentos e deu origem aos modelos primitivos de processadores. Essa foi a ideia que marcou o ínicio da era dos computadores modernos.

Esses equipamentos baseados em válvulas são chamados de Computadores de Primeira Geração. No período que ficaram em uso, provaram ser altamente confiáveis e produtivos, no entanto, não muito duradoures, quebrando, geralmente, após poucas horas de uso.

Os equipamentos de segunda geração tem a válvula substituída por transistores, que tinham um tamanho menor, consumiam menos energia e realizavam a mesma tarefa com maior velocidade. O primeiro deles "foi criado em dezembro de 1947 pelos pesquisadores da empresa *Bell Laboratory*, anunciando uma nova era da eletrônica." (ALMEIDA, 2012, p. 2)

Nessa geração, temos como destaque o computador da IBM denominado 1401, que entrou em funcionamento no ano de 1959.

Figura 36 – Computador IBM 1401



Fonte: https://pt.wikipedia.org/wiki/IBM_1401

Outra grande evolução veio durante os anos 60 com a tecnologia dos circuitos integrados, que são compostos por resistores, transistores e capacitores miniaturizados e

montados num único chip. Para colocá-los em prática, a IBM desenvolveu um computador virtual conhecido por System/360, ou, simplesmente, S/360. Podemos pensar nesse sistema como um conjunto de instruções e capacidades que todos os computadores da família S/360 teriam em comum. (ARRUDA, 2011)

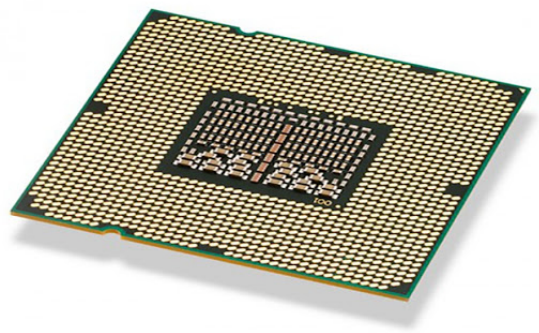
Com isso, programas seriam compatíveis entre todos esses modelos e não mais dependentes de máquinas. Essa evolução marcou a terceira geração de computadores.

Figura 37 – Computador System/360

Fonte: http://denglecomp.blogspot.com/2009/03/360-ibm-primeira-linha-de-produto.html

A quarta e atual geração começou na década de 70 e usa o mesmo argumento da anterior, mas ainda mais desenvolvido, sobretudo no que se refere à diminuição do tamanho dos chips, implicando o surgimento dos microprocessadores e dos microcomputadores.

Figura 38 – Microprocessador



Fonte: http://www.anisio.eti.br/index.php/hardware-menuvertical/item/32-microprocessador

(ALMEIDA, 2012) afirma que o primeiro microprocessador produzido no mundo foi o Intel 4004, desenvolvido no ano de 1971, pelos cientistas Federico Faggin, Ted

Hoff e Stanley Mazor. Até então, os dispostivos eletrônicos possuiam diversos chips separados para controle de teclado, display, impressora, etc, e o Intel 4004 continha todas esses acessórios interligados por um único chip. Com 4 bits e cerca de 2300 transistores, tinha tanto poder quanto o ENIAC, que ocupava mais de $900m^3$, com suas quase 18.000 válvulas.

A Intel e a ADM são as duas grandes empresas que desenvolvem esses equipamentos desde a criação dos primeiros deles, sendo que, com o passar dos anos, houve grandes evoluções nos processadores, que diminuíram de tamanho, ficaram mais rápidos e tiveram um grande aumento na quantidade de bits e de transistores, bem como a introdução de núcleos, que possibilitam ainda mais resultados positivos. Atualmente, o processador mais desenvolvido em termos de suas funcionalidades é o Intel Core $i9 - 10900$.

5.2.2 Funcionamento

(FáVERO, 2011, p. 57) ressalta que o processador "é resposável pelo processamento e execução de programas armazenados na memória principal, buscando instruções, examinando-as e, então, executando uma após a outra." Na verdade, este equipamento executa três funções básicas: receber, processar e fornecer dados de saída. É importante chamar atenção para o fato de que "eles processam apenas a linguagem booleana" (SOUZA, 2012, p. 3), já estudada no capítulo anterior. Assim, tudo que o processador entende é apenas 0 e 1.

O processador é composto de três partes: uma unidade aritmética e lógica (ULA), que é a unidade central, responsável por executar as operações aritméticas e lógicas; uma unidade de controle (UC), responsável por armazenar a posição de memória que contém a instrução que o computador está executando; e os registradores, que são elementos de armazenamento temporário. Num trabalho coletivo, a UC irá informar à ULA qual operação e informação ela deverá executar, e, em seguida, transfere esse resultado de volta para o local apropriado do registrador. Este ciclo repete-se para toda ação feita no computador. (MEYER, 2015)

Chamaremos a atenção para a ULA, que é

> um aglomerado de circuitos lógicos e componentes eletrônicos simples que, integrados, realizam as operações aritméticas e lógicas. São exemplos de operações executadas pela ULA: soma, multiplicação, operações lógicas (AND, OR, NOT, XOR, entre outras), incremento, decremento e operação complemento). (FáVERO, 2011, p. 60)

Já vimos que os computadores e demais equipamentos eletrônicos apenas interpretam 0 e 1, isto é, o sistema binário, e cabe aos processadores fazer uma espécie de tradução da nossa linguagem para binária, e vice-versa. Para que ele consiga fazer

isso, são utilizados transistores, onde a quantidade deles num processador implica diretamente na velocidade de execução das funções do equipamento.

Figura 39 – Transistor

Fonte: https://www.vidadesilicio.com.br/transistor-tip122-npn

(MEYER, 2015) chama atenção para as três "perninhas" que tem no transistor, como no da figura 39, onde uma delas é para base, outra é o coletor e a terceira é o emissor de dados. Quando a base é carregada, diz-se que ela tem valor 1, o que permite que a corrente flua para o coletor. O inverso acontece quando a base não tem carga (valor 0), onde a corrente faz o caminho contrário, fluindo para o emissor. Neste estado, o transistor funcionará como um interruptor fechado.

Assim, os estados 0 e 1 são formados a partir de correntes elétricas que geram, ou não, cargas para os pinos dos transistores.

Outro fator que é importante destacar é que, através de conexões de diferentes transistores e pinos, as ligações lógicas acontecem, onde o computador irá interpretar os comando lógicos "e", "ou" e "não", entre outros. (MEYER, 2015)

Portanto, concluímos que os processadores funcionam como uma espécie de conversor de dados da nossa linguagem para a binária, e vice-versa, a partir do recebimento ou não de correntes elétricas, que podem ser compreendidas como 1 ou 0, respectivamente, além de realizarem todos os cálculos necessário para o bom funcionamento do computadores e dos demais equipamentos eletrônicos.

5.3 O Código ASCII

Já vimos que tudo que os computadores compreendem são zeros e uns e que a conversão da nossa linguagem para binária é feita através de processadores. No entanto, essa conversão é feita seguindo padrões, onde, por exemplo, "para se codificar um texto, é necessário que se adote um valor binário para cada caractere, letra ou número do alfabeto." (LIMA, 2020)

Um desses padrões, que é muito utilizado em todo mundo, é o código ASCII [lê-se áski], que segundo (KARASINSKI, 2009)

> é uma sigla para *American Standard Code for Information Interchange* (Código Padrão Norte-americano para Intercâmbio de Informações). Esse código foi proposto por Robert W. Bemer, visando padronizar os códigos para caracteres alfa-numéricos (letras, sinais, números e acentos).

