Capítulo 1

Organização de um Computador

It is possible to invent a single machine which can be used to compute any computable sequence.

Alan Turing

Iniciemos com um exame, necessariamente superficial, da organização de um computador disponível em lojas de varejo em 2021[1]. A Figura 1.1 mostra um diagrama com os componentes principais no nosso computador. Vejamos a função de cada componente, e então um exemplo da operação conjunta de todos eles.

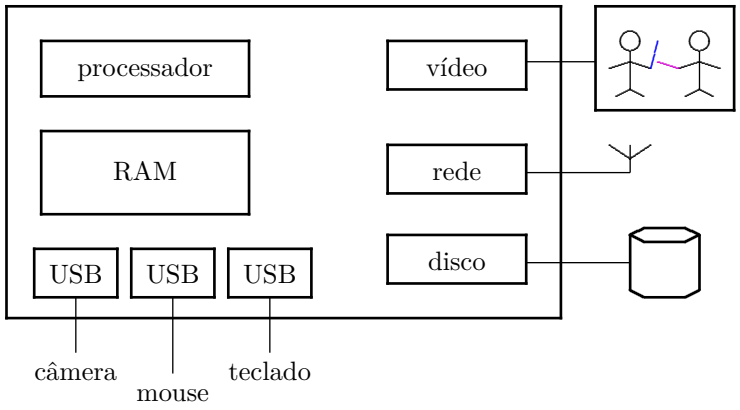

Figura 1.1: Organização de um computador pessoal.

Um pouco de notação Empregamos a notação de intervalos para denotar subconjuntos dos Inteiros, dos Naturais ou dos Reais. Um colchete fechado – '[' ou ']' – indica que o elemento adjacente ao colchete pertence ao conjunto, ou que ele faz parte do intervalo; por exemplo, o intervalo $[3, 7] \in \mathbb{N}$ inclui os Inteiros $3, 4, 5, 6, 7$. Um parêntese – '(' ou ')' – indica que o elemento adjacente *não* pertence ao intervalo; por exemplo, o intervalo $(0, 1] \in \mathbb{R}$ exclui o zero mas inclui todos os reais maiores que zero e menores ou iguais a 1.

Um *byte* é um octeto de dígitos binários e armazena um número natural no intervalo $[0, 255]$, ou um inteiro no intervalo $[-128, 127]$. Os *números mágicos* são $255 = (2^8 - 1)$, $-128 = -(2^7)$ e $+127 = (2^7 - 1)$. Essa tal *magia* é desmistificada no Capítulo 2.

[1] © Roberto André Hexsel, 2021. Versão de 1 de fevereiro de 2021.

Processador O *processador* é o componente principal de um computador pois é este quem interpreta e executa as *instruções* do programa que está a executar. As instruções representam comandos simples que o processador deve executar, tal como "adicione os conteúdos de x e y e armazene a soma em z", ou "armazene o valor de k na posição p da memória". Para consumo humano estas duas instruções são representadas como

```
z ← x + y
```

e

```
M(p) ← k.
```

No programa que o processador executa estas instruções são representadas por sequências de 32 dígitos binários (bits) tais como

```
11001010111100101110011010010001.
```

Evidentemente, a primeira representação é mais cômoda do que a segunda. A flecha que aponta para a esquerda representa a *atribuição* de um valor (`x+y`) a uma unidade de armazenamento (`z`), e o novo valor sobrescreve o conteúdo daquela unidade de armazenamento, que fora computado anteriormente.

Na disciplina Circuitos Digitais, estudamos a programação e a organização de um processador simples, que é descrito no Capítulo 10. Na disciplina de Projetos Digitais (Capítulos 10 e 11) veremos o projeto e a implementação de um processador realista. O projeto de computadores é objeto da disciplina Arquitetura de Computadores.

RAM Em Português, a memória RAM se chama *memória de escrita e de leitura*, embora o autor prefira a abreviatura do nome em Inglês: *Random Access Memory*. A razão para o *random* é histórica – nas décadas de 1940-50 a memória disponível era *sequencial* e para acessar a posição 19 da memória era necessário iniciar a busca pela posição 0, depois 1, 2, 3, ... 19. A possibilidade de acessos a quaisquer endereços, aleatoriamente, foi um grande progresso.

Do ponto de vista lógico, a RAM é uma função bijetora, os elementos do domínio são chamados de *endereços* e os elementos da imagem são o *conteúdo* armazenado nos respectivos endereços. Se a unidade de acesso é um byte (oito bits), então a memória é definida pela função

$$RAM : [0, C) \mapsto [0, 255]$$

sendo C a capacidade da memória, que é o número de bytes endereçáveis, e a cada endereço corresponde um valor representável em 8 bits, que é um número natural na faixa de 0 a 255.

Quando a memória é RAM, seus conteúdos podem ser alterados; quando a memória é ROM, ou *Read Only Memory*, seus conteúdos não podem ser alterados após o processo de fabricação ter sido concluído. Existem memórias que se comportam como ROM na maior parte do tempo, embora possibilitem que seu conteúdo seja alterado com o circuito em operação.

A organização interna de memórias ROM e RAM é introduzida na Seção 4.3.4. Voltaremos a esse assunto na disciplina de Projetos Digitais (Seções 5.5 e 5.6).

Interface de vídeo Talvez um nome mais apropriado do que *interface de vídeo* seja "tela de computador". Considere uma tela quadrada com 1024×1024 pontos, ou *pixeis*, com imagens em tons de cinza, de tal forma que cada pixel seja representado por um byte. Nesse caso, 0 representa um ponto branco, 255 representa um ponto preto, e 127 representa um ponto de tonalidade cinza-médio.

É necessária uma RAM com 1024×1024 bytes, ou 1 Mbytes, para armazenar uma cópia do que é exibido na tela. O controlador de vídeo varre esta memória e ilumina cada ponto da tela de acordo com o valor do byte que lhe corresponde. Para alterar a imagem, o processador altera o conteúdo da *memória de vídeo*.

Para uma tela colorida, são necessários 4 bytes por pixel, um para cada uma das cores fundamentais (vermelho, verde e azul) e mais um para a intensidade da iluminação do ponto. A representação mais rica envolve um aumento de um fator de quatro na capacidade de memória para manter a cópia da imagem em memória. Além de mais memória, o processamento de imagens coloridas é computacionalmente mais custoso do que para tons de cinza.

