# CONCEITOS DE COMPUTAÇÃO II

**Dados Internacionais de Catalogação na Publicação (CIP)**
**(Jeane Passos de Souza - CRB 8ª/6189)**

Barboza, Stelvio
Conceitos de computação II / Stelvio Barboza. – São Paulo : Editora Senac São Paulo, 2020. (Série Universitária)

Bibliografia.
e-ISBN 978-65-5536-223-7 (ePub/2020)
e-ISBN 978-65-5536-224-4 (PDF/2020)

1. Ciência da computação 2. Tecnologia da informação (TI) – Conceitos 3. Tecnologia da informação (TI) – Desenvolvimento 4. Tecnologia digital 5. Álgebra booleana I. Título. II. Série.

20-1174t CDD – 004
BISAC COM014000

**Índice para catálogo sistemático**
**1. Ciência da computação 004**

# CONCEITOS DE COMPUTAÇÃO II

Stelvio Barboza

## Administração Regional do Senac no Estado de São Paulo

**Presidente do Conselho Regional**
Abram Szajman

**Diretor do Departamento Regional**
Luiz Francisco de A. Salgado

**Superintendente Universitário e de Desenvolvimento**
Luiz Carlos Dourado

## Editora Senac São Paulo

**Conselho Editorial**
Luiz Francisco de A. Salgado
Luiz Carlos Dourado
Darcio Sayad Maia
Lucila Mara Sbrana Sciotti
Jeane Passos de Souza

**Gerente/Publisher**
Jeane Passos de Souza (jpassos@sp.senac.br)

**Coordenação Editorial/Prospecção**
Luís Américo Tousi Botelho (luis.tbotelho@sp.senac.br)
Dolores Crisci Manzano (dolores.cmanzano@sp.senac.br)

**Administrativo**
grupoedsadministrativo@sp.senac.br

**Comercial**
comercial@editorasenacsp.com.br

**Acompanhamento Pedagógico**
Mônica Rodrigues dos Santos

**Designer Educacional**
Patrícia Pinheiro de Sant'Ana

**Revisão Técnica**
Marco Antonio Barreto

**Preparação e Revisão de Texto**
Juliana Ramos Gonçalves

**Projeto Gráfico**
Alexandre Lemes da Silva
Emília Corrêa Abreu

**Capa**
Antonio Carlos De Angelis

**Editoração Eletrônica**
Sidney Foot Gomes

**Ilustrações**
Sidney Foot Gomes

**Imagens**
Adobe Stock

**E-pub**
Ricardo Diana





Editora Senac São Paulo
Rua 24 de Maio, 208 – 3º andar
Centro – CEP 01041-000 – São Paulo – SP
Caixa Postal 1120 – CEP 01032-970 – São Paulo – SP
Tel. (11) 2187-4450 – Fax (11) 2187-4486
E-mail: editora@sp.senac.br
Home page: http://www.livrariasenac.com.br

# Sumário

Capítulo 1

# Introdução: circuitos lógicos

No mundo moderno, somos cercados por dispositivos eletrônicos, que usam tecnologias das mais variadas e são o resultado da evolução das técnicas de projeto e fabricação ao longo dos últimos séculos. Somos herdeiros, por exemplo, de pesquisas no campo da eletricidade e da matemática que remontam ao século XVIII.

Atualmente, em um mesmo dispositivo eletrônico, é muito comum que tenham sido usadas diversas técnicas de projeto e diversas tecnologias de fabricação dos componentes eletrônicos dos produtos que fazem a vida moderna ser o que é.

Dentre as técnicas de projeto utilizadas para desenvolver determinado produto eletrônico, é comum identificar claramente trechos do

circuito que utilizam técnicas de projeto analógico ou técnicas de projeto digital, e ainda trechos nos quais as duas técnicas se misturam – em geral, as interfaces entre os trechos de técnica analógica e trechos de técnica digital.

Resumidamente, os trechos de um produto eletrônico identificados como analógicos são aqueles em que o controle da tensão e do funcionamento do circuito se dá de forma contínua. Em geral, eles se baseiam nas propriedades físicas dos componentes eletrônicos envolvidos, como um regulador de tensão linear, que utiliza a curva de resposta de um diodo Zener reversamente polarizado para regular a tensão de saída do regulador de tensão.

Outro exemplo de circuito analógico são os amplificadores utilizados para condicionar o sinal de um microfone, sendo que o sinal captado é proporcional à pressão sonora percebida pelo microfone.