O intuito principal desse código é padronizar a forma de programação dos computadores, no sentido de haver uma linguagem comum a todos, visto que, até o ano de 1960, cada máquina utilizava uma regra diferente para programação, o que dificultava a comunicação entre elas.

O código ASCII utiliza apenas 7 bits para cada caractere, o que implica a existência de 128 possíveis combinações, como visto na Tabela 25. Assim, com esses bits é possível escrever 128 caracteres (de 0 a 127), onde 33 deles são sinais de controle (caracteres não imprimíveis) e 95 são utilizados para representar sinais gráficos, que correspondem a letras, números, sinais de pontuação, entre outros (caracteres imprimíveis). Os caracteres não imprimíveis são aqueles que não produzem, necessariamente, um caractere na tela no computador, e sim executa uma determinada função. Como o código é antigo, alguns sinais não são mais utilizados (SUGAI, 2015).

No entanto, por convenção, cada caractere é representado por 8 bits, isto é, 1 byte, sendo que o primeiro deles (dígito mais significativo a esquerda) é sempre 0, sendo, portanto, considerado um bit não utilizado. Mais adiante entenderemos o porquê dessa adoção de 8 bits.

No estudo desse código, é importante sempre ter conhecimento da Tabela ASCII, que mostra qual será o valor binário de cada caractere existente, conforme encontra-se no Anexo A deste trabalho.

É importante chamar atenção para o fato de que o código ASCII segue a ordem do alfabeto e uma sequência crescente de números binários.

Vale destacar que esse código foi concebido para língua inglesa, por isso ele não contém caracteres acentuados. Para codificar esse tipo de caractere e outros especiais, como o cedilha (Ç), foi necessário recorrer a alterações no código inicial, com a inclusão de mais 1 bit, totalizando 8 bits significativos, que equivale a 1 byte (perceba que a modificação consiste na mudança do primeiro bit, que deixa de ser um bit não utilizado e passa a ser utilizado, isto é, 1).

Esse "novo" código, que é a união do inicial acrescentado de 1 bit e passa a ter 256 (0 a 255) caracteres, é chamado de Código ASCII Completo, onde, de forma análoga ao inicial, ele fornece a cada caractere (letras, números, marcas de pontuação e outros caracteres, incluindo os acentuados) um valor binário específico. (MUXFELDT, 2017)

Os 128 bytes acrescidos ao código inicial (caracteres de 128 a 255) dão origem a Tabela ASCII estendida, que encontra-se no Anexo B desta pesquisa.

A união da Tabela ASCII original (caracteres de 0 a 127) com a estendida (caracteres de 128 a 255) é chamada de Tabela ASCII Completa.

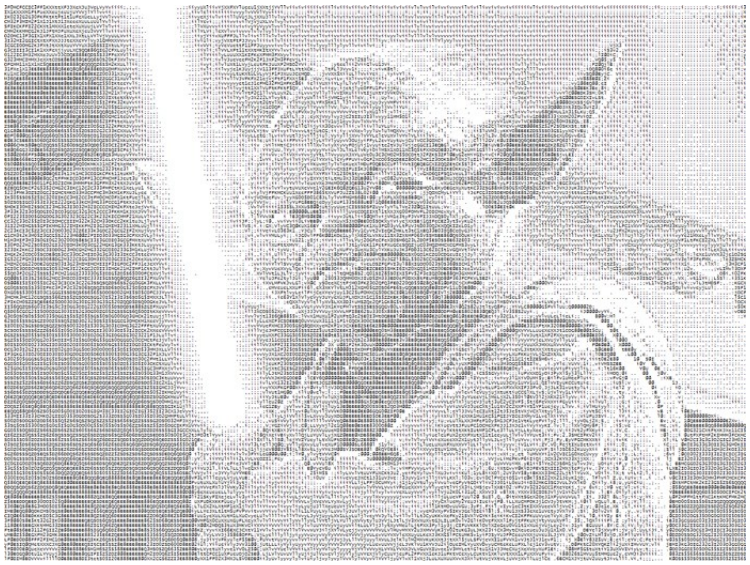
Vale destacar que há diferentes variações na tabela ASCII de 8 bits, sendo que a apresentada neste trabalho está de acordo com o Windows-1252 (CP-1252), que é um superconjunto da ISO 8859-1, também chamado de ISO Latin-1. A partir dela, é possível escrever textos em quase todos os idiomas, visto que o Windows-1252 é um dos codificadores de caracteres mais utilizados no mundo. (FERNANDES, 2020)

5.3.1 Arte ASCII

Através do código ASCII, também é possível criar desenhos. Assim, de acordo com (KARASINSKI, 2009), a chamada arte ASCII, do termo original *ASCII art* "é baseada justamente no uso de caracteres para criar desenhos e mensagens. Ela é bastante antiga, pois sempre foi utilizada nos computadores. A arte pode ser construída da maneira que o artista quiser: em preto-e-branco ou colorida."

Logo, esse tipo de arte utiliza apenas os caracteres disponíveis no código ASCII para expressão artística. Um exemplo de uma delas segue na figura 40.

Figura 40 – Arte ASCII com Mestre Yoda, de Star Wars



Fonte: (SUGAI, 2015)

Vale salientar que essa manifestação artística é compatível com todos os computadores que utilizem um sistema de caracteres e que há vários artistas que dedicam-se à criação de artes ASCII.

5.4 Unicode e Padrão UTF-8

O Unicode é um sistema de codificação de caracteres desenvolvido em 1991, que permite representar qualquer caractere por um código, geralmente, em 16 bits, independente de qualquer sistema operacional ou linguagem de programação que o computador estiver utilizando. Ele reúne, assim, quase todos os alfabetos existentes e é compatível com o código ASCII. (MUXFELDT, 2017)

É importante acrescentar que esse código contempla caracteres a mais do que foi apresentado nas tabelas dos Anexos A e B, como, por exemplo, símbolos utilizados na escrita de países orientais.

Nesse contexto, o Unicode, como o próprio nome sugere, é uma espécie de código único para computadores, que é adotado mundialmente e visa padronizar a linguagem utilizada pelas máquinas. Para isto, ele fornece um número diferente para cada caractere, seja ele letras do nosso alfabeto ou símbolos utilizados por outros idiomas, números, sinais de pontuação, etc.

No padrão Unicode, a notação é feita através da utilização do prefixo $U+$, seguido de números escritos na base hexadecimal (base 16).

> O alfabeto Unicode tem mais de 1 milhão caracteres. Portanto, o código de cada caractere precisaria de pelo menos 3 bytes se usássemos notação binária. Usar um número fixo de bytes por caractere não seria eficiente, já que 1 byte é suficiente para codificar os caracteres mais comuns. A solução é recorrer a um código multibyte, que emprega um número variável de bytes por caractere: alguns caracteres usam 1 byte, outros usam 2 bytes, e assim por diante. O código multibyte mais usado é conhecido como UTF-8. Ele associa uma sequência de 1 a 4 bytes (8 a 32 bits) com cada caractere Unicode. Os primeiros 128 caracteres usam o velho e bom código ASCII de 1 byte por caractere. Os demais caracteres têm um código mais complexo. (FEOFILOFF, 2018)

Nesse sentido, surge o UTF-8, criado em 1992 por Ken Thompson, que pode ter de 1 a 4 bytes e é o mais utilizado. No entanto, ainda existem o padrão UTF-16 (2 ou 4 bytes) e o UTF-32 (4 bytes). A definição de qual é mais eficiente depende diretamente do contexto em que se está inserido, porém, as diferenças entre eles são mínimas, por isso, em geral, usa-se o padrão UTF-8.