Interface de Rede Considere uma *interface de rede* que opere com mensagens com capacidade de 1 a 1024 bytes. No lado da recepção, as mensagens chegam a qualquer instante e por isso a interface deve conter memória para acomodar algumas mensagens, caso o processador não esteja disponível para tratá-las no instante em que são recebidas. No lado da transmissão, bastaria espaço para acomodar uma mensagem de tamanho máximo; por razões de eficiência, também aloca-se espaço para algumas mensagens no circuito de transmissão.

Como deve ser gerenciado o fluxo de mensagens para possibilitar a transferência de um arquivo grande? Como a ordem das mensagens é preservada? É desejável que ocorram várias transferências simultaneamente? Como os fluxos distintos são mantidos separados? Como os computadores ligados à Internet são endereçados? O que fazer quando da ocorrência de erros? Quais seriam os erros? Respostas precisas para estas perguntas serão obtidas ao cursar a(s) disciplina(s) de Redes de Computadores.

Disco Magnético como Memória Secundária A RAM é chamada de *memória primária* porque é essa memória que o processador acessa diretamente para executar programas e ler ou atualizar dados. Contudo, a capacidade da RAM é menor do que o conjunto de dados com que trabalhamos, e assim a *memória secundária* é empregada para armazenar arquivos tais como livros, filmes, etc. Em 2021, a memória secundária é tipicamente implementada com discos magnéticos. Como o nome indica, o 'disco' é coberto com óxido ferroso e porções diminutas da superfície do disco são magnetizadas para codificar 0s e 1s.

Cada superfície de um disco magnético é organizada em trilhas concêntricas, cada trilha dividida em setores. Um setor armazena de 1024 a 4096 bytes. Uma unidade de disco magnético pode ter mais de uma superfície. A posição de um determinado byte no disco é dada pela tripla $\langle \text{setor}, \text{trilha}, \text{superfície} \rangle$.

Por uma série de razões que serão discutidas mais adiante no curso, é conveniente trabalhar com um *disco lógico*, que é endereçado somente pelo número do bloco desejado. Em geral, um *bloco lógico* tem o mesmo tamanho de um setor, ou um *bloco físico*.

O controlador do disco faz a conversão de endereços físicos de 3 dimensões para endereços lógicos de uma dimensão

$$\langle \text{setor}, \text{trilha}, \text{superfície} \rangle \mapsto \langle \text{número do bloco} \rangle.$$

O endereço de um byte no disco lógico pode ser determinado pelas funções

$$\mathit{disco} : [0, N) \mapsto \mathit{bloco}, \quad \mathit{bloco} : [0, B) \mapsto [0, 255]$$

sendo N o número de blocos, e B o tamanho de um bloco.

Dispositivos de armazenamento com memória de estado sólido (*Solid State Disks* ou SSDs) estão se tornando populares pelo seu tempo de acesso, que é cerca de 100 vezes mais curto do que o dos discos magnéticos. É provável que em breve, os discos magnéticos sejam suplantados por SSDs. SSDs são acessados como discos magnéticos, e indexados pelo número do bloco. Em tese, estes dispositivos poderiam ser acessados como RAM, byte a byte ao invés de "bloco depois byte".

As memórias primária e secundária são estudadas nas disciplinas Software Básico e Sistemas Operacionais. A tradução de programas escritos em Pascal ou C para linguagem de máquina é estudada aqui, em Projetos Digitais, Software Básico e em Construção de Compiladores.

Exemplo 1.1 Vejamos como é o fluxo dos bytes quando obtemos uma cópia de um arquivo a partir de um servidor de conteúdos da Internet. A Figura 1.2 mostra uma versão ligeiramente expandida do computador da Figura 1.1.

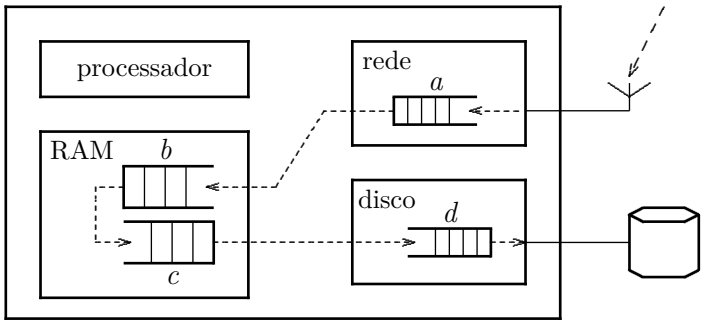

Figura 1.2: Fluxo de bytes entre interface de rede e memória secundária.

As mensagens são recebidas do servidor remoto em tempos imprevisíveis e indeterminados, e por isso emprega-se uma fila (a) para acomodar as diferenças na taxa em que as mensagens são recebidas, e a taxa com que elas são tratadas pelo processador. Esta fila tem a mesma função que a fila de um banco, aonde as pessoas (mensagens) esperam pelo atendimento por um dos caixas (processador). No caso da interface de rede, a fila é necessária para desacoplar as velocidades de chegada na antena e de atendimento pelo processador. A fila a é gerenciada pelo controlador da interface de rede, que é, ele próprio, um 'computador' dedicado a tratar da recepção e envio de mensagens através da interface de rede.

A fila b é usada para garantir que todas as mensagens que deveriam ter sido recebidas foram recebidas, e mais ainda, que a ordem dos bytes no arquivo original foi preservada durante a transmissão. Essa fila é gerenciada pelo sistema operacional (SO) do computador, e é acessada pelo processador para inserir novos elementos na cauda, e retirar os elementos mais antigos da cabeça da fila. Esta fila é parte do componente do SO que gerencia a transmissão e recepção de mensagens.

Uma vez que o SO verificou que o segmento do arquivo que está na fila b foi recebido corretamente, este segmento é removido da fila b e inserido na fila c. Esta fila é parte do componente do SO que gerencia o sistema de arquivos, e tem uma finalidade semelhante à da fila b, exceto que, ao invés de mensagens, os elementos desta fila são blocos lógicos que serão gravados no disco. Em geral, o tamanho dos blocos lógicos é distinto da capacidade máxima das mensagens, e é por isso que mensagens e blocos devem ser mantidos em filas separadas.