Por sua vez, os trechos do circuito que foram projetados utilizando a técnica digital são, em geral, trechos em que o controle do circuito se dá por sinais discretos de tensão elétrica (diferença de potencial em volts), ou seja, são trechos cujo funcionamento se dá por mudanças abruptas de tensão elétrica e que possuem apenas dois níveis de tensão elétrica válidos para o funcionamento do circuito. Mais adiante, vamos perceber que os dois níveis costumam identificar nível lógico 0 ou nível lógico 1, e por que isso é importante.

A tecnologia de fabricação dos componentes eletrônicos, em geral, determina os níveis de tensão elétrica que são válidos para identificar os níveis lógicos 0 e 1. Por exemplo, temos o padrão TTL, definido inicialmente na década de 1960, que tinha como tensões para os níveis lógicos 0 e 1, respectivamente, 0 V e 5 V.

Esse padrão foi sucedido por outros, que trabalham com níveis 0 V e 3,3 V ou 0 V e 1,8 V. Há, ainda, padrões que não identificam o nível lógico 0 como 0 V, como o padrão LVPECL, que identifica o nível 0 como 1,6

V e o nível lógico 1 como 2,4 V (HOLLAND, 2002), ou mesmo o padrão RS232 de comunicação serial, em que o nível lógico 0 corresponde a uma tensão entre 5 V e 15 V, e o nível lógico 1 corresponde a uma tensão entre -5 V e -15 V.

Não cabe, neste momento, discutir se determinado padrão é melhor ou pior que outro, ou por que foram inventados tantos padrões de tensão para representar os valores lógicos 0 e 1. O mais importante é perceber que os circuitos digitais trabalham com dois níveis distintos de tensão e que esses níveis correspondem aos níveis lógicos de que vamos tratar a seguir.

# 1 Sistemas binários e a relação com a álgebra de Boole

As técnicas de projeto digitais atualmente em uso tiveram sua base matemática inicialmente descrita na obra *An investigation of the laws of thouth on which are founded the mathematical theories of logic and probabilities*, escrita pelo matemático inglês George Boole e publicada em 1854.

Paralelamente a George Boole, Augustus De Morgan – em sua obra *Formal logic*, publicada em 1847, e em artigos subsequentes, publicados pela *Transactions of the Cambridge Philosophical Society* – criou as leis de De Morgan, que, em conjunto com as teorias de Boole, formam a base da matemática binária.

**PARA SABER MAIS**

Para saber mais sobre a trajetória e a importância de George Boole, busque o documentário *The genius of George Boole* (direção de Stephen Mizelas, Reino Unido, 2015, 59 min.).

Entretanto, apesar de as bases da matemática binária terem sido descritas no século XIX, seu uso prático em engenharia teve início apenas no século XX, quando o engenheiro norte-americano Claude Elwood Shannon aplicou as ideias e teorias de Boole na resolução de problemas relacionados ao projeto de sistemas de telefonia da época: era o início das centrais telefônicas automáticas. Como fruto desse trabalho, Shannon publicou o artigo "Symbolic analisys of relay and switching circuits", em 1938.

Nesse artigo pioneiro, ele descreve o uso prático das ideias de Boole, o modo de aplicar suas equações e como isso podia ser implementado no nível físico dos relês utilizados naquela época para a implementação dos circuitos de comutação e controle das centrais telefônicas automáticas, fazendo uma comparação entre os cálculos de lógica propostos por Boole e o seu uso prático.

Ainda não se tratava de um projeto de computador de uso geral, mas da implementação de uma técnica de projeto que anos mais tarde viria a tornar viável a construção de computadores de uso geral, ou seja, máquinas que poderiam ser programadas posteriormente à sua fabricação e cuja função seria dada pelo programa a ser carregado nela.

Essa técnica de projeto desenvolvida por Shannon viria a ser a base de toda a técnica de projeto digital no nível lógico, independentemente de como seria implementado o nível físico (componentes eletrônicos).

Em termos de implementação, podemos entender o trabalho de Shannon da seguinte forma: quando há tensão para acionar a bobina de um relê, isso corresponde a um nível lógico 1, ou verdadeiro, e quando não temos tensão para acionar a bobina de um relê, temos um nível lógico 0, ou falso.

Dessa forma, podemos perceber que, tendo apenas esses dois estados, a implementação física das equações de Boole se torna viável mesmo com uma técnica de fabricação bastante simples (a utilização de relês).