Conforme acrescenta (CHICONI, 2020),

> em teoria, o Unicode é muito bom. Porém, na prática, a história é outra. Normalmente, em Unicode, um caractere usa 2 bytes. Em outras palavras, qualquer texto usa duas vezes mais espaço do que no ASCII. É um desperdício. Além disso, se tomarmos como exemplo um texto em português, a grande maioria dos caracteres só utiliza o código ASCII. São raros os caracteres que requerem Unicode.

Assim, ao digitar, por exemplo, um texto em UTF-8, este será, simultaneamente, digitado em ASCII e, caso haja um caracetere que não se encontre no ASCII, somente esse fará uso do padrão UTF-8 para ser convertido.

A lista de alguns os códigos utilizados pelo padrão UTF-8 encontra-se no Anexo C.

Vale acrescentar que UTF é a sigla utilizada para *Unicode Transformation Format*, cuja tradução é Formato de Transformação Unicode.

Nesse padrão, teríamos, por exemplo, as seguintes correspondências entre os caracteres e as notações do Unicode:

$$A\text{: } U+0041$$

$$a\text{: } U+0061$$

$$5\text{: } U+0035$$

$$=\text{: } U+003D.$$

5.5 Aplicações

Nesta seção iremos apresentar como, de fato, os comandos inseridos nos computadores são processados, isto é, codificados e descodificados, resultando na operação correta.

Como já foi mencionado na Seção 5.2, o equipamento responsável por realizar esta tarefa é o processador ou, simplemente, CPU, que faz a conversão de dados da nossa linguagem para binária e vice-versa, possibilitando, assim, uma comunicação rápida, fácil e eficiente entre humanos e máquinas.

Mas como isso acontece? A resposta é simples: através de zeros e uns e baseado em um dos padrões mencionados nas Seções 5.3 e 5.4. Vale salientar, no entanto, que os 0 e 1 não significam necessariamente números, mas sim estados do sistema, que funcionam e interpretam comandos através de correntes elétricas ou ausência delas que, por sua vez, faz a comunicação direta com os processadores através da linguagem da álgebra booleana.

Nesse contexto, ao digitar algum caractere do teclado do computador (seja ele alguma letra, número, sinal de pontuação ou comando), a tecla pressionada irá enviar pulsos elétricos para o computador, isto é, sequências de zeros e uns (sequências de bits), e, a partir daí, o processador vai localizá-la de acordo com o código ASCII ou com o padrão UTF-8 e, então, irá compreender qual foi a tecla pressionada, passará a informação para o computador, que, por sua vez, dará como resposta a inserção do caractere na tela ou execução do comando desejado. Vale salientar que tudo isso é feito em fração de segundos, dependendo da potência do processador, sendo que, quanto

mais desenvolvido ele for, menos tempo gastará para executar determinada operação. Este processo é repetido inúmeras vezes durante o uso da máquina.

Assim, por exemplo, o computador não entende a letra A. Para ele, esta letra será apenas um desenho impresso na tela quando ele receber uma determinada sequência numérica de bits.

Exemplo 23. *Os bytes a seguir, apresentados em números binários, representam uma palavra da forma original que foi armazenado na memória de um computador. Sabendo que cada byte é um código ASCII, qual é a palavra?*

0101 0000 0101 0010 0100 1111 0100 0110 0100 1101 0100 0001 0101 0100

Para decifrar qual a palavra, basta verificar, no Anexo A, qual letra corresponde ao binário apresentado. Fazendo isso, constatamos que a palavra que está escrita em binário é PROFMAT.

Exemplo 24. *Usando a Tabela ASCII completa, qual seria a combinação de bits que o computador receberia ao ser digitado a frase* ***Números Binários****?*

De posse da tabela (que encontra-se nos Anexos A e B), a codificação binária para o termo Números Binários seria:

0100 1110 1010 0011 0110 1101 0110 0101 0111 0010 0110 1111 0111 0011

0010 0000 0100 0010 0110 1001 0110 1110 1010 0000 0111 0010 0110 1001

0110 1111 0111 0011

6 Conclusões

Geralmente, no nosso cotidiano, utilizamos o sistema de numeração decimal, isto é, na base 10, que, como o próprio nome sugere, é composto de 10 algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Ele consegue suprir quase todas as nossas necessidades de contagem, no entanto, este sistema não é único.

Existem outros sistemas de numeração, que também são utilizados por nós, muitas vezes sem percebermos. Por exemplo, ao ver as horas no relógio, nos deparamos com um sistema na base 60 (sexagesimal), ou ao ir ao mercado comprar uma dúzia de algum produto, onde encontramos o uso da base 12 (duodecimal), entre outros. Pela revisão de literatura, ficou constatado que a utilização de diferentes bases de numeração é antiga, sendo encontrada desde os primeiros indícios da utilização da matemática e varia de acordo com cada civilização.

Ainda nesse contexto, podemos citar o sistema binário (base 2), onde os únicos algarismos existentes são o 0 e o 1, que começou a ser utilizado por volta do século III a.C. e continua sendo até os dias atuais, tendo uma de suas principais aplicações voltada para a programação dos equipamentos digitais.

Neste trabalho, que foi desenvolvido através de uma pesquisa qualitativa e de uma revisão bibliográfica, foi dissertado sobre como os números binários são utilizados dentro da computação, onde foi possível constatar como, de fato, esses números são úteis dentro do nosso cotidiano e que, mesmo sem perceber, estamos diariamente em contato com eles.

Os equipamentos eletrônicos, onde enfatizamos os computadores atuais, são capazes de fazer coisas incríveis, mas um fato que chama muito a nossa atenção é que eles só entendem 0 e 1, ou seja, a linguagem binária, no sentido de que toda programação utilizada neles é feita através desses dois algarismos. Mas nem sempre foi assim. Na invenção do primeiro computador, por exemplo, a linguagem utilizada era a decimal, no entanto, com o passar do tempo, percebeu-se que utilizar um base menor, na verdade, a menor possível (base binária), tornaria estes equipamentos mais rápidos e eficazes, visto que todo trabalho seria feito em cima de apenas dois algarismos. Essa ideia deu muito certo e impulsinou nas máquinas potentes que temos hoje em dia.

Além dos números binários, para o bom funcionamento dos computadores, ele utilizada a álgebra de Boole, que consiste na utilização de letras para traduzir o pensamento humano. Um dos fatores que chama a atenção nesssa álgebra é que ela também só faz uso dos algarismos 0 e 1. Munidas de definições, axiomas, propriedades e teoremas, ela apresenta grande semelhança com a tradicional, como foi visto no decorrer desta pesquisa.

Embora seja fundamentada com base na linguagem binária, nesta álgebra os números 0 e 1 não exprimem quantidades, mas sim estados do sistema, que são transmitidos para os computadores através de pulsos elétricos, onde o 1 indica a passagem da corrente e 0, a ausência dela. Nos equipamentos eletrônicos, tudo isso é feito através de operadores e portas lógicas, que, quando interligadas, dão origem aos circuitos lógicos.

Ideia semelhante é utilizada nos circuitos elétricos e na programação do buscador "Google", que utilizam, sobretudo, os conceitos dos conectores lógicos. Sem esta álgebra, os computadores e outros equipamentos digitais não teriam a mesma estrutura e funcionalidade que têm atualmente.

Como os computadores só entendem 0 e 1, foi necessário a criação de um equipamento que fizesse essa "tradução" da nossa linguagem para binária, e vice-versa. A este equipamento que é considerado o cérebro dos computadores, dá-se o nome de processador.

Também chamado de CPU, ele sofreu grandes evoluções ao longo do tempo, que impulsionaram grandes mudanças nos equipamentos eletrônicos, tanto físicas, quanto em termos de programação e funcionalidades, uma vez que sua principal função é processar dados e executá-los.