A fila d é gerenciada pelo controlador do disco, e seus elementos são blocos lógicos, esperando para ser gravados nos respectivos blocos físicos. O tempo de acesso de um disco magnético é longo e uma

escrita em disco demora o tempo equivalente à execução de 10^5 a 10^6 instruções pelo processador. Para todos os fins práticos, o tempo de acesso aos blocos físicos é imprevisível, e a fila d acomoda as diferenças de velocidade com que blocos lógicos são nela inseridos pelo processador, e blocos físicos são removidos e gravados na superfície magnetizada pelo controlador do disco. Assim como o controlador da interface de rede, o controlador de disco é um 'computador' dedicado a controlar as transferências entre memória RAM e a superfície magnetizada. ◁

Exercícios

Ex. 1.1 Quais as diferenças, do ponto de vista operacional, da aplicação da vacina tríplice em bebês e da vacina – quando esta ficar disponível – para a COVID-19?

Ex. 1.2 Considere um posto de saúde que atende a 100.000 pessoas, e que a vacina para COVID-19 está, finalmente, disponível. No posto existem quatro filas de aplicação de vacina, e cada aplicação demora 5 minutos. Supondo um horário de atendimento de oito horas, quantos dias são necessários para vacinar todas as 10^5 pessoas?

Ex. 1.3 Considerando os dados do Ex. 1.2, e que um lote de 5.000 doses é entregue ao posto a cada 3 dias, como isso altera o tempo de vacinação das 10^5 pessoas? Compensaria alterar o número de filas de aplicação? Se sim, como?

Ex. 1.4 Considerando os dados do Ex. 1.3, e que a fábrica de vacinas produz 1.000.000 de doses por semana, e supondo que todos os postos de saúde sejam como aquele do Ex. 1.2, quanto tempo levaria para vacinar todos os 220 milhões de brasileiros?

Ex. 1.5 Considerando os dados dos Ex. 1.2 a 1.4, compensaria alterar a logística de vacinação? Se sim, como?

Ex. 1.6 Podem ser necessárias duas doses para a imunização efetiva e eficaz. Como isso alteraria sua resposta ao Ex. 1.5?

> *Por mais estranho que possa parecer, estas questões são muito similares àquelas que devem ser respondidas por um projetista de computadores. As questões são abertas, e a resposta a uma questão posterior frequentemente impacta e altera as respostas para questões anteriores.*

Capítulo 2

Representação de Informação

Lord Polonius: What do you read, my lord?
Hamlet: Words, words, words.
Lord Polonius: What is the matter, my lord?
Hamlet: Between who?
Lord Polonius: I mean, the matter that you read, my lord.
William Shakespeare, Hamlet, II,2.

Qual é a informação contida na frase "*a Terra é redonda*"? Para alguém que cursa Ciência da Computação, o conteúdo desta frase é trivial porque *qualquer* pessoa que tenha cursado o *primeiro ano* do Ensino Fundamental *sabe* que a Terra é (aproximadamente) esférica. Por "conteúdo trivial" entenda-se que não há novidade na frase, logo aprendemos muito pouco ao ouvi-la, pois a frase não traz nenhuma informação nova para alguém que tenha concluído o primeiro ano do Fundamental.

Do ponto de vista da Computação, *informação* é algo que nos ajuda a compreender melhor o mundo em que vivemos, num determinado instante. Por exemplo, a frase "*a Lua é um icosaedro*", caso o satélite tivesse sofrido uma transformação extraordinária nos últimos dez minutos, conteria uma quantidade enorme de informação. Se, contudo, o satélite permanece como sempre o vemos no céu, a frase tem conteúdo informacional nulo.

A cena da peça citada acima é uma das minhas favoritas porque Hamlet exprime um fato – ele está a ler palavras no livro – mas Polônio espera por outro fato – Hamlet está a ler 'algo' no livro. Os dois estão corretos, exceto pelos distintos significados atribuídos à mesma informação, Hamlet num contexto micro – palavras soltas – e Polônio num contexto macro – texto. Para além disso, Hamlet está a fazer troça de Polônio ao interpretar a frase "*what is the matter?*" com um sentido diferente daquele pretendido por Polônio.

Se a frase "*a Terra é redonda*" for representada em caracteres Kanji, em cirílico, ou em sânscrito, teremos mais ou menos dificuldade em apreender seu significado. Para a imensa maioria dos habitantes de Angola, Brasil, Moçambique e Portugal, o significado da frase é um só. Para a imensa maioria dos habitantes da China, a frase não faz sentido algum, porque a interpretação dos símbolos com que a frase é desenhada não faz parte do dia a dia da maioria dos chineses. Além do consenso sobre o significado das palavras, é necessário um consenso sobre a forma de *representar* as palavras.

Para que a informação seja transmitida entre fonte e destino, é necessário um *alfabeto* – um *conjunto de símbolos* – mais uma gramática – um *conjunto de regras* – que define como os símbolos podem ser combinados. Além disso, um *conjunto de significados* é necessário para que informação seja transferida de uma entidade para outra. Por exemplo, a frase "*a Lua é feita de queijo*" somente é aceitável como exprimindo um fato verdadeiro no contexto das extraordinárias aventuras de Wallace & Gromit.

Vejamos um alfabeto e uma gramática que nos permitam exprimir informação sobre quantidades, ou que nos permitam *representar* quantidades.

2.1 Uma Representação para Números Inteiros

O alfabeto

$$A_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{␣}\} \tag{2.1}$$

nos permite exprimir, de forma simbólica, quantidades a que chamamos de *números inteiros*. A gramática é relativamente simples: uma quantidade maior que o número de dedos das mãos de um humano pode ser representada pela justaposição dos símbolos do alfabeto. Duas quantidades distintas são separadas por um espaço em branco, que também é um símbolo do nosso alfabeto. O espaço é indicado como '␣'.

A regra de formação dos números emprega a *representação posicional*: cada posição da quantidade representada corresponde a uma potência de dez. Um elemento 'solteiro' do alfabeto é multiplicado por 10^0. Com dois elementos justapostos, o da esquerda é multiplicado por 10^1 e o da direita por 10^0:

$$27 \;=\; 2 \times 10^1 + 7 \times 10^0 \;=\; 2 \times 10 + 7 \times 1.$$

Nesta *representação posicional* cada posição corresponde a uma potência de 10:

posição →	10^6	10^5	10^4	10^3	10^2	10^1	10^0
4.259.301	4	2	5	9	3	0	1

Para facilitar a leitura, acrescentamos o símbolo '.' ao nosso alfabeto, para separar os milhares.