Hoje, os circuitos lógicos são feitos utilizando transistores, que possuem vantagens evidentes em relação às implementações originais de Shannon, como o tamanho e o tempo de transição de um estado a outro, sem contar o consumo de energia.

## 2 Representando números

Alguns conceitos, depois que os utilizamos por muito tempo, parecem muito simples, e não paramos muito para pensar neles. Um deles diz respeito à representação de números, para que eles possam ser úteis em nosso dia a dia.

A forma mais "natural" de representação numérica, que as pessoas que não são da área conhecem, é a forma decimal, em que há dez símbolos básicos para a representação de quantidades, além de uma lei de formação segundo a qual podemos combinar os símbolos a fim de representar qualquer quantidade de determinado item que estejamos contando.

A lei de formação do sistema decimal pode ser descrita como um polinômio, em que cada elemento corresponde a um multiplicador e a uma mantissa. Antes de partir para um exemplo, vamos lembrar uma das propriedades da potencialização:

$10^0 = 1$

$10^1 = 10$

$10^2 = 100$

$10^3 = 1.000$

$10^4 = 10.000$

$10^5 = 100.000$

Para ilustrar isso, observe a decomposição do número 1976.

**Tabela 1 – Decomposição do número 1976**

| | UNIDADE | DEZENA | CENTENA | MILHAR |
|---|---|---|---|---|
| 1976 = | (6 × 1) + | (7 × 10) + | (9 × 100) + | (1 × 1.000) |

A decomposição apresentada pelo quadro é equivalente a:

$$1976 = (6 \times 10^0) + (7 \times 10^1) + (9 \times 10^2) + (1 \times 10^3)$$

Nessa expressão, cada número da representação decimal multiplica a base elevada ao número da posição.

**Tabela 2 – Decompondo números na base 10**

| | | | |
|---|---|---|---|
| 3 | 2 | 1 | 0 |
| $10^3$ | $10^2$ | $10^1$ | $10^0$ |
| 1.000 | 100 | 10 | 1 |
| 1 x 1.000 | 9 x 100 | 7 x 10 | 6 x 1 |
| 1 | 9 | 7 | 6 |

A partir da regra apresentada, podemos extrair o polinômio para a base 10:

$$(n)_{10} = n_i \times 10^i + n_{i-1} \times 10^{(i-1)} + n_{i-2} \times 10^{(i-2)} + \ldots + n_2 \times 10^2 + n_1 \times 10^1 + n_0 \times 10^0$$

Entretanto, a regra para a base 10 também se aplica a qualquer base de representação, basta termos símbolos suficientes que equivalham a todos os elementos da base que desejamos representar.

Assim, podemos criar o que é chamado de polinômio geral:

$$(n)_b = n_i \times b^i + n_{i-1} \times b^{(i-1)} + n_{i-2} \times b^{(i-2)} + \ldots + n_2 \times b^{(2)} + n_1 \times b^{(1)} + n_0 \times b^{(0)}$$

Nesse polinômio, b é a base em que queremos representar o valor n, e $n_i$ são as decomposições parciais do número n na base b.

Podemos usar o polinômio geral para representar números em qualquer base e, por consequência, podemos usá-lo para fazer conversões de base.

# 3 Conversão da base 10 para a base 2

Antes de tratarmos da conversão da base 10 para a base 2, precisamos definir o que significa um bit. Um bit é a menor unidade de informação existente. Em um circuito lógico, representa se determinado circuito está ligado ou desligado; porém, em um sistema para representar um número, um bit simboliza a menor quantidade que é possível representar, no caso, zero ou uma unidade. Ou seja, sempre que estivermos falando sobre sistemas de representação binário, haverá apenas a possibilidade, em cada bit da representação, de ter ou não a quantidade que aquela posição representa. Isso ficará mais claro ao longo deste capítulo, quando definirmos a base 16.

No sistema binário, a cada bit, só temos duas possibilidades: ou a quantidade da posição existe, ou ela é nula. Para simplificar a representação, particularmente quando estamos fazendo os cálculos, é necessário utilizar símbolos que representem os elementos da base com a qual estamos trabalhando.

Os dois símbolos que temos para representar quantidades na base 2 são o 0 e o 1, em que 0 representa a ausência daquela quantidade e 1, a presença daquela quantidade.