Vale lembrar que estes equipamentos também só entendem 0 e 1 e fazem uso da ideia dos estados que esses números representam, sugerida pela álgebra de Boole, isto é, eles interpretam os pulsos elétricos ou a ausência deles para compreender se o número (bit - dígito binário) em questão é 0 ou 1. Esta tarefa é repetida inúmeras vezes e, quando formam grupos de 8 bits, dão origem a um byte (termo binário).

Com a evolução tecnológica, os computadores foram ganhando cada vez mais espaço em todo mundo. Este fato impulsionou a criação de uma linguagem padrão para que fosse utilizados em todos os computadores, tendo em vista que, até então, cada um tinha sua linguagem de programação específica, fato que dificultava a comunicação entre eles. Nessa visão, surgiu o código ASCII, com 7 bits, sendo que, em pouco tempo, foi necessário ampliá-lo para 8 bits e, ainda não sendo suficiente para contemplar todos os idiomas existentes, foi desenvolvido o Unicode, isto é, um código único, sendo o padrão UTF-8 o mais utilizado deles.

O código ASCII, que também utiliza a linguagem binária, consiste na atribuição de um byte, isto é, uma sequência de 8 bits, para cada caractere utilizado pelo computador, seja ele letras, números, sinais de pontuação, etc, ou, simplesmente, um comando. O padrão UTF-8, que é uma expansão do código ASCII, porém, ainda assim, compatível com ele, tem funcionamento análogo ao do código, no entanto, utiliza a base hexadecimal (base 16), tendo em vista que possui mais de um milhão de caracteres e, se continuasse utilizando os binários, os números ficariam muito extensos, com muitos dígitos.

Na prática, cabe aos processadores desempenhar essa tarefa por repetidas vezes e em fração de segundos, que consiste em identificar qual foi a tecla pressionada no teclado através da sequência de bits recebida, e enviá-la ao computador, que desenhará o caractere desejado na tela do computador ou executará o comando solicitado. Assim, cabe ao processador fazer essa "tradução", que possibilita nossa comunicação com as máquinas.

Por fim, e encerrando o texto, diante da pesquisa realizada e dos resultados obtidos, podemos constatar que os objetivos foram alcançados, tendo em vista que conseguimos mostrar como os números binários são aplicados à computação, atendendo, portanto, todas as nossas expectativas diante do tema central.

Referências

ABAR, C. *Noções de álgebra booleana*. 2004. Disponível em: <https://www4.pucsp.br/~logica/Booleana.htm>. Acesso em: 04 feb 2021. Citado 3 vezes nas páginas 31, 54 e 56.

ALMEIDA, R. B. Evolução dos processadores. *Instituto de Computação - Unicamp*, 2012. Citado 2 vezes nas páginas 82 e 83.

ARRUDA, F. *A História dos Processadores*. 2011. Disponível em: <https://www.tecmundo.com.br/historia/2157-a-historia-dos-processadores.htm#:~:text=Em%201945%2C%20a%20ideia%20de,da%20forma%20como%20os%20conhecemos.> Acesso em: 17 abr 2021. Citado 2 vezes nas páginas 79 e 83.

BARANAUSKAS, J. A. Álgebra de boole e simplificação de circuitos lógicos. *Depertamento de computação e matemática - FFCLRP - USP*, Universidade de São Paulo, 2012. Citado 3 vezes nas páginas 31, 53 e 54.

BOOLE, G. *The mathematical analysis of logic*. [S.l.]: Philosophical Library, 1847. Citado na página 30.

CANDIDO, C. Sistemas numéricos – bit / byte. 2013. Citado 3 vezes nas páginas 76, 77 e 78.

CHICONI, N. *O que é ASCII, Unicode e UTF-8?* 2020. Disponível em: <https://br.ccm.net/faq/9956-o-que-e-ascii-unicode-e-utf-8>. Acesso em: 03 mai 2021. Citado na página 88.

DIAS, C. M. C. Álgebra booleana e lógica digital uma aplicação da lógica matemática. *Revista Acadêmica*, Curitiba, p. 47–56, 1994. Citado na página 67.

DIDáTICO-PEDAGóGICAS, P. Os desafios da escola pública paranaense na perspectiva do professor pde. 2013. Citado na página 74.

EVES, H. *Introdução à história da matemática*. Campinas - SP: Unicamp, 2004. Citado na página 16.

FEOFILOFF, P. *Unicode e UTF-8*. 2018. Disponível em: <https://www.ime.usp.br/~pf/algoritmos/apend/unicode.html>. Acesso em: 15 abr 2021. Citado na página 88.

FERNANDES, H. M. *Código ASCII – Tabela ASCII Completa*. 2020. Disponível em: <https://marquesfernandes.com/desenvolvimento/codigo-ascii-tabela-ascii-completa/>. Acesso em: 30 abr 2021. Citado na página 87.

FRANZON, C. R. P. *A característica universal de leibniz: contextos, trajetórias e implicações*. Tese (Doutorado) — Universidade Estadual Paulista, Rio Claro - SP, 2015. Citado na página 24.

FáVERO, E. M. de B. Organização e arquitetura de computadores. *Universidade Tecnológica Federal do Paraná*, 2011. Citado na página 84.

GIRELLI, M.; BEZERRA, R. C. Matemática e raciocínio lógico: trabalhando e discutindo os jogos de boole. 2014. Citado na página 72.

GOGONI, R. *Bit ou Byte?* 2019. Disponível em: <https://tecnoblog.net/303263/bit-ou-byte/>. Acesso em: 15 abr 2021. Citado 3 vezes nas páginas 75, 76 e 77.

GÜNTZEL, J. L.; NASCIMENTO, F. A. do. Álgebra booleana e circuitos lógicos. *Introdução aos sistemas digitais*, 2001. Citado 13 vezes nas páginas 32, 34, 35, 45, 46, 47, 49, 52, 56, 59, 61, 63 e 64.

HIRATA, N. S. T. Álgebra booleana e aplicações. *Depto. de ciências da computação*, Universidade de São Paulo, 2005. Citado 7 vezes nas páginas 36, 41, 48, 50, 51, 52 e 63.

KARASINSKI, E. *O que é Código ASCII?* 2009. Disponível em: <https://www.tecmundo.com.br/imagem/1518-o-que-e-codigo-ascii.htm>. Acesso em: 30 abr 2021. Citado 2 vezes nas páginas 86 e 87.

LIMA, T. *Tabela ASCII*. 2020. Disponível em: <https://www.embarcados.com.br/tabela-ascii/>. Acesso em: 30 abr 2021. Citado na página 85.

LIPSCHUTZ, S.; LIPSON, M. L. *Theory and problems of discrate mathematics*. [S.l.]: Philosophical Library, 1976. Citado 4 vezes nas páginas 37, 42, 47 e 64.

LOPES, F. J. A. Leibniz e a aritmética binária. *Revista Brasileira de História da Matemática*, Sociedade Brasileira de História da Matemática, 2011. Citado na página 17.

LUNA, J. E. L. *O algoritmo do par binário*. Dissertação (Mestrado) — Universidade Federal de Alagoas, Maceió - AL, 2013. Citado na página 20.

MARTINES, V. M. *Base de numeração e o sistema binário*. Dissertação (Mestrado) — Universidade Federal da Grande Dourados, Dourados - MS, 2019. Citado 4 vezes nas páginas 16, 19, 23 e 24.

MELLO, A. M. Jogos boole: o desenvolvimento do raciocínio através de histórias lógicas. 2010. Citado 2 vezes nas páginas 30 e 73.

MEYER, M. *Como os processadores interpretam fisicamente os comandos?* 2015. Disponível em: <https://www.oficinadanet.com.br/post/13472-como-os-processadores-interpretam-fisicamente-os-comandos>. Acesso em: 18 abr 2021. Citado 2 vezes nas páginas 84 e 85.