Para uma representação com k dígitos, o valor do número natural N representado pela justaposição de k dígitos é obtido pela Equação 2.2, e é a soma dos dígitos $d_i,\ i \in [0, k)$ multiplicados pela base b elevada à i-ésima potência. Na Equação 2.2 temos $b = 10$ e $d \in A_{10}$.

$$N \;=\; \sum_{i=0}^{k-1} d_i \times b^i \;=\; d_0 \times b^0 + d_1 \times b^1 + d_2 \times b^2 + \ldots + d_{k-1} \times b^{k-1} \tag{2.2}$$

Se quisermos representar números reais, nosso alfabeto deve ser ampliado com uma ',' para separar as potências positivas das potências negativas, e um '−' para indicar números negativos. Numa representação com m dígitos para as potências positivas e n dígitos para as negativas, o número R é obtido da Equação 2.3, com $i \in [0, m)$ e $j \in [-n, 0)$. Números negativos são prefixados por '−'.

$$R = (\sum_{i=0}^{m-1} d_i \times 10^i)\,,\ (\sum_{j=-n}^{-1} d_j \times 10^j) \tag{2.3}$$

Com nosso alfabeto estendido, a palavra "$2.408,753$" significa

$$2 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 8 \times 10^0 + 7 \times 10^{-1} + 5 \times 10^{-2} + 3 \times 10^{-3}.$$

2.2 Quatro Dedos em Cada Mão

Se tivéssemos somente quatro dedos em cada mão, nosso alfabeto estendido para representar números naturais seria

$$A_8 = \{0, 1, 2, 3, 4, 5, 6, 7, \textvisiblespace, .\} \tag{2.4}$$

e a sequência de contagem com oito dedos seria

$$0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, \ldots$$

O sistema de numeração com base oito, chamado de *octal*, era usado em computação até meados da década de 1980, e felizmente caiu em desuso. Mãos com 8 dedos tornaram-se mais comuns na década de 1990 porque $8 + 8 = 16$ dedos propiciam representações mais compactas do que com $4 + 4 = 8$ dedos.

Para uma representação com k dígitos, o valor do número natural O representado pela justaposição de k dígitos é obtido pela Equação 2.5, e é a soma dos dígitos $d_i,\ i \in [0, k)$ multiplicados pelas potências de oito.

$$O = \sum_{i=0}^{k-1} d_i \times 8^i, \quad d \in A_8 \tag{2.5}$$

Quando usamos representações com mais de uma base, é usual apensar um subscrito ao número, indicando a sua base. A quantidade 254_{10} é representada na base 10, enquanto que 744_8 é representada na base 8. Geralmente, números na base 10 são representados sem o subscrito.

A conversão de uma quantidade representada em octal para a sua equivalente na base 10 é obtida pela aplicação direta da Equação 2.5: cada dígito é multiplicado pela potência de 8 que lhe corresponde e o valor representado é a soma das parcelas.

A conversão de uma quantidade representada na base 10 para sua equivalente em octal é um pouco mais complexa porque as bases são distintas, e dígitos maiores do que 7 são representados por dois dígitos na base 8, mas a correspondência não é direta porque $9_{10} = 7_8 + 2_8$.

O *módulo* de um número, com relação a uma base, é o resto da divisão inteira do número pela base. O operador módulo é representado pelo símbolo %. Com decimais, $5 \,\%\, 10 = 5$, $10 \,\%\, 10 = 0$, $12 \,\%\, 10 = 2$, e $12 \,\%\, 8 = 4$.

A conversão de uma quantidade D representada na base 10 para a sua equivalente na base 8 é obtida pela aplicação do *algoritmo de conversão de bases*:

Algoritmo de conversão de bases

(1) divida D por 8;

(2) guarde o quociente da divisão inteira para a próxima iteração;

(3) mantenha o módulo da divisão, que é o "dígito da vez";

(4) repita os passos (1) a (3) até que o quociente seja zero.

Esta sequência de passos produz os dígitos do valor convertido do menos significativo para o mais significativo.

Exemplo 2.1 Façamos a conversão de 1987_{10} para sua representação em octal. A conversão é mostrada na Tabela 2.1, e o valor convertido é 3703_8. O dígito menos significativo é obtido do módulo

de 1987 por 8, o segundo dígito é o módulo de 248 por 8, o terceiro é o módulo de 31 por 8, e o quarto dígito – o quociente da divisão da parte inteira é zero – é o módulo de 3 por 8. Para calcular o módulo, multiplicamos a parte fracionária do quociente por 8.

Na medida em que 'descemos' na tabela, o número original é dividido por uma potência cada vez maior da base. Na quarta linha, D já foi dividido por 8 três vezes: $8 \times 8 \times 8 = 8^3$. ◁

Tabela 2.1: Conversão de 1987 para octal.

quociente	módulo	dígito	potência
$1987 \div 8 = 248,375$	$1987 \,\%\, 8 = 0,375 \times 8 = 3$	$d_0 = 3$	8^0
$248 \div 8 = 31,000$	$248 \,\%\, 8 = 0,000 \times 8 = 0$	$d_1 = 0$	8^1
$31 \div 8 = 3,875$	$31 \,\%\, 8 = 0,875 \times 8 = 7$	$d_2 = 7$	8^2
$3 \div 8 = 0,375$	$3 \,\%\, 8 = 0,375 \times 8 = 3$	$d_3 = 3$	8^3

2.3 Oito Dedos em Cada Mão

Se tivéssemos oito dedos em cada mão, nosso alfabeto estendido seria

$$A_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, \lrcorner, .\} \tag{2.6}$$

e a sequência de contagem seria

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, \ldots$$

O sistema de numeração com base dezesseis, chamado de *hexadecimal*, substituiu o octal porque as representações em hexadecimal são mais compactas. O alfabeto para expressar quantidades na base 10 foi estendido com mais seis símbolos, que por conveniência são as seis primeiras letras do alfabeto usado para representar texto.

Para uma representação com k dígitos, o valor do número natural H representado pela justaposição de k dígitos é obtido pela Equação 2.7, e é a soma dos dígitos $d_i,\ i \in [0, k)$ multiplicados pelas correspondentes potências de dezesseis.

$$H = \sum_{i=0}^{k-1} d_i \times 16^i, \quad d \in A_{16} \tag{2.7}$$

A conversão de um número na base 10 para um na base 16 emprega o mesmo algoritmo que na conversão de decimal para octal, trocando-se a base.