A regra de formação dos números na base 2 é:

$$(n)_2 = n_i \times 2^i + n_{i-1} \times 2^{(i-1)} + n_{i-2} \times 2^{(i-2)} + \ldots + n_2 \times 2^{(2)} + n_1 \times 2^{(1)} + n_0 \times 2^{(0)}$$

Como exemplo, temos:

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

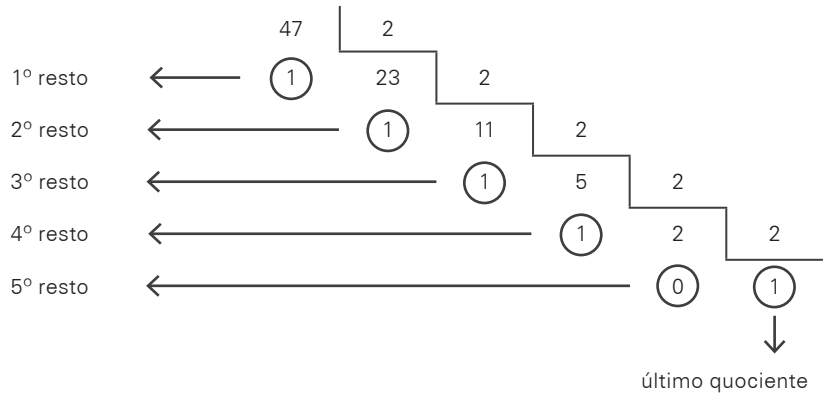
$$= 32 + 0 + 8 + 4 + 0 + 1$$

$$= (45)_{10}$$

E como fazemos para converter a base 10 para a base 2?

A forma mais simples é utilizando o resto por divisões sucessivas na referida base (IDOETA, 1999, p. 5). A técnica se resume a dividir o número em sua base original por 2, até obtermos o último quociente diferente de zero. Esse processo pode ser demonstrado de uma forma simplificada na figura 1, que representa a conversão do número $(47)_{10}$ para a base 2.

**Figura 1 – Técnica das divisões sucessivas**




Observe que, para compor o número na base para a qual queremos converter, estamos pegando o resto de cada divisão até o último quociente, sendo que o primeiro resto é o dígito menos significativo, e o último quociente, o dígito mais significativo.

No jargão da computação e da eletrônica digital, o bit menos significativo é chamado de LSB (least significant bit), e o bit mais significativo é o MSB (most significant bit). Para a representação do número binário, usamos o esquema apresentado na tabela 3.

**Tabela 3 – Representação do número binário obtido na técnica de divisões sucessivas**

| ÚLTIMO QUOCIENTE | 5° RESTO | 4° RESTO | 3° RESTO | 2° RESTO | 1° RESTO |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 |

A representação binária do número 47 decimal gera o número binário $(101111)_2$, que é equivalente a $(47)_{10}$.

Também podemos descrever essa operação por meio das equações a seguir.

$$47 \,/\, 2 = 23 \text{ com resto } 1$$

Podemos escrever isso da seguinte forma:

$$(23 \times 2) + 1 = 47 \text{ (Equação A)}$$

Também podemos representar 23 por uma divisão:

$$23 \,/\, 2 = 11 \text{ com resto } 1$$

Que, por sua vez, podemos representar por:

$$(11 \times 2) + 1 = 23 \text{ (Equação B)}$$

Substituindo 23 da equação A pela equação B, temos:

$$(11 \times 2 + 1) \times 2 + 1 = 47$$

Podemos reescrever isso de uma outra forma, mais adequada:

$$(11 \times 2) \times 2 + 1 \times 2 + 1 = 47$$

$$11 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47 \text{ (Equação C)}$$

Podemos repetir esse processo com o número 11:

$$11 \,/\, 2 = 5 \text{ com resto } 1$$

$$(2 \times 5) + 1 = 11 \text{ (Equação D)}$$

Substituindo 11 na equação C pela equação D, temos:

$$(11) \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$$

$$((5 \times 2) + 1) \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$$

$(5 \text{ x } 2) \times 2^2 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$5 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$ (Equação E)

Repetindo o processo, temos:

$5 / 2 = 2$ com resto 1

$(2 \times 2) + 1 = 5$

$(5) \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$((2 \times 2) + 1) \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$(2 \times 2) \times 2^3 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$(2^2) \times 2^3 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$ (Equação F)

Podemos representar a equação F de uma forma mais conveniente:

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

Por sua vez, podemos colocar a equação acima em uma tabela, de modo que a representação na base 2 fique mais clara.