MIYASCHITA, W. Y. *Sistemas de numeração: como funcionam e como são estruturados os números binários*. Bauru - SP: Unesp, 2002. Citado 2 vezes nas páginas 16 e 17.

MUXFELDT, P. *O código ASCII*. 2017. Disponível em: <https://br.ccm.net/contents/54-o-codigo-ascii>. Acesso em: 30 abr 2021. Citado 2 vezes nas páginas 86 e 88.

NEWS, B. *Como matemático inventou há mais de 150 anos a fórmula de buscas usada pelo Google*. 2015. Disponível em: <https://www.bbc.com/portuguese/noticias/2015/11/151102_boole_google_tg>. Acesso em: 20 feb 2021. Citado 2 vezes nas páginas 30 e 31.

OLIVEIRA, J. M. de. Álgebra booleana e suas utilizações. Seropédia - RJ, 2017. Citado 5 vezes nas páginas 36, 60, 67, 72 e 74.

RIZZATO, F. B.; RINALDI, B. L. *Álgebra: álgebra Booleana; métodos álgébricos para a solução de equações diferenciais*. 2005. Disponível em: <http://www.matematica.br/historia/boole.html#:~:text=Boole%20viu%20a%20l%C3%B3gica%20de,%2C%20via%20linguagens%20de%20programa%C3%A7%C3%A3o)>. Acesso em: 03 feb 2021. Citado na página 30.

SALES, G. *Evolução dos computadores*. 2018. Disponível em: <https://medium.com/@gustavosales086/evolu%C3%A7%C3%A3o-dos-computadores-bd87102e01cf>. Acesso em: 29 abr 2021. Citado 2 vezes nas páginas 80 e 81.

SANTOS, J. N.; SANTANA, J. R. O sistema de numeração decimal posicional e as operações fundamentais: explorando os algoritmos com a manipulação do material dourado. 2013. Citado na página 19.

SANTOS, R. *Você sabe a história, entende como funciona o Sistema Binário?* 2020. Disponível em: <https://www.portalgsti.com.br/2020/02/voce-sabe-a-historia-entende-como-funciona-o-sistema-binario.html>. Acesso em: 20 nov 2020. Citado na página 17.

SANTOS, V. R. dos; OLIVEIRA, C. G. de. A álgebra booleana presente nos circuitos lógicos. *Ciências exatas e tecnológicas*, v. 3, n. 3, p. 63–72, 2016. Citado na página 59.

SOUZA, M. L. de. Evolução dos processadores e seu futuro. *Univiersidade São Francisco*, 2012. Citado 2 vezes nas páginas 79 e 84.

SUGAI, A. *O que é o código ASCII e para que serve?* 2015. Disponível em: <https://www.techtudo.com.br/noticias/noticia/2015/02/o-que-e-o-codigo-ascii-e-para-que-serve-descubra.html>. Acesso em: 30 abr 2021. Citado 2 vezes nas páginas 86 e 87.

TEDESCO, K. *Bits, bytes e unidades de medida*. 2020. Disponível em: <https://www.treinaweb.com.br/blog/bits-bytes-e-unidades-de-medida/>. Acesso em: 12 abr 2021. Citado 3 vezes nas páginas 75, 77 e 78.

VIEIRA, F. M. S. Álgebra booleana. *Educ. Tecnol.*, v. 5, n. 1, p. 10–12, 2000. Citado 2 vezes nas páginas 32 e 59.

Anexos

ANEXO A – Tabela ASCII - 128 Códigos (7 Bits)

Tabela 26 – Tabela ASCII - Caracteres não imprimíveis

Caractere	Decimal	Hexadecimal	Binário	Comentário
NUL	00	00	0000 0000	Caractere Nulo
SOH	01	01	0000 0001	Começo de cabeçalho de transmissão
STX	02	02	0000 0010	Começo de texto
ETX	03	03	0000 0011	Fim de texto
EOT	04	04	0000 0100	Fim de transmissão
ENQ	05	05	0000 0101	Interroga
ACK	06	06	0000 0110	Confirmação
BEL	07	07	0000 0111	Sinal sonoro
BS	08	08	0000 1000	Volta um caractere
HT	09	09	0000 1001	Tabulação horizontal
LF	10	0A	0000 1010	Próxima linha
VT	11	0B	0000 1011	Tabulação vertical
FF	12	0C	0000 1100	Próxima página
CR	13	0D	0000 1101	Início da linha
SO	14	0E	0000 1110	Shift-out
SI	15	0F	0000 1111	Shift-in
DLE	16	10	0001 0000	Data link escape
D1	17	11	0001 0001	Controle de dispositivo
D2	18	12	0001 0010	Controle de dispositivo
D3	19	13	0001 0011	Controle de dispositivo
D4	20	14	0001 0100	Controle de dispositivo
NAK	21	15	0001 0101	Caractere Nulo
SYN	22	16	0001 0110	Caractere Nulo
ETB	23	17	0001 0111	Caractere Nulo
CAN	24	18	0001 1000	Caractere Nulo
EM	25	19	0001 1001	Caractere Nulo
SUB	26	1A	0001 1010	Caractere Nulo
ESC	27	1B	0001 1011	Caractere Nulo
FS	28	1C	0001 1100	Caractere Nulo
GS	29	1D	0001 1101	Caractere Nulo
RS	30	1E	0001 1110	Caractere Nulo
US	31	1F	0001 1111	Caractere Nulo

Fonte: https://repositorio.ufu.br/bitstream/123456789/14443/4/SFOLima4DISSPRT.pdf

Tabela 27 – Tabela ASCII - Caracteres imprimíveis

Caractere	Decimal	Hexadecimal	Binário
Espaço	32	20	0010 0000
!	33	21	0010 0001
"	34	22	0010 0010
#	35	23	0010 0011
$	36	24	0010 0100
%	37	25	0010 0101
&	38	26	0010 0110
’	39	27	0010 0111
(	40	28	0010 1000
)	41	29	0010 1001
*	42	2A	0010 1010
+	43	2B	0010 1011
,	44	2C	0010 1100
-	45	2D	0010 1101
.	46	2E	0010 1110
/	47	2F	0010 1111
0	48	30	0011 0000
1	49	31	0011 0001
2	50	32	0011 0010
3	51	33	0011 0011
4	52	34	0011 0100
5	53	35	0011 0101
6	54	36	0011 0110
7	55	37	0011 0111
8	56	38	0011 1000
9	57	39	0011 1001
:	58	3A	0011 1010
;	59	3B	0011 1011
<	60	3C	0011 1100
=	61	3D	0011 1101
>	62	3E	0011 1110
?	63	3F	0011 1111
@	64	40	0100 0000
A	65	41	0100 0001
B	66	42	0100 0010
C	67	43	0100 0011
D	68	44	0100 0100
E	69	45	0100 0101
F	70	46	0100 0110
G	71	47	0100 0111
H	72	48	0100 1000
I	73	49	0100 1001
J	74	4A	0100 1010
K	75	4B	0100 1011

Fonte: https://repositorio.ufu.br/bitstream/123456789/14443/4/SFOLima4DISSPRT.pdf

Tabela 28 – Tabela ASCII - Caracteres imprimíveis (Continuação)