Exemplo 2.2 Façamos a conversão de 1987_{10} para sua representação em hexadecimal. A conversão é mostrada na Tabela 2.2. Na medida em que descemos na tabela, o número original é dividido por uma potência cada vez maior da base 16.

A representação de 1987 necessita de quatro dígitos tanto em decimal (1987_{10}) quanto em octal (3703_8), mas somente três dígitos em hexadecimal ($7C3_{16}$). ◁

Exemplo 2.3 Façamos a conversão de $DC49_{16}$ (56.393_{10}) para sua representação em octal. A conversão é mostrada na Tabela 2.3. A representação em octal para $DC49_{16}$ é 156111_8, e esta emprega dois dígitos adicionais para representar a mesma quantidade. ◁

Tabela 2.2: Conversão de 1987 para hexadecimal.

quociente	módulo	dígito	potência
$1987 \div 16 = 124,1875$	$1987 \,\%\, 16 = 0,1875 \times 16 = 3$	$d_0 = 3$	16^0
$124 \div 16 = 7,7500$	$124 \,\%\, 16 = 0,7500 \times 16 = C$	$d_1 = C$	16^1
$7 \div 16 = 0,4375$	$7 \,\%\, 16 = 0,4375 \times 16 = 7$	$d_2 = 7$	16^2

Tabela 2.3: Conversão de DC49 para octal.

quociente	módulo	dígito	potência
$DC49 \div 8 = 1B89$	$DC4\mathbf{9} \,\%\, 8 = 9 \,\%\, 8 = 1$	$d_0 = 1$	8^0
$1B89 \div 8 = 371$	$1B8\mathbf{9} \,\%\, 8 = 9 \,\%\, 8 = 1$	$d_1 = 1$	8^1
$371 \div 8 = 6E$	$37\mathbf{1} \,\%\, 8 = 1 \,\%\, 8 = 1$	$d_2 = 1$	8^2
$6E \div 8 = D$	$6\mathbf{E} \,\%\, 8 = 14_{10} \,\%\, 8 = 6$	$d_3 = 6$	8^3
$D \div 8 = 1$	$\mathbf{D} \,\%\, 8 = 13_{10} \,\%\, 8 = 5$	$d_4 = 5$	8^4
$1 \div 8 = 0$	$\mathbf{1} \,\%\, 8 = 1 \,\%\, 8 = 1$	$d_5 = 1$	8^5

Na conversão do Exemplo 2.3 usamos uma propriedade do módulo. Como $16 = 2 \times 8$, ao computar o módulo de 8, pode-se usar somente o último dígito da representação hexadecimal para computar o resto da divisão inteira. Os dígitos para além do primeiro são múltiplos de 16, e portanto são também múltiplos de 8. O dígito relevante é mostrado em negrito na Tabela 2.3.

2.4 Um Dedo em Duas Mãos

Até agora vimos três representações para números naturais que são usadas porque facilitam o trabalho dos humanos mas que não são usadas internamente em sistemas digitais e nem nos circuitos de computadores. Os circuitos que empregamos para construir sistemas digitais representam números na base 2, ou na *representação binária*.

Na representação binária o alfabeto estendido é

$$A_2 = \{0, 1, \text{␣}, .\} \tag{2.8}$$

e a sequência de contagem é

$$0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, \ldots$$

Números representados na base 2 não são nada compactos e é por isso que são usadas as representações octal ou hexadecimal. A Tabela 2.4 mostra as representações em binário, hexadecimal e octal dos primeiros 16 naturais.

A expressão *dígito binário* é abreviada para *bit*, do Inglês ***b**inary dig**it***. Cada dígito hexadecimal é representado em 4 bits, e cada dígito octal em 3 bits. Na Tabela 2.4, por causa da reduzida faixa de valores, o dígito mais significativo da representação octal corresponde a um único bit.

Para uma representação com k dígitos, o valor do número natural B representado pela justaposição de k bits é obtido pela Equação 2.9, e é a soma dos dígitos $d_i,\ i \in [0, k)$ multiplicados pelas correspondentes potências de dois.

Tabela 2.4: Representação em decimal, binário, hexadecimal e octal.

decimal	binário	hexa.	octal
0	0000	0	00
1	0001	1	01
2	0010	2	02
3	0011	3	03
4	0100	4	04
5	0101	5	05
6	0110	6	06
7	0111	7	07
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

$$B = \sum_{i=0}^{k-1} d_i \times 2^i, \quad d \in A_2 \tag{2.9}$$

A conversão de um número na base 10 para um na base 2 emprega o algoritmo de conversão das seções anteriores. A sequência de divisões é algo mais longa porque a base é 2.

Exemplo 2.4 Façamos a conversão de 123_{10} para sua representação binária. A conversão é mostrada na Tabela 2.5, e a representação na base 2 é 111.1011_2. Sempre que o número por dividir for ímpar, o bit correspondente é 1. ◁

Tabela 2.5: Conversão de 123 para binário.

quociente	módulo	dígito	potência
$123 \div 2 = 61,5$	$123 \,\%\, 2 = 0,5 \times 2 = 1$	$d_0 = 1$	2^0
$61 \div 2 = 30,5$	$61 \,\%\, 2 = 0,5 \times 2 = 1$	$d_1 = 1$	2^1
$30 \div 2 = 15,0$	$30 \,\%\, 2 = 0,0 \times 2 = 0$	$d_2 = 0$	2^2
$15 \div 2 = 7,5$	$15 \,\%\, 2 = 0,5 \times 2 = 1$	$d_3 = 1$	2^3
$7 \div 2 = 3,5$	$7 \,\%\, 2 = 0,5 \times 2 = 1$	$d_4 = 1$	2^4
$3 \div 2 = 1,5$	$3 \,\%\, 2 = 0,5 \times 2 = 1$	$d_5 = 1$	2^5
$1 \div 2 = 0,5$	$1 \,\%\, 2 = 0,5 \times 2 = 1$	$d_6 = 1$	2^6

Para facilitar a leitura de números com muitos dígitos, os bits são agrupados em quartetos, de forma similar ao agrupamento de milhares.

As conversões entre binário e hexadecimal são triviais. Ao converter para hexadecimal, basta agrupar os bits de quatro em quatro e então converter cada quarteto segundo a Tabela 2.4. Ao converter para binário, cada dígito hexadecimal deve ser expandido para o quarteto que lhe corresponde.