**Tabela 4 – Representando um número binário na forma de uma tabela**

| $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | 101111 = 47 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | |

O resultado apresentado na tabela 4, obtido por meio de equações, é equivalente ao resultado obtido por meio da primeira técnica de conversão, usando o resto da divisão por 2.

# 4 Convertendo para a base 16

A segunda base mais importante para a representação numérica, em termos de computação, é a base 16, que também é conhecida como hexadecimal (IDOETA, 1999, p. 19).

A base hexadecimal é composta pelos dez algarismos tradicionais da base decimal somados com mais seis símbolos, a fim de completar o número de símbolos necessário para representar, em um único algarismo, os 16 símbolos usados para a formação da base.

O conjunto de símbolos que formam a base hexadecimal e seus valores equivalentes na representação decimal estão representados na tabela 5.

**Tabela 5 – Equivalência entre os números hexadecimais e decimais**

| HEXADECIMAL | DECIMAL |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | 10 |
| B | 11 |
| C | 12 |

(*Cont.*)

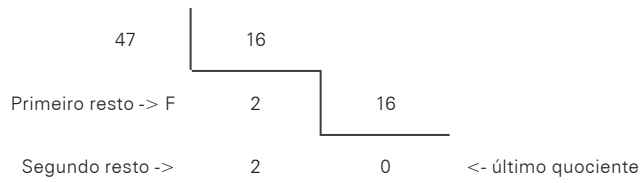
| HEXADECIMAL | DECIMAL |
|---|---|
| D | 13 |
| E | 14 |
| F | 15 |

Na tabela 5, observe que o valor 10 em decimal corresponde ao símbolo A em hexadecimal, que o valor 11 corresponde ao símbolo B, e assim sucessivamente, até o F, que corresponde à quantidade 15.

A regra de conversão da base 10 para a base 16 é a mesma da base 10 para a base 2, ou seja, fazem-se divisões sucessivas do número representado pelo número de símbolos da base de destino e usa-se o resto da divisão como os algarismos para representar o número.

Vamos tomar como exemplo o mesmo valor $(47)_{10}$, que utilizamos no exemplo de conversão da base 10 para a base 2.

**Figura 2 – Divisões sucessivas na base 16**




Na figura, podemos observar que o primeiro resto, que em decimal seria 15, na base hexadecimal é representado pela letra F.

Assim como no exemplo para a base 2, o elemento mais significativo é o último quociente, que no caso é 0, seguido pelo último resto, que no caso é o símbolo 2, e o menos significativo é o primeiro resto, que no caso corresponde ao símbolo F.

**Tabela 6 – Representação do resultado das divisões sucessivas na base 16**

| ÚLTIMO QUOCIENTE | 2° RESTO | 1° RESTO |
|---|---|---|
| 0 | 2 | F |

Utilizando equações para descrever a mesma operação apresentada, temos:

$$47 \,/\, 16 = 2 \text{ com resto F}$$

O que podemos escrever como:

$$(2 \times 16^1) + \mathrm{F} \times 16^0 = 47 \text{ (Equação A)}$$

Podemos representar 2 também por uma divisão:

$$2 \,/\, 16 = 0 \text{ com resto } 2$$

Que, por sua vez, podemos representar por:

$$(0 \times 16) + 2 = 2 \text{ (Equação B)}$$

Substituindo 2 da equação A pela equação B, temos:

$$(0 \times 16 + 2) \times 16^1 + \mathrm{F} \times 16^0 = 47$$

$$(0 \times 16 \times 16) + 2 \times 16^1 + \mathrm{F} \times 16^0 = 47$$

$$0 \times 16^2 + 2 \times 16^1 + \mathrm{F} \times 16^0 = 47$$

E podemos simplificar a equação acima por 02F.

# 5 Nibbles e bytes

Uma forma de entender a equivalência entre os números binários, hexadecimais e decimais é por meio da associação apresentada na tabela 7, a seguir.