Caractere	Decimal	Hexadecimal	Binário
L	76	4C	0100 1100
M	77	4D	0100 1101
N	78	4E	0100 1110
O	79	4F	0100 1111
P	80	50	0101 0000
Q	81	51	0101 0001
R	82	52	0101 0010
S	83	53	0101 0011
T	84	54	0101 0100
U	85	55	0101 0101
V	86	56	0101 0110
W	87	57	0101 0111
X	88	58	0101 1000
Y	89	59	0101 1001
Z	90	5A	0101 1010
[	91	5B	0101 1011
\	92	5C	0101 1100
]	93	5D	0101 1101
^	94	5E	0101 1110
_	95	5F	0101 1111
`	96	60	0110 0000
a	97	61	0110 0001
b	98	62	0110 0010
c	99	63	0110 0011
d	100	64	0110 0100
e	101	65	0110 0101
f	102	66	0110 0110
g	103	67	0110 0111
h	104	68	0110 1000
i	105	69	0110 1001
j	106	6A	0110 1010
k	107	6B	0110 1011
l	108	6C	0110 1100
m	109	6D	0110 1101
n	110	6E	0110 1110
o	111	6F	0110 1111
p	112	70	0111 0000
q	113	71	0111 0001
r	114	72	0111 0010
s	115	73	0111 0011
t	116	74	0111 0100
u	117	75	0111 0101
v	118	76	0111 0110
w	119	77	0111 0111
x	120	78	0111 1000

Fonte: https://repositorio.ufu.br/bitstream/123456789/14443/4/SFOLima4DISSPRT.pdf

Tabela 29 – Tabela ASCII - Caracteres imprimíveis (Continuação)

Caractere	Decimal	Hexadecimal	Binário
y	121	79	0111 1001
z	122	7A	0111 1010
{	123	7B	0111 1011
|	124	7C	0111 1100
}	125	7D	0111 1101
∽	126	7E	0111 1110
DELETE	127	7F	0111 1111

Fonte: https://repositorio.ufu.br/bitstream/123456789/14443/4/SFOLima4DISSPRT.pdf

ANEXO B – Tabela ASCII Estendida - 128 Códigos (8 Bits)

Figura 41 – Tabela ASCII Estendida

Decimal	Binário	Hexadecimal	Caractere
128	10000000	80	Ç
129	10000001	81	ü
130	10000010	82	é
131	10000011	83	â
132	10000100	84	ä
133	10000101	85	à
134	10000110	86	å
135	10000111	87	ç
136	10001000	88	ê
137	10001001	89	ë
138	10001010	8A	è
139	10001011	8B	ï
140	10001100	8C	î
141	10001101	8D	ì
142	10001110	8E	Ä
143	10001111	8F	Å
144	10010000	90	É
145	10010001	91	æ
146	10010010	92	Æ
147	10010011	93	ô
148	10010100	94	ö
149	10010101	95	ò
150	10010110	96	û
151	10010111	97	ù

Decimal	Binário	Hexadecimal	Caractere
152	10011000	98	ÿ
153	10011001	99	Ö
154	10011010	9A	Ü
155	10011011	9B	ø
156	10011100	9C	£
157	10011101	9D	Ø
158	10011110	9E	×
159	10011111	9F	ƒ
160	10100000	A0	á
161	10100001	A1	ù
162	10100010	A2	ó
163	10100011	A3	ú
164	10100100	A4	ñ
165	10100101	A5	Ñ
166	10100110	A6	ª
167	10100111	A7	º
168	10101000	A8	¿
169	10101001	A9	®
170	10101010	AA	¬
171	10101011	AB	½
172	10101100	AC	¼
173	10101101	AD	¡
174	10101110	AE	«
175	10101111	AF	»

Fonte: https://www.matematica.pt/util/resumos/tabela-ascii.php (Adaptada)

Figura 42 – Tabela ASCII Estendida

Decimal	Binário	Hexadecimal	Caractere
175	10101111	AF	»
176	10110000	B0	░
177	10110001	B1	▒
178	10110010	B2	▓
179	10110011	B3	│
180	10110100	B4	┤
181	10110101	B5	Á
182	10110110	B6	Â
183	10110111	B7	À
184	10111000	B8	©
185	10111001	B9	╣
186	10111010	BA	║
187	10111011	BB	╗
188	10111100	BC	╝
189	10111101	BD	¢
190	10111110	BE	¥
191	10111111	BF	┐
192	11000000	C0	└
193	11000001	C1	┴
194	11000010	C2	┬
195	11000011	C3	├
196	11000100	C4	─
197	11000101	C5	┼
198	11000110	C6	ã
199	11000111	C7	Ã

Decimal	Binário	Hexadecimal	Caractere
200	11001000	C8	╚
201	11001001	C9	╔
202	11001010	CA	╩
203	11001011	CB	╦
204	11001100	CC	╠
205	11001101	CD	═
206	11001110	CE	╬
207	11001111	CF	¤
208	11010000	D0	ð
209	11010001	D1	Ð
210	11010010	D2	Ê
211	11010011	D3	Ë
212	11010100	D4	È
213	11010101	D5	ı
214	11010110	D6	Í
215	11010111	D7	Î
216	11011000	D8	Ï
217	11011001	D9	┘
218	11011010	DA	┌
219	11011011	DB	█
220	11011100	DC	▄
221	11011101	DD	¦
222	11011110	DE	Ì
223	11011111	DF	▀
224	11100000	E0	Ó

Fonte: https://www.matematica.pt/util/resumos/tabela-ascii.php (Adaptada)

Figura 43 – Tabela ASCII Estendida

Decimal	Binário	Hexadecimal	Caractere
225	11100001	E1	ß
226	11100010	E2	Ô
227	11100011	E3	Ò
228	11100100	E4	õ
229	11100101	E5	Õ
230	11100110	E6	µ
231	11100111	E7	þ
232	11101000	E8	Þ
233	11101001	E9	Ú
234	11101010	EA	Û
235	11101011	EB	Ù
236	11101100	EC	ý
237	11101101	ED	Ý
238	11101110	EE	¯
239	11101111	EF	´
240	11110000	F0	
241	11110001	F1	±
242	11110010	F2	‗
243	11110011	F3	¾
244	11110100	F4	¶
245	11110101	F5	§
246	11110110	F6	÷
247	11110111	F7	¸

Decimal	Binário	Hexadecimal	Caractere
248	11111000	F8	°
249	11111001	F9	¨
250	11111010	FA	·
251	11111011	FB	¹
252	11111100	FC	³
253	11111101	FD	²
254	11111110	FE	■
255	11111111	FF	

Fonte: https://www.matematica.pt/util/resumos/tabela-ascii.php (Adaptada)

ANEXO C – Tabela Unicode - Padrão UTF-8

Segue algumas tabelas de alguns dos idiomas contemplados pelo Unicode. Para isto, utilizaremos a seguinte legenda:

Figura 44 – Legenda Unicode

Legend:			
Unicode 1.0	Unicode 1.0.1	Unicode 1.1	Unicode 2.0
Unicode 2.1	Unicode 3.0	Unicode 3.1	Unicode 3.2
Unicode 4.0	Unicode 4.1	Unicode 5.0	Unicode 5.1
Unicode 5.2	Unicode 6.0	Unicode 6.1	Unicode 6.2
Unicode 6.3	Unicode 7.0	Unicode 8.0	Unicode 9.0
Unicode 10.0	Unicode 11.0	Unicode 12.0	Unicode 12.1
Unicode 13.0			
Private Use	Surrogate	Reserved	Noncharacter

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 45 – Controles C0 e latim básico

C0 Controls and Basic Latin																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000x	[NUL]	[SOH]	[STX]	[ETX]	[EOT]	[ENQ]	[ACK]	[BEL]	[BS]	[HT]	[LF]	[VT]	[FF]	[CR]	[SO]	[SI]
001x	[DLE]	[DC1]	[DC2]	[DC3]	[DC4]	[NAK]	[SYN]	[ETB]	[CAN]	[EM]	[SUB]	[ESC]	[FS]	[GS]	[RS]	[US]
002x	[SP]	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	[DEL]

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 46 – Controles C1 e suplemento de Latin-1