Exemplo 2.5 Façamos a conversão de F1C7_{16} para sua representação binária. Ao consultar a Tabela 2.4 vemos que

$\text{F} \mapsto 1111,\ 1 \mapsto 0001,\ \text{C} \mapsto 1100,\ 7 \mapsto 0111,$

e $\text{F1C7}_{16} = 1111.0001.1100.0111_2$. ◁

Exemplo 2.6 Façamos a conversão de $1011101001101001010110 1_2$ para hexadecimal. É necessário agrupar os quartetos, iniciando pela direita

$101.1101.0011.0100.1010.1101_2$

e então consultar a Tabela 2.4:

$0101 \mapsto 5,\ 1101 \mapsto \text{D},\ 0011 \mapsto 3,\ 0100 \mapsto 4,\ 1010 \mapsto \text{A},\ 1101 \mapsto \text{D}.$

Assim, $1011101001101001010110 1_2 = \text{5D.34AD}_{16}$. ◁

2.5 Uma Representação para Números Reais na Base 2

A Equação 2.3 define a representação posicional de números reais na base 10. A representação na base 2 é definida pela Equação 2.10, com m dígitos na parte inteira, e n na parte fracionária.

$$R = (\sum_{i=0}^{m-1} d_i \times 2^i)\,,\,(\sum_{j=n}^{-1} d_j \times 2^j) \tag{2.10}$$

As quatro primeiras potências negativas de 2 são

$$2^{-1} = 0,5 \qquad 2^{-2} = 0,25 \qquad 2^{-3} = 0,125 \qquad 2^{-4} = 0,0625.$$

O número $5,3125_{10}$ é representado na base dois por $101,0101_2$

$$5,3125_{10} = 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-2} + 1 \times 2^{-4} = 4 + 1 + 0,25 + 0,0625.$$

O algoritmo de conversão de base para a parte inteira é aquele da página 18. O algoritmo para converter a parte fracionária D da base 10 para a base 2 é definido abaixo.

Algoritmo de conversão de base de frações

(1) multiplique D por 2;

(2) guarde a parte fracionária para a próxima iteração;

(3) se $D \times 2 \geqslant 1$, então o "dígito da vez" é 1, do contrário é 0;

(4) repita os passos (1) a (3) até que a parte fracionária seja zero.

Esta sequência produz os dígitos do valor convertido do mais significativo para o menos significativo. As partes inteira e fracionária devem ser convertidas separadamente.

Exemplo 2.7 Vejamos a conversão de $0,123_{10}$ para binário empregando somente 8 bits na representação da fração. A Tabela 2.6 mostra os passos da conversão, para oito dígitos à direita da vírgula.

Da mesma forma que um computador emprega um número finito de bits para representar a parte inteira, o número de bits empregado para representar a fração também é finito. Neste exemplo, a conversão foi truncada no oitavo bit. Quanto maior a largura do campo usado para representar a parte fracionária dos números, maior é a precisão da representação.

Tabela 2.6: Conversão de 0,123 para binário.

produto	$D \times 2 \geqslant 1$	dígito	potência
$0,123 \times 2 = \mathbf{0},246$	0	$d_{-1} = 0$	2^{-1}
$0,246 \times 2 = \mathbf{0},492$	0	$d_{-2} = 0$	2^{-2}
$0,492 \times 2 = \mathbf{0},984$	0	$d_{-3} = 0$	2^{-3}
$0,984 \times 2 = \mathbf{1},968$	1	$d_{-4} = 1$	2^{-4}
$0,968 \times 2 = \mathbf{1},936$	1	$d_{-5} = 1$	2^{-5}
$0,936 \times 2 = \mathbf{1},872$	1	$d_{-6} = 1$	2^{-6}
$0,872 \times 2 = \mathbf{1},744$	1	$d_{-7} = 1$	2^{-7}
$0,744 \times 2 = \mathbf{1},488$	1	$d_{-8} = 1$	2^{-8}

O número $0{,}123_{10}$ representado em 8 bits é $0{,}0001.1111_2$. Este número convertido novamente para um número real é 0,12109375, com um erro de representação que é 0,123−0,12109375=0,00190625, ou de aproximadamente dois milésimos. Este erro é chamado de *erro absoluto* porque é a diferença entre o valor real e o valor representado. O *erro relativo* é a razão entre o erro de representação e o valor representado, e é uma medida da proporção entre o erro e o valor real. Neste exemplo, o erro relativo é 0,00190625/0,123=0,0155, ou de 1,55%. ◁

Considere uma representação com 4 bits na fração. Dos 16 valores possíveis ($\langle 0000 \rangle$ a $\langle 1111 \rangle$), o diagrama na Figura 2.1 evidencia cinco deles.

$$\{0,\ \frac{1}{2},\ (\frac{1}{2}+\frac{1}{4}),\ (\frac{1}{2}+\frac{1}{4}+\frac{1}{8}),\ (\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16})\}$$

As representações indicadas são exatas, e portanto o erro de representação para esses cinco valores é zero. O erro máximo de representação no intervalo $[0, 1)$ é $\approx \frac{1}{16}$ e ocorre entre cada par de valores representáveis. O erro entre o maior valor representável (0,9375) e o fim do intervalo é indicado pela faixa hachurada no diagrama.

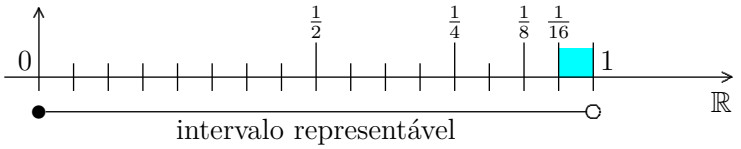

Figura 2.1: Valores representáveis com quatro bits na fração.

Os limites inferiores dos intervalos são representáveis exatamente $(0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \ldots, \frac{15}{16})$ mas os limites superiores não o são porque os intervalos são abertos no limite superior: $[\frac{k}{16}, \frac{k+1}{16}), k \in [0, 15]$. Assim, o erro é uma aproximação assintótica a $\frac{1}{16}$. O erro máximo com uma representação com n bits é $\approx \frac{1}{2^n}$. O limite superior não faz parte do intervalo porque ele é o primeiro elemento do intervalo seguinte, aquele com a parte inteira que é 1 maior do que a 'deste' intervalo. O início do 'próximo' intervalo é 1 na Figura 2.1.