**Tabela 7 – Representação dos números de 0 a 15 em hexadecimal, binário e decimal**

| HEXADECIMAIS | BINÁRIOS | DECIMAIS |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| A | 1010 | 10 |
| B | 1011 | 11 |
| C | 1100 | 12 |
| D | 1101 | 13 |
| E | 1110 | 14 |
| F | 1111 | 15 |

É possível notar que, no padrão hexadecimal de representação de números, cada símbolo na coluna “Hexadecimais” corresponde a 4 bits na coluna “Binários”. Isso tem um bom motivo: no início da computação, havia implementações em que um byte era composto de 6 bits para implementar os códigos BCD, pois com 6 bits — ou seja, 64 caracteres

— ficava viável representar o alfabeto alfanumérico completo e alguns caracteres especiais. Em impressoras e terminais teletipo (TTY), conectados por meio de interfaces seriais, existiam muitas implementações na comunicação de dados, em que 1 byte correspondia a 7 bits, e atualmente ainda se pode configurar uma interface RS232C para operar em 7 bits de dados.

A primeira codificação em que 1 byte foi implementado com 8 bits foi feita pela IBM, com a criação do código EBCDIC, em 1960. Como a IBM era a maior empresa do setor e as demais empresas queriam que seus equipamentos fossem compatíveis com as padronizações dela, padronizou-se que 1 byte seria formado por 8 bits. Posteriormente, em 1961, também surgiu o código ASCII, de 8 bits.

O fato de padronizar um byte com 8 bits fez com que os programadores e engenheiros de computadores da época buscassem uma forma mais fácil de representar os números e comandos em vez de usar sequências de 8 bits.

Como não temos 256 símbolos imprimíveis no teclado, buscou-se uma forma mais simples de representar um byte, mas que fosse uma tradução direta do código binário. Foi aí que surgiram as ideias para o padrão hexadecimal.

Se você observar, com dois conjuntos de 4 bits, temos um byte. Ou seja, com dois símbolos hexadecimais podemos formar um byte, sem qualquer sombra de dúvida de como isso pode ser representado no código binário.

Um byte de 8 bits pode representar de 0 a 255 em decimal ou de 00 a FF em hexadecimal. Para ilustrar como essa forma de agrupamento é importante e simplifica a representação numérica, vamos considerar um número binário com 24 bits:

0011 1111 0100 0101 1011 1010

Podemos representá-lo em hexadecimal:

3F 45BA

E podemos montar um pequeno quadro para ilustrar como a conversão é direta.

**Tabela 8 – Representação de um número em binário e hexadecimal**

| 0011 | 1111 | 0100 | 0101 | 1011 | 1010 |
|---|---|---|---|---|---|
| 3 | F | 4 | 5 | B | A |

Observe que todo o trabalho de conversão pode ser feito de 4 em 4 bits, que correspondem a apenas um símbolo em hexadecimal. Essa é a forma mais simples e direta de conversão entre as duas bases, com pouca ou nenhuma operação matemática, bastando consultar a tabela 7 (p. 20).

Obviamente, é possível aplicar o mesmo polinômio geral para a conversão da base 16 para a base 2, ou a potenciação para converter da base 2 para a base 16, porém, o método apresentado é muito mais direto.

# Considerações finais

Neste capítulo, vimos como podemos usar elementos muito simples para representar números dentro de uma máquina e como podemos converter o mesmo número em diversas bases, tornando mais fácil a representação dos números e símbolos de que precisamos ao trabalharmos em um projeto digital ou escrevermos um programa de computador.

Essas são as ferramentas básicas para qualquer sistema que tenha componentes digitais, seja um sistema pequeno e simples ou um grande e complexo.

# Referências


BOOLE, George. **An investigation of the laws of thouth on which are founded the mathematical theories of logic and probabilities**. Cambridge: Cambridge University Press, 2009. (Cambridge Library Collection – Mathematics.)

DE MORGAN, Augustus. **Formal logic**. Londres: Taylor & Walton, 1847. Disponível em: https://books.google.com.br/books?id=HscAAAAAMAAJ&hl=pt-BR&pg=PR5#v=onepage&q&f=false. Acesso em: 12 ago. 2020.

HOLLAND, Nick. **Interfacing between LVPECL, VML, CML, and LVDS levels**. Texas: Texas Instruments – Application Report, dez. 2002. Disponível em: https://www.ti.com/lit/an/slla120/slla120.pdf. Acesso em: 5 mar. 2020.

IDOETA, Ivan V.; CAPUANO, Francisco G. **Elementos de eletrônica digital**. São Paulo: Érica, 1999.

SHANNON, Claude E. Symbolic analisys of relay and switching circuits in electrical engineering. **Transactions American Institute of Electrical Engineers**, [*s. l.*], vol. 57, n. 12, p. 713-723, dez. 1938.