C1 Controls and Latin-1 Supplement																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
008x	[XXX]	[XXX]	[BPH]	[NBH]	[IND]	[NEL]	[SSA]	[ESA]	[HTS]	[HTJ]	[VTS]	[PLD]	[PLU]	[RI]	[SS2]	[SS3]
009x	[DCS]	[PU1]	[PU2]	[STS]	[CCH]	[MW]	[SPA]	[EPA]	[SOS]	[XXX]	[SCI]	[CSI]	[ST]	[OSC]	[PM]	[APC]
00Ax	[NB SP]	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	[SHY -]	®	¯
00Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 47 – Latim Extended-A

Latin Extended-A																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
010x	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď
011x	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ
012x	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į
013x	İ	ı	IJ	ij	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ
014x	ŀ	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ŉ	Ŋ	ŋ	Ō	ō	Ŏ	ŏ
015x	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş
016x	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů
017x	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ

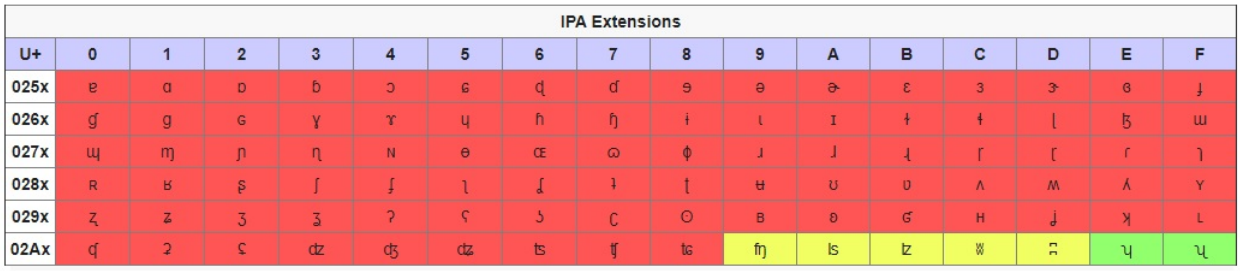
Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 48 – Latim Extended-B

Latin Extended-B																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
018x	ƀ	Ɓ	Ƃ	ƃ	Ƅ	ƅ	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə
019x	Ɛ	Ƒ	ƒ	Ɠ	ƙ	ƚ	ƛ	Ɯ	Ɲ	ƞ	Ɵ	Ơ	ơ	Ƣ	ƣ	Ƥ
01Ax	ƥ	Ʀ	Ƨ	ƨ	Ʃ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư	ư	Ʊ	Ʋ	Ƴ	ƴ
01Bx	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ	ǀ	ǁ	ǂ	ǃ	DŽ
01Cx	Dž	dž	LJ	Lj	lj	NJ	Nj	nj	Ǎ	ǎ	Ǐ	ǐ	Ǒ	ǒ	Ǔ	ǔ
01Dx	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ	Ǡ	ǡ	Ǣ	ǣ	Ǥ
01Ex	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ	ǰ	DZ	Dz	dz	Ǵ
01Fx	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ					
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
020x	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
021x	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
022x	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
023x	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ƚ	Ⱦ	ȿ
024x	ɀ	Ɂ	ɂ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

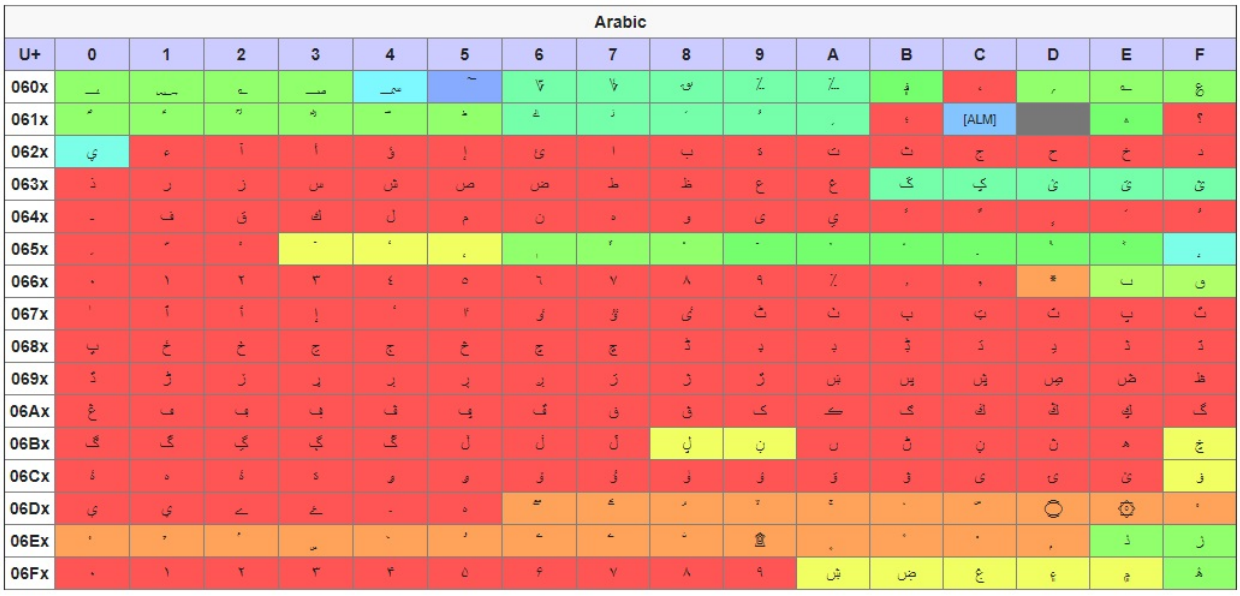
Figura 49 – Extensões IPA



IPA Extensions																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
025x	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ
026x	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ
027x	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ
028x	ʀ	ʁ	ʂ	ʃ	ʄ	ʅ	ʆ	ʇ	ʈ	ʉ	ʊ	ʋ	ʌ	ʍ	ʎ	ʏ
029x	ʐ	ʑ	ʒ	ʓ	ʔ	ʕ	ʖ	ʗ	ʘ	ʙ	ʚ	ʛ	ʜ	ʝ	ʞ	ʟ
02Ax	ʠ	ʡ	ʢ	ʣ	ʤ	ʥ	ʦ	ʧ	ʨ	ʩ	ʪ	ʫ	ʬ	ʭ	ʮ	ʯ