A representação com 4 bits é obviamente incompleta e insuficiente porque somente 16 valores podem ser representados exatamente; todos os demais devem ser aproximados para o valor representável mais próximo. Por exemplo, pode-se aproximar para baixo, quando se emprega o valor menor mais próximo de um valor representável, ou pode-se aproximar para cima e escolhe-se o valor próximo maior do que aquele por representar.

Exemplo 2.8 A constante e é chamada de *número de Neper*, ou de *número de Euler*, e é a base dos logaritmos naturais. Esta constante pode ser representada pela somatória dos inversos dos fatoriais:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \cdots$$

A cada termo adicional, a soma fica mais próxima do valor de e. A representação de e com 15 dígitos após a vírgula é

$$2,718.281.828.459.045\,.$$

Se representarmos o número transcendental somente com os quatro primeiros termos da soma, obtemos

$$e_4 = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} = 2,66666\cdots$$

e esta representação produz um erro de $0,05161\cdots$ com relação à representação com 15 dígitos. Se melhorarmos a representação, empregando um termo a mais, obtemos

$$e_5 = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} = 2,708333\cdots$$

com um erro de representação de $0,009948\cdots$.

O termo *erro de representação* é usado para explicitar a influência que a forma de representar o número, com mais ou menos bits na parte fracionária, tem na precisão da representação. Quanto mais detalhado for o desenho de uma flor, melhor é a representação da flor real.

Qual é o limite para a quantidade de bits numa representação? Evidentemente, quanto maior o número de bits, melhor a qualidade dos valores representados. Um número infinito de bits permite representações perfeitas; contudo, a preços de 2021, pouquíssimas aplicações teriam custo razoável com tal representação. O Exemplo 2.9 ilustra as escolhas preço *versus* qualidade de representação. ◁

Exemplo 2.9 Você foi contratada para escrever o programa do controle remoto de um aparelho de ar condicionado residencial. O processador opera com inteiros de 8 bits e você deve projetar uma representação que permita exibir décimos de grau. A faixa de temperaturas de aplicação do aparelho é de -10C a $+50$C, e o termômetro fornece medidas com precisão de um centésimo de centígrado.

A faixa de valores de temperatura é de 60 graus – ou 60 valores inteiros – e para representá-la são necessários 6 bits (2^6=64), logo sobram dois bits para representar a fração. Estes 2 bits não são suficientes para representar décimos de grau, conforme a especificação – com dois bits podem ser representados somente as frações 0, 0,25, 0,50 e 0,75.

Como outra possibilidade de representação pode-se usar um byte para representar a parte inteira e outro byte para a parte fracionária. Neste caso, sobram valores para as temperaturas inteiras ($256 \gg 60$) e as frações podem ser representadas com precisão de 1/256, o que é um tanto melhor do que 1/10. Esta representação é mais custosa mas atende ao especificado. ◁

Representação *versus* realidade Essa é uma questão fundamental e que pode provocar nós na cabeça de vários dentre os leitores: uma coisa é a quantidade representada, outra coisa diferente é a representação da quantidade. $\frac{1}{3}$ e $0,33333\cdots$ são dois *desenhos* distintos que representam a mesma quantidade. O desenho pode ser em numerais romanos, indo-arábicos, Klingon, binário, hexadecimal ou decimal; essas todas podem ser usadas para representar uma mesma quantidade. A base com que se representa um número é só um *desenho*, uma representação, daquele número e não a *quantidade* ela própria.

Exercícios

Ex. 2.1 Escreva, em binário, todos os naturais entre zero e 31, representando-os todos com cinco bits.

Ex. 2.2 Supondo que os Klingons têm cinco dedos em cada mão, e considerando que a versão Klingon do alfabeto da Equação 2.1, estendido com o separador de milhares e o separador de frações, seja $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \wr, *, \circ\}$, codifique as quantidades 123, 5.483,99 e π no alfabeto Klingon.

Ex. 2.3 Repita o Ex. 2.2 usando a representação empregada na Roma antiga.

Ex. 2.4 Some, e depois multiplique, 345 e 78, representados em numerais romanos.

Ex. 2.5 Converta 100, 1.000, 10.000 e 100.000 para octal e para hexadecimal.

Ex. 2.6 Converta 100_b, 1.000_b, 10.000_b e 100.000_b para decimal, com b valendo 8 e depois 16.

Ex. 2.7 Converta 100, 1.000, 10.000 e 100.000 para binário.

Ex. 2.8 Converta os números 133 e 1033 para binário e para hexadecimal.

Ex. 2.9 Encontre as potências de 2 (2^n) mais próximas de $10^m, m \in \{3, 6, 9\}$.

Ex. 2.10 Estime o erro percentual entre as potências de 10 e as respectivas potências de 2 do Ex. 2.9.

Ex. 2.11 Converta 1/3, 1/5, 1/7, 1/11 e π para binário, empregando 4 bits na parte fracionária. Qual a magnitude dos erros de representação, arredondando para baixo? Considere números reais com 12 dígitos decimais após a vírgula.

Ex. 2.12 Repita o Ex. 2.11 empregando 8 bits na parte fracionária. Qual a magnitude dos erros de representação, arredondando para baixo?

Ex. 2.13 Repita o Ex. 2.11 empregando 12 bits na parte fracionária. Qual a magnitude dos erros de representação, arredondando para baixo?

Ex. 2.14 Estime o erro máximo de representação para reais representados com n bits na parte fracionária.

Ex. 2.15 Estime o erro mínimo de representação para reais representados com n bits na parte fracionária.

Ex. 2.16 Considere o sistema do Exemplo 2.9. Uma representação com 12 bits atenderia à especificação? Qual é o número mínimo de bits necessário? Justifique sua resposta. Neste tipo de projeto, frequentemente é uma boa ideia empregar uma representação interna – que fica escondida do usuário – que seja melhor do que a representação externa – aquela que fica visível ao usuário.

Ex. 2.17 Ainda com relação ao Exemplo 2.9, suponha que o aparelho será empregado numa espaçonave que se dirige a Mercúrio. O controle deverá operar na faixa de -220C a $+420$C, e manter a precisão de um décimo de grau. Quantos bits são necessários para representar a parte inteira e quantos para a parte fracionária? Justifique sua resposta.

Referências Bibliográficas

[Ash08] Peter J Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 3rd edition, 2008. ISBN 978-0-12-088785-9.

[BJ97] Gerrit A Blaauw and Frederick P Brooks Jr. *Computer Architecture: Concepts and Evolution*. Addison-Wesley, 1997. ISBN 0201105578.

[Bob87] Leonard S Bobrow. *Elementary Linear Circuit Analysis*. Holt, Rinehart & Winston, 1987. ISBN 0030072980.

[Bro04] Stuart Brorson. *Circuit simulation using gEDA and SPICE – HOWTO*, 2004. <http://www.brorson.com/gEDA/SPICE/intro.html>, 8/5/2012.

[Car03] Nicholas Carter. *Arquitetura de Computadores*. Coleção Schaum. Bookman, 2003. ISBN 853630250x.

[Cla80] Wesley A Clark. From electron mobility to logical structure: A view of integrated circuits. *ACM Computing Surveys*, 12(3):325–356, Set 1980.

[Fle97] William I Fletcher. *An Engineering Approach to Digital Design*. Prentice Hall, 1997. ISBN 9780132776998.

[Hex15] Roberto A Hexsel. cMIPS – a synthesizable VHDL model for the classical five stage pipeline. Repositório de *software*, Depto de Informática, UFPR, 2015. Disponível em <https://github.com/rhexsel/cmips>.

[Hex17] Roberto A Hexsel. Why it is so hard to write "system software"? To be submitted, 2017.

[HJS00] M D Hill, N P Jouppi, and G S Sohi. *Readings in Computer Architecture*. Morgan Kaufmann, 2000. ISBN 1558605398.

[HP90] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1st edition, 1990. ISBN 1558600698.

[HP12] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 5th edition, 2012.

[HU79] John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN 020102988X.

[Hwa93] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, 1993. ISBN 0070316228.

[JNW08] B L Jacob, S W Ng, and D T Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008. ISBN 0123797513.

[Kat94] Randy H Katz. *Contemporary Logic Design*. Benjamin-Cummings, 1994. ISBN 0805327037.

[KB04] Randy H Katz and Gaetano Borriello. *Contemporary Logic Design*. Prentice Hall, 2004. ISBN 978-0201308570.

[KL96] Sung-Mo Kang and Yusuf Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design*. McGraw-Hill, 1996. ISBN 0070380465.

[Koh78] Zvi Kohavi. *Switching and Finite Automata Theory*. Tata McGraw-Hill, 2nd edition, 1978. ISBN 0070993874.

[Kor01] Israel Koren. *Computer Arithmetic Algorithms*. A K Peters, 2nd edition, 2001. ISBN 1568811608.

[KP90] Al Kelley and Ira Pohl. *A Book on C, Programming in C*. Benjamin/Cummings, 2nd edition, 1990. ISBN 0805300600.

[KR88] Brian W Kernighan and Dennis M Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988. ISBN 9780131103628.

[Lam84] Butler W Lampson. Hints for computer system design. *IEEE Software*, pages 11–28, Jan 1984.

[Lip64] Seymour Lipschutz. *Set Theory and Related Topics*. Schaum Publishing, 1964.

[Man02] M Morris Mano. *Digital Design*. Prentice Hall, 3rd edition, 2002. ISBN 01306211218.

[MIP05a] MIPS. *MIPS32 Architecture for Programmers, Volume I: Introduction to the MIPS32 Architecture*, 2005.

[MIP05b] MIPS. *MIPS32 Architecture for Programmers, Volume II: The MIPS32 Instruction Set*, 2005.

[MIP05c] MIPS. *MIPS32 Architecture for Programmers, Volume III: The MIPS32 Privileged Resource Architecture*, 2005.

[MK00] M Morris Mano and Charles R Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, 2nd edition, 2000. ISBN 0130124680.

[Mou01] Arnaldo V Moura. *Especificações em Z*. Editora da Unicamp, 2001. ISBN 8526805754.

[Ped10] Volnei A Pedroni. *Circuit Design and Simulation with VHDL*. MIT Press, 2nd edition, 2010. ISBN 978-0262014335.

[PH94] David A Patterson and John L Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufmann, 1994. ISBN 155860281X.

[PH14] David A Patterson and John L Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufmann, 5th edition, 2014.

[PP03] Yale N Patt and Sanjay J Patel. *Introduction to Computing Systems: From Bits and Gates to C and Beyond*. McGraw-Hill, 2nd edition, 2003.

[PT96] David Pellerin and Douglas Taylor. *VHDL Made Easy!* Prentice Hall, 1996. ISBN 0136507638.

[RCN03] Jan M Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits – A Design Perspective*. Prentice Hall, 2nd edition, 2003. ISBN 0130909963.

[San90] Jeff W Sanders. Lectures on the foundations of hardware design. Lecture notes, Programming Research Group, Oxford University Computing Laboratory, 1990.

[Sco65] Ronald E Scott. *Elements of Linear Circuits*. Addison-Wesley, 1965. ISBN 0201068427.

[SCO96] The Santa Cruz Operation SCO. *System V Application Binary Interface – MIPS RISC Supplement*, 3rd edition, 1996.

[Spi89] J M Spivey. *The Z Notation*. Prentice Hall, 1989. ISBN 013983768X.

[SS90] Adel S Sedra and Kenneth C Smith. *Microeletronic Circuits*. Holt, Rinehart & Winston, 3rd edition, 1990. ISBN 003051648X.

[Swe07] Dominic Sweetman. *See MIPS Run – Linux*. Morgan Kaufmann, 2nd edition, 2007. ISBN 0120884216.

[Tau82] Herbert Taub. *Digital Circuits and Microprocessors*. McGraw-Hill, 1982. ISBN 0070629455.

[TS89] Herbert Taub and Donald Schilling. *Digital Integrated Electronics*. McGraw-Hill, 1989. ISBN 007Y857881.

[vN93] John von Neumann. First draft of a report on the EDVAC. *IEEE Ann History of Computing*, 15(4):27–75, Out 1993.

[WE82] Jim Welsh and John Elder. *Introduction to Pascal*. Prentice-Hall, 2nd edition, 1982. ISBN 0134915496.

[WH10] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 4th edition, 2010. ISBN 0321547748.

[Wol05] Wayne Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann, 2005. ISBN 0123694590.

Índice Remissivo