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

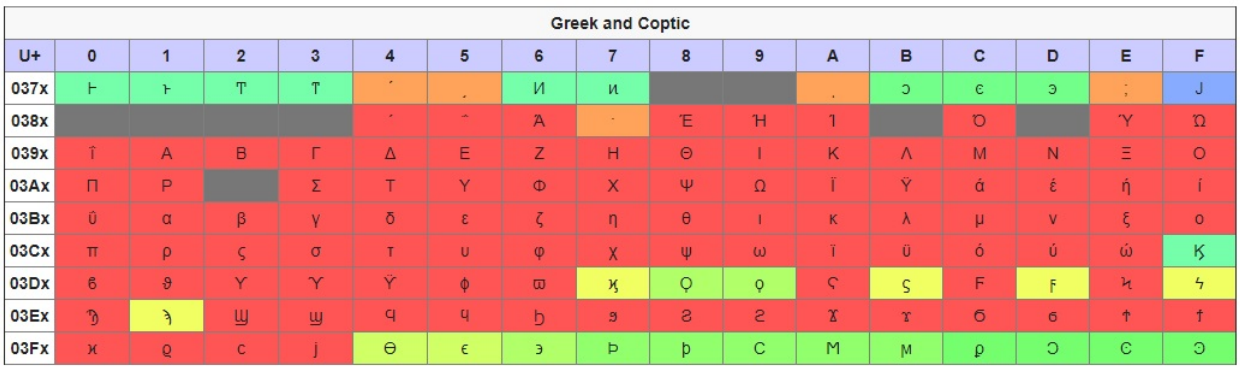
Figura 50 – Arábico



Arabic																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
060x	ـً	ـٌ	ـٍ	ـَ	ـُ	ـِ	ـّ	ـْ	٘	ٙ	ٚ	ٛ	ٜ	ٝ	ٞ	ٟ
061x	٠	١	٢	٣	٤	٥	٦	٧	٨	٩	٪	٫	٬	[ALM]	ٮ	ٯ
062x	ٰ	ٱ	ٲ	ٳ	ٴ	ٵ	ٶ	ٷ	ٸ	ٹ	ٺ	ٻ	ټ	ٽ	پ	ٿ
063x	ڀ	ځ	ڂ	ڃ	ڄ	څ	چ	ڇ	ڈ	ډ	ڊ	ڋ	ڌ	ڍ	ڎ	ڏ
064x	ڐ	ڑ	ڒ	ړ	ڔ	ڕ	ږ	ڗ	ژ	ڙ	ښ	ڛ	ڜ	ڝ	ڞ	ڟ
065x	ڠ	ڡ	ڢ	ڣ	ڤ	ڥ	ڦ	ڧ	ڨ	ک	ڪ	ګ	ڬ	ڭ	ڮ	گ
066x	ڰ	ڱ	ڲ	ڳ	ڴ	ڵ	ڶ	ڷ	ڸ	ڹ	ں	ڻ	ڼ	ڽ	ھ	ڿ
067x	ۀ	ہ	ۂ	ۃ	ۄ	ۅ	ۆ	ۇ	ۈ	ۉ	ۊ	ۋ	ی	ۍ	ێ	ۏ
068x	ې	ۑ	ے	ۓ	ە	ۥ	ۦ	ۧ	ۨ	۩	۪	۫	۬	ۭ	ۮ	ۯ
069x	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	ۺ	ۻ	ۼ	۽	۾	ۿ
06Ax	݀	݁	݂	݃	݄	݅	݆	݇	݈	݉	݊	݋	݌	ݍ	ݎ	ݏ
06Bx	ݐ	ݑ	ݒ	ݓ	ݔ	ݕ	ݖ	ݗ	ݘ	ݙ	ݚ	ݛ	ݜ	ݝ	ݞ	ݟ
06Cx	ݠ	ݡ	ݢ	ݣ	ݤ	ݥ	ݦ	ݧ	ݨ	ݩ	ݪ	ݫ	ݬ	ݭ	ݮ	ݯ
06Dx	ݰ	ݱ	ݲ	ݳ	ݴ	ݵ	ݶ	ݷ	ݸ	ݹ	ݺ	ݻ	ݼ	ݽ	ݾ	ݿ
06Ex	ހ	ށ	ނ	ރ	ބ	ޅ	ކ	އ	ވ	މ	ފ	ދ	ތ	ލ	ގ	ޏ
06Fx	ސ	ޑ	ޒ	ޓ	ޔ	ޕ	ޖ	ޗ	ޘ	ޙ	ޚ	ޛ	ޜ	ޝ	ޞ	ޟ

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 51 – Grego e copta



Greek and Coptic																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
037x	ͰͰ	ͱͱ	ͲͲ	ͳͳ	ʹ	͵	ͶͶ	ͷͷ	͸	͹	;	ͻ	ͼ	ͽ	Ϳ	Ϳ
038x	΄	΅	Ά	·	Έ	Ή	Ί	ΌΌ	Ύ	Ώ	ΐ	Α	Β	Γ	Δ	Ε
039x	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π	Ρ	΢	Σ	Τ	Υ
03Ax	Φ	Χ	Ψ	Ω	Ϊ	Ϋ	ά	έ	ή	ί	ΰ	α	β	γ	δ	ε
03Bx	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	ρ	ς	σ	τ	υ
03Cx	φ	χ	ψ	ω	ϊ	ϋ	ό	ύ	ώ	Ϗ	ϐ	ϑ	ϒ	ϓ	ϔ	ϕ
03Dx	ϖ	ϗ	Ϙ	ϙ	Ϛ	ϛ	Ϝ	ϝ	Ϟ	ϟ	Ϡ	ϡ	Ϣ	ϣ	Ϥ	ϥ
03Ex	Ϧ	ϧ	Ϩ	ϩ	Ϫ	ϫ	Ϭ	ϭ	Ϯ	ϯ	ϰ	ϱ	ϲ	ϳ	ϴ	ϵ
03Fx	϶	Ϸ	ϸ	Ϲ	Ϻ	ϻ	ϼ	Ͻ	Ͼ	Ͽ	Ѐ	Ё	Ђ	Ѓ	Є	Ѕ

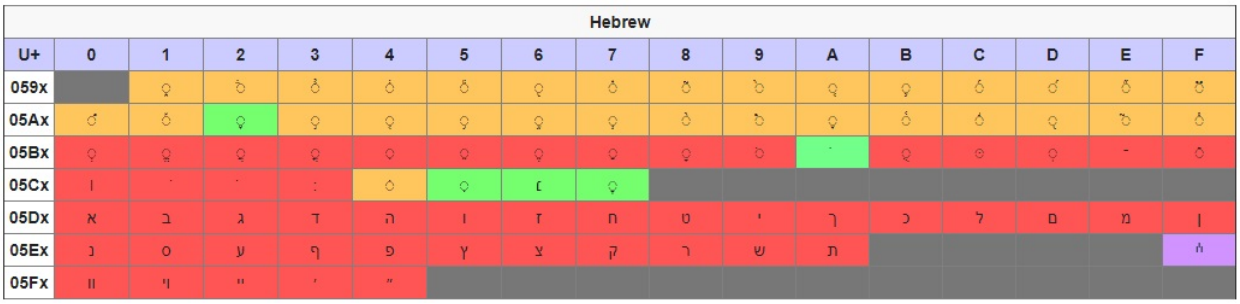
Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 52 – Armênio

Armenian																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
053x		Ա	Բ	Գ	Դ	Ե	Զ	Է	Ը	Թ	Ժ	Ի	Լ	Խ	Ծ	Կ
054x	Հ	Ձ	Ղ	Ճ	Մ	Յ	Ն	Շ	Ո	Չ	Պ	Ջ	Ռ	Ս	Վ	Տ
055x	Ր	Ց	Ւ	Փ	Ք	Օ	Ֆ			ՙ	՚	՛	՜	՝	՞	՟
056x	ՠ	ա	բ	գ	դ	ե	զ	է	ը	թ	ժ	ի	լ	խ	ծ	կ
057x	հ	ձ	ղ	ճ	մ	յ	ն	շ	ո	չ	պ	ջ	ռ	ս	վ	տ
058x	ր	ց	ւ	փ	ք	օ	ֆ	և	ֈ	։	֊			֍	֎	֏

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF

Figura 53 – Hebraico



Hebrew																
U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
059x		֑	֒	֓	֔	֕	֖	֗	֘	֙	֚	֛	֜	֝	֞	֟
05Ax	֠	֡	֢	֣	֤	֥	֦	֧	֨	֩	֪	֫	֬	֭	֮	֯
05Bx	ְ	ֱ	ֲ	ֳ	ִ	ֵ	ֶ	ַ	ָ	ֹ	ֺ	ֻ	ּ	ֽ	־	ֿ
05Cx	׀	ׁ	ׂ	׃	ׄ	ׅ	׆	ׇ								
05Dx	א	ב	ג	ד	ה	ו	ז	ח	ט	י	ך	כ	ל	ם	מ	ן
05Ex	נ	ס	ע	ף	פ	ץ	צ	ק	ר	ש	ת					׾
05Fx	ׯ	װ	ױ	ײ	׳	״										

Fonte: https://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF