J. GLENN BROOKSHEAR

# CIÊNCIA DA COMPUTAÇÃO

## UMA VISÃO ABRANGENTE

7a EDIÇÃO

Bookman®

Obra originalmente publicada sob o título
*Computer Science: An Overview, 7/ed;*
BROOKSHEAR, J. GLENN

Tradução autorizada a partir do original em língua inglesa,
publicado por Pearson Education, Inc., sob o selo Addison Wesley.
ISBN 0-201-78130-1

*Capa:*
GUSTAVO MACRI

*Leitura final:*
LETÍCIA V. ABREU DORNELLES

*Supervisão editorial:*
ARYSINHA JACQUES AFFONSO e DENISE WEBER NOWACZYK

*Editoração e filmes:*
WWW.GRAFLINE.COM.BR





SÃO PAULO
Av. Angélica, 1091 - Higienópolis
01227-100 São Paulo SP
Fone (11) 3665-1100 Fax (11) 3667-1333

SAC 0800-703-3444

IMPRESSO NO BRASIL
*PRINTED IN BRAZIL*

# Sumário

## PARTE DOIS: *SOFTWARE* 109

CAPÍTULO 0

# *Introdução*



A Ciência da Computação é a disciplina que busca construir uma base científica para diversos tópicos, tais como a construção e a programação de computadores, o processamento de informações, as soluções algorítmicas de problemas e o processo algorítmico propriamente dito. Nesse sentido, estabelece os fundamentos para as aplicações computacionais existentes, assim como as bases para as futuras aplicações. Essa extensão significa que não podemos aprender a Ciência da Computação pelo simples estudo de alguns tópicos isolados, ou pela utilização das ferramentas computacionais atuais. Em vez disso, para entender a Ciência da Computação, é preciso compreender o escopo e a dinâmica da grande variedade de tópicos.

Este livro foi projetado para prover tais fundamentos. Apresenta uma introdução integrada aos tópicos que compõem os currículos universitários típicos desta área. O livro pode servir, portanto, como referência para estudantes que se iniciam na Ciência da Computação, além de ser uma fonte de consulta para outros que estejam em busca de uma introdução a esta ciência, que fundamenta a sociedade computadorizada dos dias de hoje.

## 0.1 O estudo de algoritmos

Começamos com o conceito mais fundamental da Ciência da Computação — o de algoritmo. Informalmente, um algoritmo é um conjunto de passos que definem a forma como uma tarefa é executada.[1] Por exemplo, há algoritmos para a construção de aeromodelos (expressos na forma de folhas de instruções de montagem), para a operação de lavadoras de roupa (normalmente afixados no lado interno da tampa da máquina), para tocar música (expressos na forma de partituras) e para a execução de truques de mágica (Figura 0.1).

Antes que uma máquina possa executar uma tarefa, um algoritmo que a execute deve ser descoberto e representado em uma forma compatível com a máquina. Uma representação compatível com a máquina de um algoritmo é chamada **programa**. Os programas e os algoritmos que eles representam são coletivamente chamados ***software***, em contraste com a máquina propriamente dita, que é conhecida como ***hardware***.

O estudo dos algoritmos começou como um objeto da matemática. De fato, a procura por algoritmos era uma atividade significativa dos matemáticos muito antes do desenvolvimento dos computadores. O objetivo principal era descobrir um conjunto único de diretrizes que descrevessem como todos os

**Efeito:** O artista coloca algumas cartas de baralho sobre uma mesa, com a face voltada para baixo, e as embaralha bem enquanto as distribui sobre a mesa. Então, à medida que a audiência solicita cartas, especificando a sua cor (vermelha ou preta), o mágico as vira, exatamente da cor solicitada.

**Segredo e Truque:**

**Passo 1.** Selecionar dez cartas vermelhas e dez pretas. Fazer duas pilhas com essas cartas, abertas, separando-as de acordo com a sua cor.

**Passo 2.** Anunciar que você selecionou algumas cartas vermelhas e algumas cartas pretas.

**Passo 3.** Apanhar as cartas vermelhas. Com o pretexto de formar com elas uma pilha pequena, segurá-las, voltadas para baixo, com a mão esquerda, e, com o polegar e o indicador da mão direita, empurrar para trás cada extremidade da pilha, de modo que cada carta fique ligeiramente encurvada para trás. Então, colocar sobre a mesa a pilha de cartas vermelhas, voltadas para baixo, e dizer: "Aqui, nesta pilha, estão as cartas vermelhas".

**Passo 4.** Apanhar as cartas pretas. De forma análoga ao que foi feito no passo 3, imprimir a essas cartas uma leve curvatura para a frente. Então, devolvê-las à mesa, com a face voltada para baixo, e dizer: "E aqui estão as cartas pretas".

**Passo 5.** Imediatamente após devolver as cartas pretas à mesa, usar as duas mãos para misturar as cartas vermelhas e pretas (ainda voltadas para baixo, da mesma forma como foram anteriormente espalhadas na mesa). Explicar que isso está sendo feito para que as cartas fiquem bem embaralhadas.

**Passo 6.** Enquanto houver cartas na mesa, executar repetidamente os seguintes passos:

**6.1** Pedir que a audiência solicite uma carta de uma cor específica (vermelha ou preta).

**6.2** Se a cor solicitada for vermelha e houver uma carta voltada para baixo, com formato côncavo, abrir essa carta e dizer: "Aqui está uma carta vermelha".

**6.3** Se a cor solicitada for preta e houver uma carta que tenha aparência convexa, abrir a carta e dizer: 'Aqui está uma carta preta".

**6.4** Caso contrário, declarar que não há mais cartas da cor solicitada e abrir as cartas restantes para provar sua afirmação.

**Figura 0.1** Um algoritmo para executar um truque de mágica.

[1]Mais precisamente, um algoritmo é um conjunto ordenado e não-ambíguo de passos executáveis que definem uma atividade finita. Esses detalhes serão discutidos no Capítulo 4.

problemas de um determinado tipo poderiam ser resolvidos. Um dos mais conhecidos exemplos dessa antiga pesquisa é o algoritmo de divisão, para encontrar o quociente de dois números inteiros compostos de vários dígitos. Outro exemplo é o algoritmo de Euclides, descoberto pelo matemático grego de mesmo nome, que permite calcular o máximo divisor comum de dois inteiros positivos (Figura 0.2).

Uma vez descoberto um algoritmo que execute uma dada tarefa, sua execução já não dependerá do conhecimento dos princípios nos quais se baseia, restringindo-se apenas a seguir as instruções estabelecidas. (Pode-se seguir o algoritmo da divisão longa, para calcular um quociente, ou o algoritmo de Euclides, para obter o máximo divisor comum, sem necessitar compreender os princípios do seu funcionamento.) Em outras palavras, o algoritmo constitui uma codificação do raciocínio necessário à resolução do problema.

**Descrição:** Este algoritmo pressupõe que sejam fornecidos dois inteiros positivos e calcula o seu máximo divisor comum.

**Procedimento:**

**Passo 1.** Atribuir, inicialmente, a M e N os valores correspondentes ao maior e menor dos dois números inteiros positivos fornecidos, respectivamente.

**Passo 2.** Dividir M por N, e chamar de R o resto da divisão.

**Passo 3.** Se R não for 0, atribua a M o valor de N, a N o valor de R e retorne ao passo 2; caso contrário, o máximo divisor comum será o valor corrente de N.

**Figura 0.2** O algoritmo de Euclides, que calcula o máximo divisor comum de dois inteiros positivos.

É por meio desta capacidade de captar e transferir inteligência mediante os algoritmos que são construídas máquinas que exibem comportamento inteligente. Por conseguinte, o nível de inteligência demonstrado pelas máquinas fica limitado pela inteligência que um algoritmo é capaz de transportar. Somente quando for possível obter um algoritmo que possa controlar a operação de uma tarefa será viável construir alguma máquina capaz de executá-la. Por outro lado, se não houver algoritmo capaz de executar tal tarefa, então a sua realização excederá as capacidades da máquina.

A tarefa do desenvolvimento de algoritmos torna-se, dessa forma, vital no campo da computação e, por isso, boa parte da Ciência da Computação se ocupa de assuntos a ela relacionados. No entanto, mediante estudo de alguns desses tópicos, adquire-se uma melhor compreensão da abrangência da Ciência da Computação. Um deles estuda como projetar novos algoritmos, uma questão muito semelhante ao problema geral de como solucionar problemas genéricos. Descobrir um algoritmo para solucionar um problema corresponde, essencialmente, a descobrir uma solução para esse problema. Logo, estudos nesta área da Ciência da Computação têm forte relação com outras áreas, tais como a da psicologia de resolução de problemas humanos e a das teorias de educação. Algumas dessas idéias serão consideradas no Capítulo 4.

Uma vez descoberto um algoritmo para solucionar um problema, o passo seguinte consiste em representá-lo de forma apropriada para que seja transmitido a alguma máquina, ou para que seja lido por outros seres humanos. Isto significa que se torna necessário transformar o algoritmo conceitual em um conjunto de instruções fácil de compreender e que elas sejam representadas sem ambigüidade. Sob esta perspectiva, estudos fundamentados em um conhecimento lingüístico e gramatical conduziram a uma grande diversidade de esquemas para a representação de algoritmos conhecidos como **linguagens de programação**, baseados em várias abordagens ao processo de programação conhecidas como paradigmas de programação. Algumas dessas linguagens e os paradigmas em que se baseiam serão tratados no Capítulo 5.

Uma vez que a tecnologia dos computadores tem sido aplicada a problemas cada vez mais complexos, os cientistas da computação descobriram que o projeto de grandes sistemas de *software* envolve mais do que o desenvolvimento de algoritmos individuais que executem as atividades requeridas. Acarreta também o projeto das interações entre os componentes. Para lidar com tais complexidades, os cientistas da computação se voltaram ao campo bem-estabelecido da engenharia, com o intuito de encontrar as ferramentas adequadas a esses problemas. O resultado é o ramo da Ciência da Computação chamado engenharia de *software*, que hoje se abastece em diversos campos, como engenharia, gerência de projetos, gerência de pessoal e projetos de linguagens de programação.

Outra área importante de Ciência da Computação é a que se ocupa do projeto e da construção de máquinas para executar algoritmos. Esses assuntos são tratados nos Capítulos 1 e 2. Embora nosso estudo sobre arquitetura de computadores incorpore algumas discussões tecnológicas, o objetivo não é o de dominar minuciosamente a forma como tal arquitetura é implementada por meio de circuitos eletrônicos. Isso desviaria demasiadamente para o campo da engenharia elétrica. Além disso, da mesma maneira que as antigas calculadoras, constituídas de engrenagens, deram lugar a dispositivos eletrônicos, a atual eletrônica pode vir a ser em breve substituída por outras tecnologias, entre as quais a ótica se apresenta como uma forte candidata. Nosso objetivo é ter uma compreensão da atual tecnologia que seja suficiente para que possamos observar suas ramificações nas máquinas de hoje, bem como a sua influência no desenvolvimento da própria Ciência da Computação.

O ideal seria que a arquitetura de computadores fosse produto exclusivo do nosso conhecimento dos processos algorítmicos, e que não fosse limitada pelas capacidades tecnológicas: em vez de permitir que a tecnologia forneça diretrizes para o projeto das máquinas e, portanto, para o modo como serão representados os algoritmos, gostaríamos que o nosso conhecimento de algoritmos funcionasse como força motriz por trás da arquitetura moderna das máquinas. Com os avanços da tecnologia, este sonho vai se tornando cada vez mais uma realidade. Hoje, é possível construir máquinas que permitem representar algoritmos como seqüências múltiplas de instruções, simultaneamente executadas, ou como padrões de conexões entre numerosas unidades de processamento, funcionando de forma parecida com o modo como nossas mentes representam a informação pelas conexões existentes entre os neurônios (Capítulo 10).

Outro contexto em que estudamos a arquitetura de computadores se refere ao armazenamento de dados e seu acesso. Neste ponto, as características internas de uma máquina freqüentemente se refletem nas suas características externas, as quais, bem como as formas de evitar seus efeitos indesejáveis, são objeto de estudo dos Capítulos 1, 7, 8 e 9.

Outro ponto fortemente relacionado com a concepção de máquinas de computação é o projeto da interface entre o computador e o mundo externo. Por exemplo, de que forma os algoritmos deverão ser inseridos na máquina, e de que maneira se pode informar a esta qual algoritmo deve ser executado? Solucionar tais problemas para ambientes nos quais se espera que a máquina preste serviços variados requer que sejam solucionados muitos outros problemas, que envolvem a coordenação de atividades e a alocação de recursos. Algumas dessas soluções são discutidas em nosso estudo sobre sistemas operacionais, no Capítulo 3.

À medida que se exigiu das máquinas a execução de tarefas cada vez mais inteligentes, a Ciência da Computação passou a buscar inspiração no estudo da inteligência humana. Acredita-se que, compreendendo o raciocínio da mente, poderíamos projetar algoritmos que imitassem tais processos, transferindo esta capacidade para as máquinas. Como resultado, surge a Inteligência Artificial, disciplina que se apóia fortemente em pesquisas das áreas da psicologia, biologia e lingüística. Discutimos no Capítulo 10 alguns desses tópicos da Inteligência Artificial.

A busca de algoritmos para controlar tarefas cada vez mais complexas conduz a questões relativas às suas limitações. A inexistência de algoritmos que resolvam um dado problema implica que ele não pode ser resolvido por uma máquina. Uma tarefa será considerada algorítmica se puder ser descrita por um algoritmo. Isto é, as máquinas são capazes de resolver apenas os problemas passíveis de solução algorítmica.

A constatação de que existem problemas sem solução algorítmica emergiu como um assunto na matemática na década de 1930, com a publicação do teorema da incompleteza de Kurt Gödel. Este teorema declara essencialmente que em qualquer teoria matemática baseada no sistema de aritmética tradicional existem afirmações que não podem ser provadas nem refutadas. Em suma, um estudo completo de nosso sistema aritmético está além da capacidade das atividades algorítmicas.

O desejo de estudar as limitações dos métodos algorítmicos, decorrente da descoberta de Gödel, levou os matemáticos a projetar máquinas abstratas para executar algoritmos (isso ocorreu antes que a tecnologia fosse capaz de prover as máquinas reais para a investigação) e a estudar as potencialidades teóricas de tais máquinas hipotéticas. Atualmente, o estudo de algoritmos e máquinas constitui a espinha

dorsal da Ciência da Computação. Serão discutidos alguns dos tópicos relacionados a esta área no Capítulo 11.

## 0.2 As origens das máquinas computacionais

As máquinas abstratas idealizadas pelos matemáticos nos primeiros anos do século XX constituem importante parcela da árvore genealógica dos computadores modernos. Outros ramos desta árvore advêm de épocas mais remotas. De fato, é longa a história da busca pela máquina que executasse tarefas algorítmicas.

Um dos primeiros dispositivos de computação foi o ábaco. Sua história remonta às antigas civilizações grega e romana. Essa máquina é bastante simples e consiste em um conjunto de contas que correm em barras fixadas em uma armação retangular. As contas são movidas de um lado para outro sobre as barras, e suas posições representam valores armazenados. É por meio das posições das contas que esse "computador" representa e armazena dados. A execução do algoritmo é comandada por um operador humano. Assim, o ábaco constitui apenas um sistema de armazenamento de dados e deve ser combinado com uma pessoa para constituir uma máquina computacional completa.

Mais recentemente, o projeto das máquinas computacionais passou a se basear na tecnologia de engrenagens. Entre os inventores, estavam Blaise Pascal (1623-1662) da França, Gottfried Wilhelm Leibniz (1646-1716) da Alemanha e Charles Babbage (1792-1871) da Inglaterra. Essas máquinas representavam os dados por meio do posicionamento de engrenagens, e a introdução dos dados era mecânica, consistindo no estabelecimento conjunto das posições das diversas engrenagens. Os dados de saída das máquinas de Pascal e Leibniz eram obtidos observando-se as posições finais das engrenagens, da mesma forma como são lidos números nos hodômetros dos automóveis. Entretanto, Babbage projetou uma máquina que ele denominou máquina analítica, a qual podia imprimir em papel os dados de saída, eliminando assim erros de transcrição.

Tornou-se necessário um aumento da flexibilidade das máquinas para facilitar a execução dos algoritmos. A máquina de Pascal foi construída para efetuar somente o algoritmo de adição. Para tanto, uma seqüência apropriada de instruções foi embutida na própria estrutura da máquina. De modo semelhante, a máquina de Leibniz tinha seus algoritmos firmemente incorporados à sua arquitetura, embora apresentasse uma série de operações aritméticas que podiam ser selecionadas pelo operador. A máquina das diferenças de Babbage (só foi construído um modelo de demonstração), ao contrário, podia ser modificada de modo a realizar diversos cálculos, mas a sua máquina analítica (ele nunca obteve os recursos necessários para sua construção) foi projetada para ler as instruções sob a forma de furos em cartões de papel. Assim, a máquina analítica de Babbage era programável. Com efeito, sua assistente Augusta Ada Byron é conhecida como a primeira programadora do mundo.

### *Augusta Ada Byron*

Desde que o Departamento de Defesa dos EUA a homenageou usando seu nome em uma linguagem de programação, Augusta Ada Byron, a Condessa de Lovelace, tem sido muito comentada pela comunidade da computação. Ada Byron teve uma vida um tanto trágica em menos de 37 anos (1815-1852), complicada por saúde débil e o fato de ela não ter sido conformista em uma sociedade que limitava o papel profissional das mulheres. Ela ficou fascinada com as máquinas de Charles Babbage quando presenciou a demonstração de um protótipo de sua máquina das diferenças, em 1833. Sua contribuição à Ciência da Computação envolveu a tradução do francês para o inglês de um artigo que discutia os projetos de Babbage para a Máquina Analítica. Nessa tradução, Babbage a encorajou a anexar um adendo que descrevesse as aplicações da máquina e contivesse exemplos de como poderia ser programada para executar diversas tarefas. O entusiasmo de Babbage pelo trabalho de Ada Byron (ele fez numerosas sugestões e aparentemente contribuiu com alguns dos exemplos) era motivado pela esperança de que, com a publicação, teria o suporte financeiro para a construção da Máquina Analítica. (Sendo filha de Lord Byron, Ada Byron mantinha *status* de celebridade e tinha contatos potencialmente significativos em termos financeiros.) Esse suporte jamais se concretizou, mas o adendo de Ada Byron sobreviveu e é considerado um repositório dos primeiros programas de computador. Assim, ela é reconhecida atualmente como a primeira programadora do mundo.

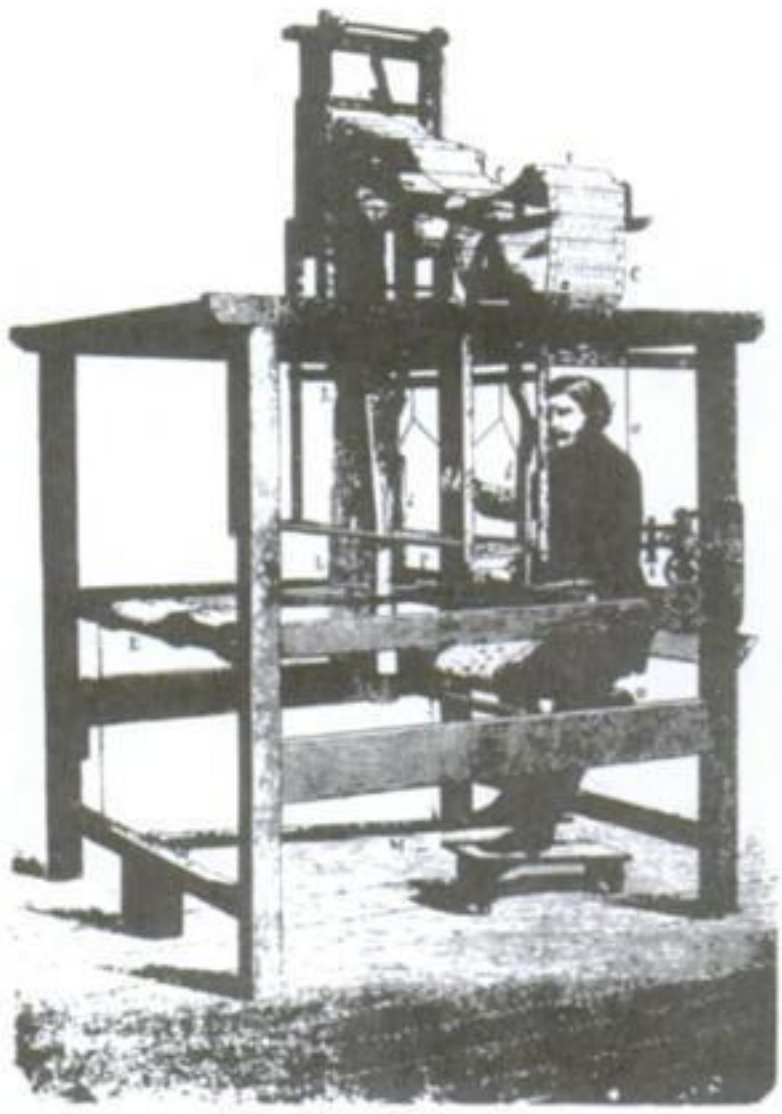


**Figura 0.3** O tear de Jacquard. (Cortesia de International Business Machines Corporation. Proibido o uso sem permissão.)

A idéia de comunicar à máquina um algoritmo por intermédio de orifícios perfurados em papel não foi de Babbage. Em 1801, na França, Joseph Jacquard (1752-1834) aplicara uma técnica semelhante para controlar teares (Figura 0.3). Ele desenvolveu um tear no qual os passos a executar durante o processo de tecelagem eram determinados pelos padrões de orifícios em cartões de papel. Desta maneira, o algoritmo executado pela máquina poderia ser facilmente alterado, para produzir diferentes padronagens nos tecidos. Posteriormente, Herman Hollerith (1860-1929) aplicou a idéia de representar informação por meio de cartões perfurados para acelerar o processo de tabulação do censo americano de 1890. (Esse trabalho levou à criação da IBM.) Esses cartões passaram a ser conhecidos como cartões perfurados e sobreviveram até a década de 1970, como um meio popular de comunicação com os computadores. De fato, a técnica vive até os dias atuais, como atestam as questões de legitimidade de votos levantadas nas eleições presidenciais de 2000 nos EUA.

Com a tecnologia disponível na época, era financeiramente inviável a produção das complexas máquinas de engrenagens de Pascal, Leibniz e Babbage, e elas não se popularizaram. Somente com o advento da eletrônica, no início do século XX, a barreira foi ultrapassada. Como exemplos desse avanço, citam-se a máquina eletromecânica de George Stibitz, concluída em 1940 no Bell Laboratories, e o Mark I, concluído em 1944 na Harvard University por Howard Aiken e um grupo de engenheiros da IBM (Figura 0.4). Essas máquinas fizeram intenso uso de relés mecânicos eletronicamente controlados. Por essa razão, tornaram-se obsoletas logo depois de construídas, pois outros pesquisadores estavam então aplicando a tecnologia de válvulas na construção de computadores digitais totalmente eletrônicos. Aparentemente, a primeira dessas máquinas foi a de Atanasoff-Berry, construída de 1937 a 1941 no Iowa State College (atual Iowa State University) por John Atanasoff e seu assistente Clifford Berry. Outra pioneira foi a máquina conhecida como Colossus, construída na Inglaterra sob a direção de Tommy Flowers para decodificar mensagens alemãs durante o final da Segunda Guerra Mundial. (De fato, foram construídas dez dessas máquinas, mas os segredos militares, bem como as questões de segurança nacional, impediram que elas constassem na "árvore genealógica dos computadores".) Logo surgiram outras máquinas mais flexíveis, como o ENIAC (*electronic numerical integrator and calculator*, ou integrador e calculador numérico eletrôni-

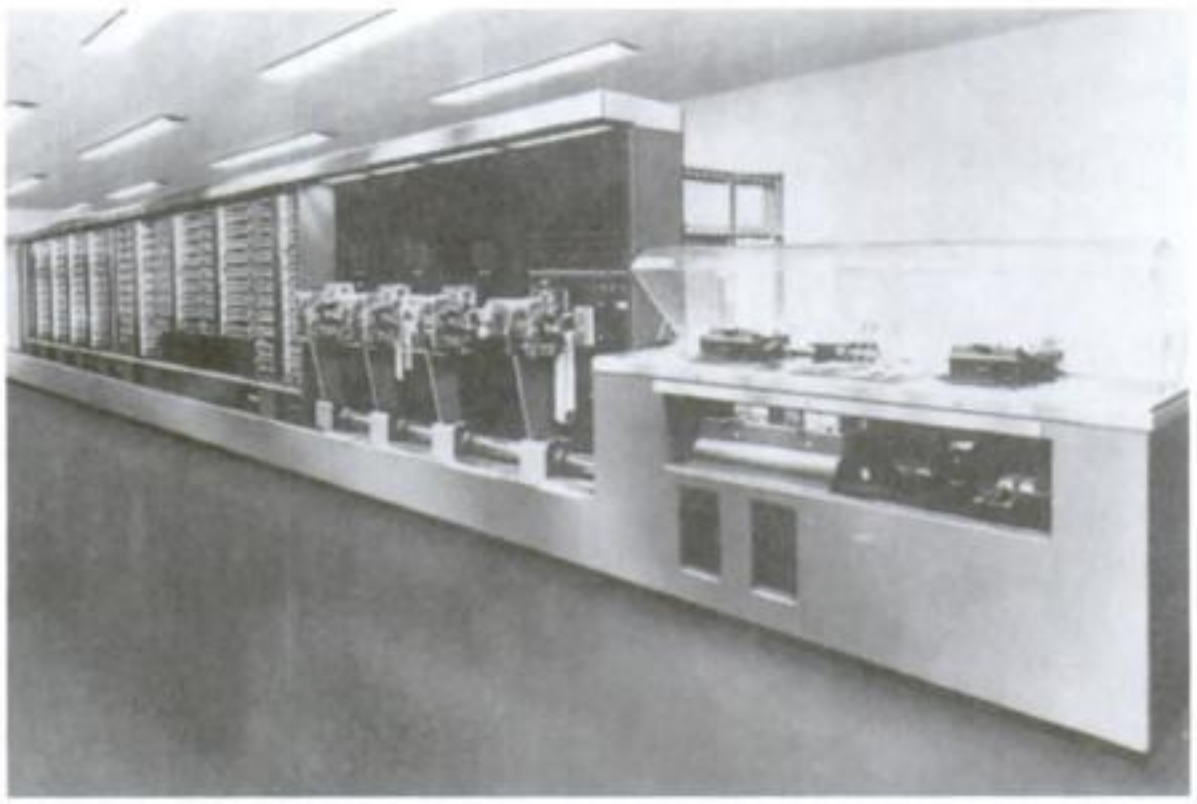


**Figura 0.4** O computador Mark I.

co), desenvolvido por John Mauchly e J. Presper Eckert na Moore School of Electrical Engineering, University of Pennsylvania.

A partir daí, a história das máquinas computacionais foi a de uma tecnologia emergente em franco avanço, tendo sido seus marcos mais significativos a invenção dos transistores, o desenvolvimento subseqüente dos circuitos integrados, o estabelecimento da comunicação por satélite e os avanços na tecnologia ótica. Hoje em dia, os computadores de mesa (bem como seus primos menores e portáteis conhecidos como *laptops*) possuem mais capacidade computacional do que as máquinas da década de 1940, que ocupavam uma sala inteira. Além disso, podem trocar informações rapidamente via sistemas globais de comunicação.

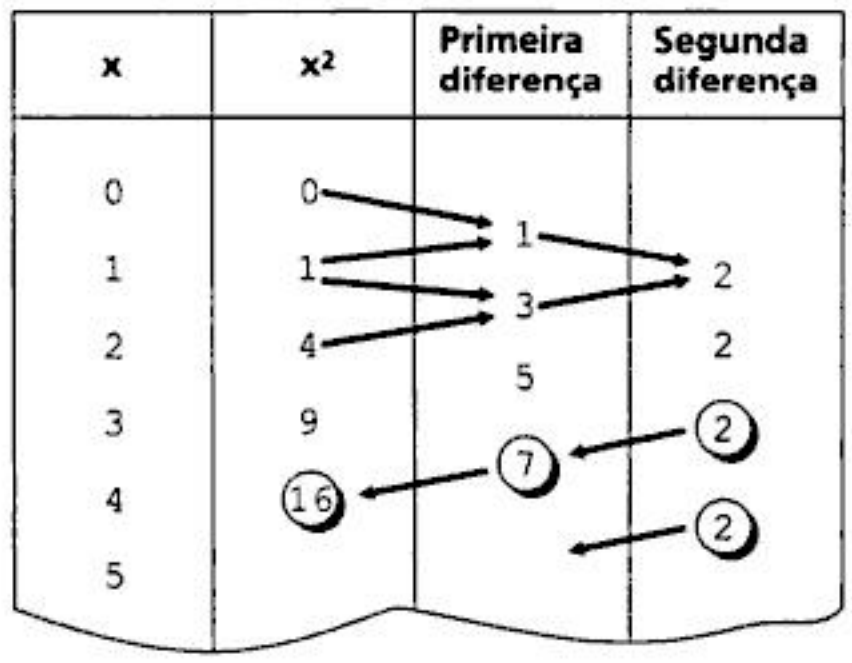
As origens dessas pequenas máquinas estão ligadas àqueles que tinham o computador como passatempo e começaram a fazer experiências com esses computadores feitos em casa logo após o desenvolvimento das grandes máquinas de pesquisa da década de 1940. Foi nessa atividade *underground* que Steve Jobs e Stephen Wosniak construíram um computador caseiro comercialmente viável e em 1976 fundaram a Apple Computer, Inc. para produzir e comercializar os seus produtos. Embora os produtos da Apple fossem populares, sua aceitação não foi grande na comunidade de negócios, que continuava procurando a bem-estabelecida IBM para suprir a maioria de suas necessidades computacionais.

Em 1981, a IBM introduziu o seu primeiro computador de mesa, chamado computador pessoal, ou simplesmente PC, cujo *software* subjacente foi desenvolvido por uma nova e esforçada companhia chamada Microsoft. O PC teve sucesso instantâneo e legitimou nas mentes da comunidade de negócios o computador de mesa como mercadoria estabelecida. Hoje em dia, o termo PC é largamente utilizado para se referir a todas essas máquinas (de vários fabricantes) cujos projetos são desdobramentos do computador pessoal da IBM, e cuja maioria é comercializada com o *software* da Microsoft. Às vezes, contudo, o termo PC é usado como sinônimo de *computador de mesa*.

A disponibilidade dos computadores de mesa colocou em destaque a tecnologia de computação na sociedade atual. Com efeito, a tecnologia da computação é tão predominante agora que saber como utilizá-la é fundamental para ser membro da sociedade moderna. É por meio dessa tecnologia que milhões de indivíduos têm acesso ao sistema de conexão global, conhecido como Internet, sistema esse que vem influenciando o estilo de vida e de comércio em base mundial.

## *A máquina das diferenças de Babbage*

Os projetos das máquinas de Charles Babbage foram verdadeiramente os precursores dos projetos dos computadores modernos. Se tivesse a tecnologia conseguido produzir essas máquinas de uma maneira viável economicamente e as demandas por processamento de dados fossem comparáveis às atuais, as idéias de Babbage poderiam ter levado à revolução da computação ainda no século XIX. Na realidade, apenas um modelo de demonstração de sua Máquina das Diferenças foi construído enquanto ele viveu. Essa máquina determinava valores numéricos computando diferenças sucessivas. Podemos entender essa técnica considerando o cálculo dos quadrados de números inteiros. Começamos com o conhecimento de que o quadrado de 0 é 0; o de 1, 1; o de 2, 4 e o de 3, 9. Com isso, podemos determinar o quadrado de 4 da seguinte maneira: primeiro calculamos as diferenças dos quadrados já conhecidos: $1^2 - 0^2 = 1$, $2^2 - 1^2 = 3$, $3^2 - 2^2 = 5$. Agora, calculamos as diferenças entre esses resultados: $3 - 1 = 2$ e $5 - 3 = 2$. Note que essas diferenças são iguais a 2. Pressupondo que essa consistência continue (a matemática pode provar isso), concluímos que a diferença entre $(4^2 - 3^2)$ e $(3^2 - 2^2)$ também é 2. Assim, $(4^2 - 3^2)$ deve ser 2 mais do que $(3^2 - 2^2)$, logo $4^2 - 3^2 = 7$ e, assim, $4^2 = 7 + 3^2 = 16$. Agora que já possuímos o quadrado de 4, podemos continuar o procedimento e calcular o quadrado de 5 baseados nos valores de $1^2$, $2^2$, $3^2$ e $4^2$. (Embora uma discussão mais aprofundada de diferenças sucessivas esteja além do escopo de nosso estudo, os estudantes de cálculo podem observar que o exemplo precedente se baseia no fato de que a derivada segunda de $y = x^2$ é uma linha reta.)



## 0.3 A ciência dos algoritmos

Dada a capacidade limitada de armazenamento de dados, o nível de detalhamento e os demorados procedimentos de programação, foi necessário restringir a complexidade dos algoritmos aplicados nas primeiras máquinas. Contudo, à medida que essas limitações foram superadas, as máquinas começaram a executar tarefas cada vez mais extensas e complexas. Conforme as tentativas de expressar a composição destas tarefas na forma algorítmica exigiam mais das habilidades da mente humana, crescentes esforços de pesquisa foram direcionados ao estudo dos algoritmos e do processo de programação.

Foi nesse contexto que começaram a dar frutos os trabalhos teóricos dos matemáticos. De fato, como decorrência do teorema da incompleteza de Gödel, os matemáticos já haviam começado a investigar as questões relativas aos processos algorítmicos, que eram levantadas pela nova tecnologia. Com isso, ficou estabelecido o surgimento da disciplina conhecida como *Ciência da Computação*.

A disciplina que então se criava se estabeleceu com o nome de Ciência dos Algoritmos. Como vimos, seu escopo é amplo pois originou-se a partir de matérias diversas, como matemática, engenharia, psicologia, biologia, administração empresarial e lingüística. Muitos dos tópicos dessa ciência serão discutidos nos capítulos subseqüentes. Em cada situação, o objetivo é introduzir as idéias principais do tópico, os mais recentes tópicos de pesquisa e algumas das técnicas empregadas para aumentar o conhecimento da área. É importante distinguir a Ciência da Computação das suas aplicações — distinção análoga à diferenciação entre a física e a engenharia mecânica. A física é a ciência que procura explicar a relação entre força, massa e aceleração e a engenharia mecânica, a aplicação dessa ciência. É necessário que o engenheiro mecânico entenda de física, mas, dependendo de sua especialidade, também deve saber as várias gradações do aço, a composição do concreto e a disponibilidade de posições no mercado para diversos produtos. Entretanto, o físico está interessado nas discrepâncias das teorias de Newton e como elas foram corrigidas nas considerações de Einstein. Portanto, não se deve esperar que um estudo de física inclua discussões sobre as limitações de carga em roldanas disponíveis no mercado, ou sobre as regulamentações governamentais a respeito dos cintos de segurança de automóveis.

De modo semelhante, nosso estudo não incluirá instruções para usar um programa particular de planilha, ou para instalar um navegador para a Internet. Isso não significa que esses tópicos não sejam importantes, apenas não fazem parte do nosso assunto. Nossa discussão a respeito da programação também não é focalizada no desenvolvimento das habilidades de programação em uma linguagem específica. Em vez disso, ela se concentra nos princípios que estão por trás das ferramentas atuais de programação, em como elas se desenvolveram e nos problemas que a pesquisa atual tenta solucionar.

À medida que se avança neste estudo de tópicos, é fácil perder a visão geral. Portanto, reunimos nossas idéias identificando algumas perguntas que definem a Ciência da Computação e estabelecendo o enfoque a ser dado para o seu estudo.

- Quais problemas podem ser solucionados por meio de processos algorítmicos?
- De que modo o processo de descoberta de algoritmos pode ser facilitado?
- Como se pode melhorar as técnicas de representação e comunicação de algoritmos?
- Como o nosso conhecimento de algoritmos e da tecnologia pode ser aplicado na obtenção de máquinas algorítmicas melhores?
- De que maneira as características de diferentes algoritmos podem ser analisadas e comparadas?

Note-se que o tema comum a todas essas perguntas é o estudo de algoritmos (Figura 0.5).

## 0.4 O papel da abstração

O conceito da abstração permeia de tal forma o estudo da Ciência da Computação e do projeto de sistemas de computadores que se torna necessário apresentá-lo neste capítulo introdutório. O termo **abstração**, como usado aqui, refere-se à distinção entre as propriedades externas de uma entida-

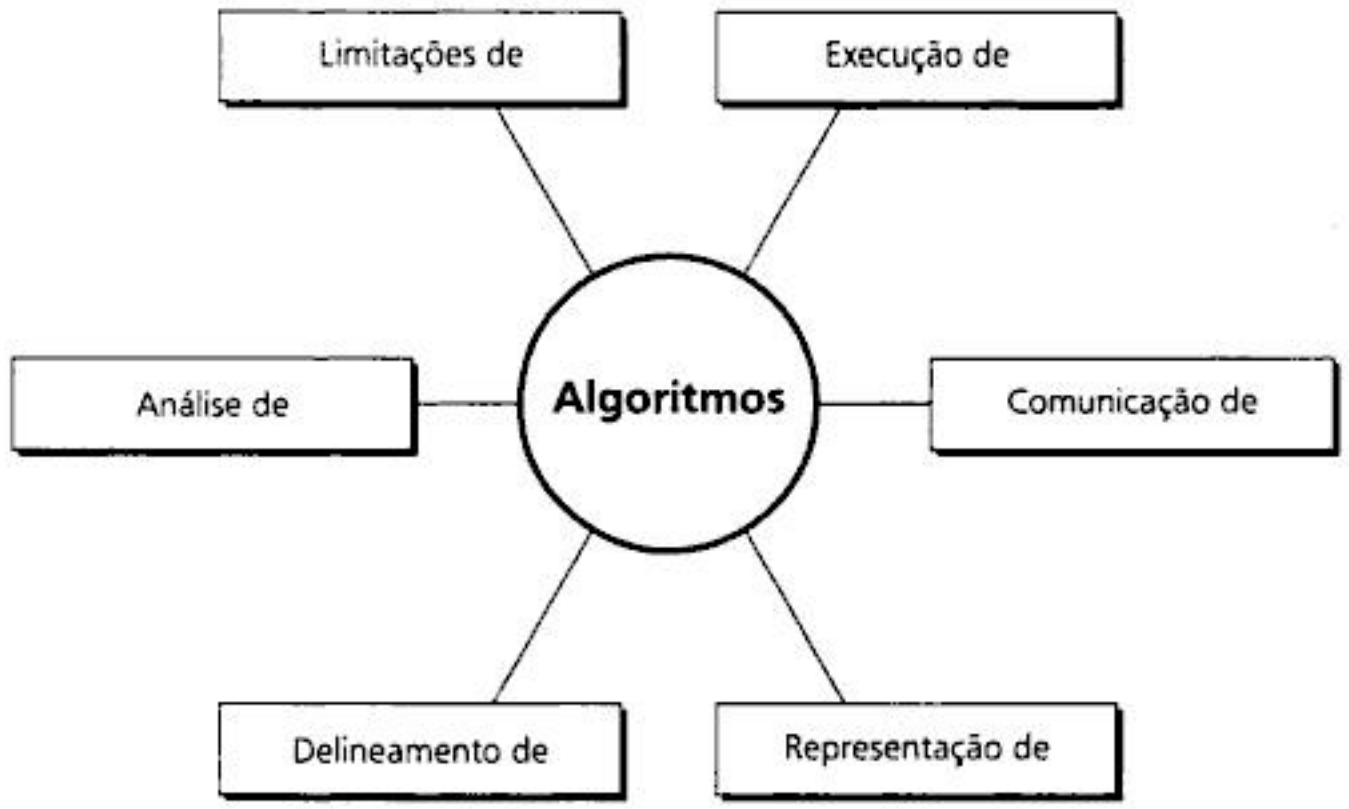



**Figura 0.5** O papel central dos algoritmos na Ciência da Computação.

de e os detalhes de sua composição interna. É a abstração que nos permite ignorar os detalhes internos de um dispositivo complexo, como um computador, um automóvel ou um forno de microondas, e usá-lo como uma unidade compreensível. Além disso, é por meio da abstração que esses sistemas complexos são projetados. Um automóvel, por exemplo, é projetado de maneira hierárquica: no nível superior, é visto como uma coleção de grandes componentes, como o motor, o sistema de suspensão e direção, sem se levar em conta os detalhes internos de cada um desses sistemas. Cada componente, por sua vez, é construído a partir de outros componentes.

O *hardware* de um computador pessoal típico possui uma decomposição similar (Figura 0.6). No nível superior, ele pode ser visto como uma coleção de componentes, como o computador propriamente dito, um teclado, um monitor, um *mouse* e uma impressora. Se focalizarmos a impressora, veremos que ela consiste em componentes menores, como um mecanismo de alimentação de papel, um sistema de controle lógico e um sistema de injeção de tinta. Além disso, o mecanismo de alimentação de papel consiste em unidades ainda menores, como uma bandeja para o papel, um motor elétrico e um caminho para a alimentação.

Vemos então que a abstração nos permite construir, analisar e manejar sistemas grandes e complexos de computação que nos confundiriam se fossem vistos em sua totalidade em nível de detalhes. Com efeito, aplicando a abstração, abordamos tais sistemas em vários níveis de detalhe. Em cada um, vemos o sistema em termos de componentes chamados **ferramentas abstratas**, cuja composição interna podemos ignorar. Isso nos permite concentrar a nossa atenção no modo como cada componente interage com os outros componentes de mesmo nível e a coleção como um todo forma um componente no nível superior. Assim, compreendemos a parte do

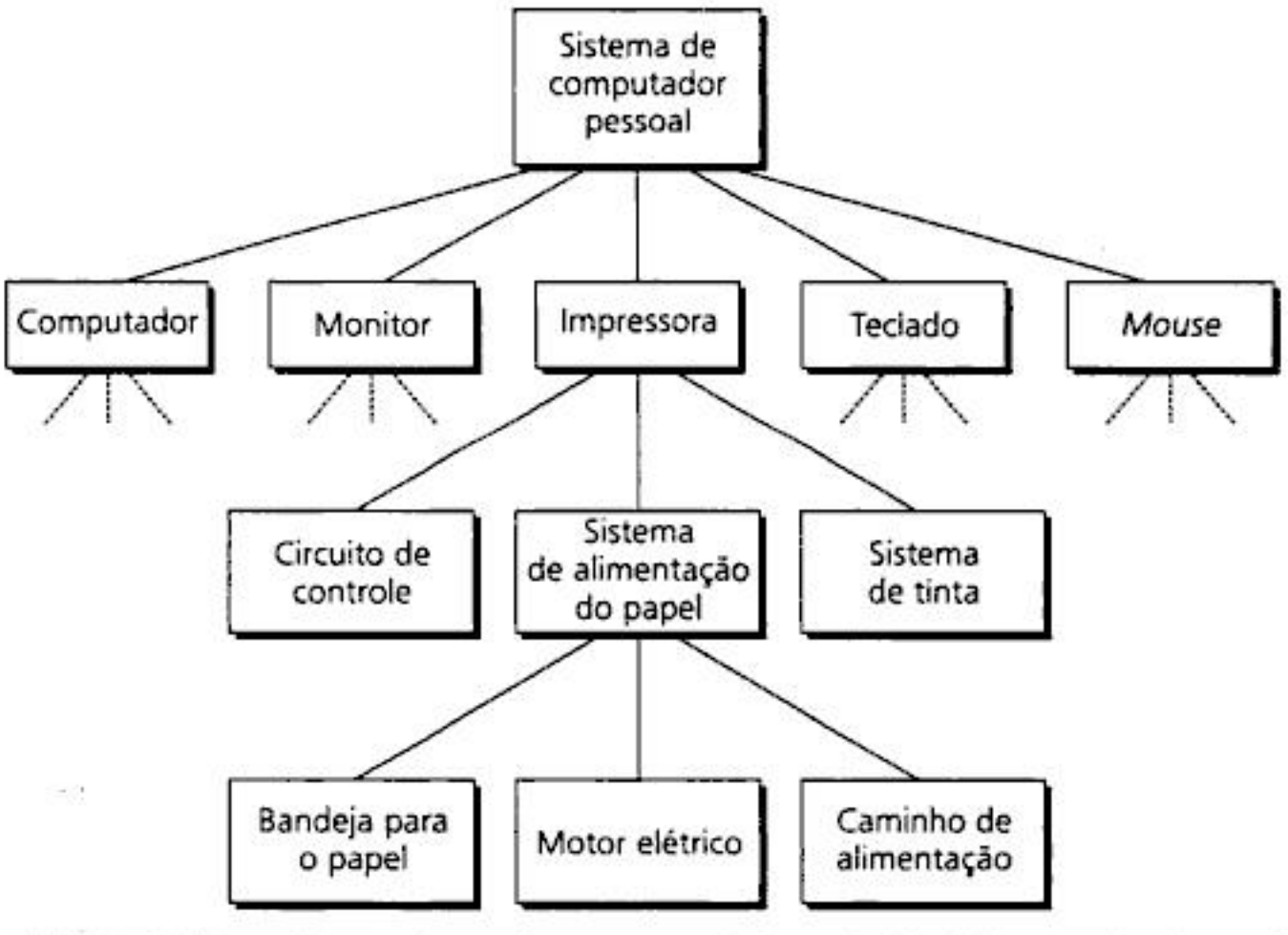



**Figura 0.6** A hierarquia da abstração no *hardware* de um computador pessoal típico.

sistema que é relevante à tarefa desejada, sem nos perdermos em um mar de detalhes. Essa é a beleza da abstração.

Devemos notar que a abstração não se limita à ciência e à tecnologia. Ela é uma técnica de simplificação importante com a qual a nossa sociedade criou um estilo de vida que de outra forma seria impossível. Poucos são os que entendem como as várias conveniências do dia-a-dia são implementadas. Ingerimos alimentos e nos vestimos com roupas que sozinhos não poderíamos produzir. Usamos dispositivos elétricos sem entender a tecnologia subjacente. Usamos os serviços de outros sem conhecer os detalhes de suas profissões. Com cada novo avanço, uma pequena parte da sociedade escolhe se especializar em sua implementação enquanto o restante aprende a usar os resultados como ferramentas abstratas. Dessa maneira, o estoque de ferramentas abstratas da sociedade se expande e crescem as condições da sociedade para avançar ainda mais.

Este texto, por sua vez, aplica a abstração para obter capítulos (e mesmo seções dentro dos capítulos) que são surpreendentemente independentes. Os tópicos cobertos nos capítulos iniciais também servem como ferramentas abstratas nos capítulos posteriores. Assim, um entendimento detalhado dos capítulos iniciais não é necessário para compreender os demais. Você pode, por exemplo, iniciar o seu estudo lendo o Capítulo 10 (Inteligência Artificial) e mesmo assim compreender a maior parte do material. (Simplesmente use o índice para encontrar o significado de qualquer termo técnico que lhe tenha escapado.) No mesmo espírito da Figura 0.6, a Figura 0.7 apresenta a composição hierárquica deste texto, que consiste em quatro partes que podem ser estudadas de modo independente. Dentro

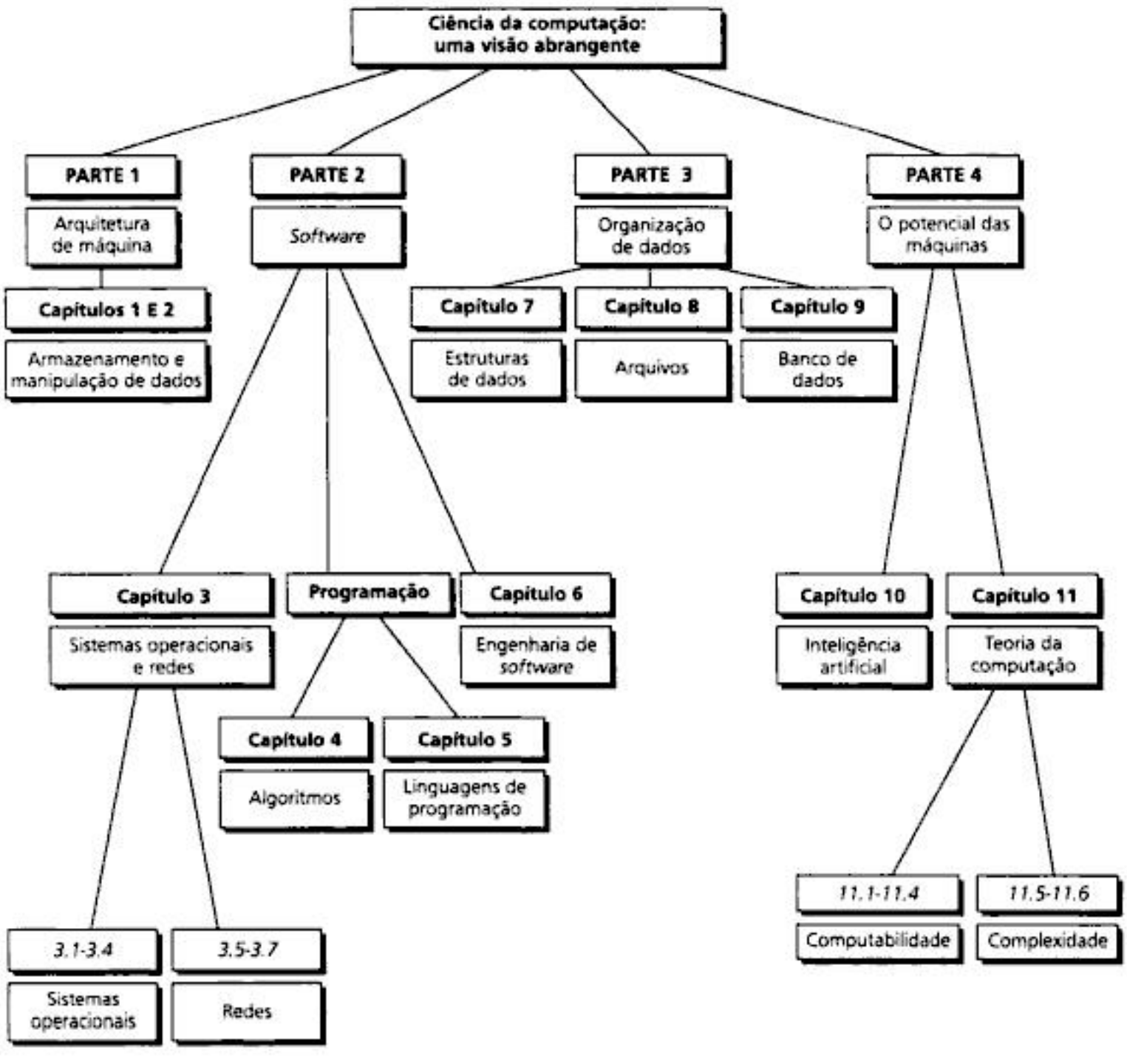



**Figura 0.7** Visão do texto como uma hierarquia de ferramentas abstratas.

de cada parte, por sua vez, estão capítulos e agrupamentos de seções a serem estudados como unidades ainda menores de abstração. Esta decomposição não é mero artefato do texto — ela reflete a estrutura da ciência. De fato, a Ciência da Computação consiste em numerosos assuntos separados, ainda que relacionados.

## 0.5 Repercussões sociais

Os avanços na ciência e na tecnologia estão obscurecendo muitas distinções nas quais a nossa sociedade se baseou para as tomadas de decisões no passado e estão mesmo desafiando muitos princípios da sociedade. Qual é a diferença entre a presença de um comportamento inteligente e a inteligência propriamente dita? Quando a vida começa e quando termina? Qual é a diferença entre uma planta e um animal? Existe vida em outros sistemas estelares? Tais perguntas desafiam o indivíduo a reavaliar seus conceitos e, muitas vezes, a reconstruir os fundamentos de suas convicções.

A Ciência da Computação suscita essas questões em inúmeras situações. No Direito, questiona-se até onde vai o direito de posse de um *software* e os deveres que devem acompanhar tais direitos. Do ponto de vista ético, os indivíduos enfrentam numerosas opções, que desafiam conceitos antigos e tradicionais nos quais se baseia a sua conduta. No governo, questiona-se até que ponto devem ser regulamentadas a tecnologia e as aplicações da computação. Na sociedade como um todo, se pergunta se as aplicações dos computadores representam novas liberdades ou novos controles.

Solucionar esses dilemas de uma maneira racional requer um conhecimento dos elementos relevantes da ciência e da tecnologia. Por exemplo, para a sociedade tomar decisões racionais quanto ao armazenamento e à disposição de lixo nuclear, os seus membros precisam conhecer os efeitos da radiação, saber como proteger-se dos seus riscos e ter uma perspectiva realista do período de permanência do risco de radiação. De modo similar, para determinar se governos ou companhias têm ou não o direito de desenvolver grandes bancos de dados integrados que contenham informações sobre seus cidadãos ou clientes, os membros dessa sociedade devem possuir um conhecimento básico das capacidades, limitações e ramificações da tecnologia de banco de dados.

Este texto proporciona um embasamento fundamental, a partir do qual você pode abordar esses tópicos de uma maneira informada. Algumas seções são dedicadas aos aspectos sociais, éticos e legais. Por exemplo, discutimos matérias de privacidade em relação à Internet e à tecnologia de banco de dados e tópicos como direito de propriedade em relação ao desenvolvimento de *software*. Embora não sejam parte da Ciência da Computação, esses tópicos são importantes para os leigos e para quem pretende fazer carreira em campos relacionados à computação.

É claro que o conhecimento factual sozinho não necessariamente provê soluções às questões geradas pelos avanços recentes na Ciência da Computação. Raramente existe uma única resposta correta, e muitas soluções válidas são compromissos entre visões antagônicas. Assim, para encontrá-las, normalmente é necessária habilidade de ouvir, reconhecer outros pontos de vista, promover debates racionais e alterar a sua própria opinião quando novos conhecimentos são adquiridos. Com isso em mente, cada capítulo deste texto é finalizado com uma coleção de questões chamada Questões Sociais. Elas não são necessariamente questões para serem respondidas e sim, consideradas. Em muitos casos, uma resposta que inicialmente parece óbvia deixará de satisfazê-lo quando você explorar as alternativas. Encerramos esta Introdução com uma coleção dessas questões relacionadas a tópicos da computação em geral.

### *Questões sociais*

As seguintes questões procuram auxiliar o leitor no entendimento de alguns assuntos éticos, sociais e legais no campo da computação. O objetivo não é meramente que sejam dadas respostas a tais questões. O leitor também deve justificá-las e verificar se as justificativas apresentadas preservam sua consistência de uma questão para outra.

1. Em geral, aceita-se a premissa de que a nossa sociedade é *diferente* do que teria sido se a revolução da computação não tivesse ocorrido. Nossa sociedade é *melhor* do que teria sido sem essa revolução? É *pior?* Sua resposta seria diferente se sua posição fosse outra nessa sociedade?

2. É aceitável participar da atual sociedade tecnológica sem fazer um esforço para conhecer os fundamentos dessa tecnologia? Por exemplo, os membros de uma democracia cujos votos geralmente determinam como a tecnologia será conduzida e utilizada têm a obrigação de conhecer essa tecnologia? A sua resposta depende de qual tecnologia está sendo considerada? Por exemplo, sua resposta será a mesma para a tecnologia nuclear e para a tecnologia da computação?

3. Usando dinheiro vivo, os indivíduos sempre tiveram a opção de administrar os seus negócios financeiros sem taxa de serviço. No entanto, como a nossa economia está cada vez mais automatizada, as instituições financeiras cobram taxas pelo acesso aos seus sistemas automáticos. Há momentos em que essa cobrança restringe, injustamente, o acesso de um indivíduo à economia? Por exemplo, suponha que um empregador remunere seus funcionários exclusivamente com cheques, e que todas as instituições financeiras cobrem uma taxa de serviço para cada cheque compensado ou depositado. Os funcionários estão sendo tratados injustamente? O que aconteceria se o empregador insistisse em pagar somente por meio de depósito direto?

4. No contexto da televisão interativa, até que ponto uma concessionária terá direito de extrair de crianças (talvez utilizando jogos interativos) informações sobre sua família? Por exemplo, deve ser permitido à companhia obter de uma criança informação sobre o perfil de consumo de seus pais? E quanto à informação relativa à própria criança?

5. Até que ponto um governo pode regulamentar a tecnologia da computação e suas aplicações? Por exemplo, consideremos os assuntos mencionados nas Questões 3 e 4. O que justificaria uma intervenção governamental?

6. Até que ponto as nossas decisões, relativas à tecnologia em geral e à tecnologia da computação em particular, afetarão as futuras gerações?

7. À medida que a tecnologia avança, nosso sistema educacional vai sendo constantemente desafiado a reconsiderar o nível de abstração em que são apresentados os diversos assuntos estudados. Questiona-se muitas vezes se uma certa habilidade do aluno continua sendo necessária, ou se deveria ser permitido aos estudantes apoiarem-se em uma ferramenta abstrata. Não se ensina mais os estudantes de trigonometria a encontrar os valores das funções trigonométricas com o uso de tabelas. Em vez disso, eles empregam calculadoras eletrônicas como ferramentas abstratas para encontrar esses valores. Alguns argumentam que a divisão de números longos também deveria ceder lugar à abstração. Que outros assuntos apresentam controvérsias semelhantes? Algum dia, o uso da tecnologia de vídeo eliminará a necessidade da leitura? Verificadores automáticos de ortografia eliminarão a necessidade de habilidades ortográficas?

8. Pressuponha que o conceito de biblioteca pública seja fortemente baseado na premissa de que todos cidadãos em uma democracia devem ter acesso à informação. À medida que a maioria da informação vai sendo guardada e disseminada por meio da tecnologia da computação, o acesso a essa tecnologia passa a ser direito de cada indivíduo? Em caso afirmativo, as bibliotecas públicas devem se constituir no canal pelo qual esse acesso é garantido?

9. Que considerações éticas aparecem em uma sociedade que se baseia no uso de ferramentas abstratas? Existem casos em que é antiético usar um produto ou serviço sem conhecer como ele é oferecido? Ou sem entender os efeitos colaterais de seu uso?

**10.** À medida que nossa economia vai se tornando cada vez mais automatizada, fica mais fácil para o governo monitorar as atividades financeiras dos cidadãos. Isso é bom ou ruim?

**11.** Quais tecnologias imaginadas por George Orwell (Eric Blair) em seu romance *1984* tornaram-se realidade? Elas são usadas da maneira que Orwell previu?

**12.** Os pesquisadores no campo da ética desenvolveram várias teorias com o intuito de analisar as decisões éticas. Uma dessas teorias é que as decisões devem ser baseadas nas conseqüências. Por exemplo, dentro dessa esfera está o utilitarismo que considera "correta" a decisão que produz o maior benefício para o maior número de indivíduos. Uma outra teoria se baseia nos direitos em vez das conseqüências. Ela considera "correta" a decisão que respeite os direitos dos indivíduos. As respostas às questões precedentes indicam que você tende a seguir a ética baseada nos direitos ou nas conseqüências?

## *Leituras adicionais*

Dejoie, D., G. Fowler, and D. Paradice. *Ethical Issues in Information Systems*. Boston: Boyd & Fraser, 1991.
Edgar, S. L. *Morality and Machines* Sudbury, MA: Jones and Bartlett, 1997.
Forester, T., and P. Forrison. *Computer Ethics: Cautionary Tales and Ethical Dilemmas*. Cambridge, MA.: MIT Press, 1990.
Goldstine, J. J. *The Computer from Pascal to von Neumann*. Princeton: Princeton University Press, 1972.
Johnson, D. G. *Computer Ethics*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1994.
Johnson, D. G. *Ethical Issues in Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
Kizza, J. M. *Ethical and Social Issues in the Information Age*. New York, Springler-Verlag, 1998.
Mollenhoff, C. R. *Atanasoff: Forgotten Father of the Computer*. Ames: Iowa State University Press, 1988.
Neumann, P. G. *Computer Related Risks*, Reading, MA: Addison-Wesley, 1995.
Randell, B. *The Origins of Digital Computers*. New York: Springer-Verlag, 1973.
Rosenoer, J. *CyberLaw: The Law of the Internet*, New York, Springer-Verlag, 1997.
Shurkin, J. *Engines of the Mind*. New York: Norton, 1984.
Spinello, R. A. and H. T. Tavani, *Readings in CyberEthics*, Sudbury, MA: Jones and Bartlett, 2001.
Woolley, B. *The Bride of Science, Romance, Reason and Byron's Daughter*, New York: McGraw-Hill, 1999.

PARTE 1

# *Arquitetura de máquina*

A etapa principal no desenvolvimento de uma ciência é a construção de teorias, a serem confirmadas ou rejeitadas pela experimentação. Em alguns casos, essas teorias permanecem adormecidas por longos períodos, à espera do aparecimento de alguma tecnologia capaz de pô-las à prova. Em outros casos, as capacidades da tecnologia atual interferem nos assuntos da alçada da ciência.

O desenvolvimento da Ciência da Computação apresenta as duas características. Vimos que a ciência progrediu a partir de teorias originadas muito antes que a tecnologia pudesse produzir as máquinas preconizadas pelos antigos pesquisadores. Mesmo hoje, nosso crescente conhecimento dos processos algorítmicos está conduzindo ao projeto de novas máquinas, que desafiam os limites da tecnologia. Outros assuntos científicos, pelo contrário, têm suas raízes na aplicação da tecnologia moderna. A Ciência da Computação, em resumo, é a integração da pesquisa teórica com o desenvolvimento tecnológico, em uma harmoniosa simbiose com benefícios gerais.

Decorre que, para apreciar o papel dos vários assuntos da Ciência da Computação, é preciso compreender os fundamentos da atual tecnologia e a forma como ela influi no projeto e na implementação dos modernos computadores. Fornecer estes fundamentos é o propósito dos dois capítulos seguintes. No Capítulo 1, são discutidas as técnicas mediante as quais a informação é representada e armazenada nos computadores. No Capítulo 2, estudam-se os modos como as máquinas de hoje manipulam os seus dados.

CAPÍTULO 1

# Armazenamento de dados

Neste capítulo, são tratados assuntos relacionados à representação e ao armazenamento de dados no computador. Às vezes, consideraremos tópicos de tecnologia, já que esses assuntos freqüentemente se refletem nas características externas das máquinas modernas. Entretanto, a maior parte da nossa discussão tratará de tópicos que só se ligarão ao projeto de computadores muito depois que a atual tecnologia vier a ser substituída, por se ter tornado obsoleta.



**Os asteriscos indicam sugestões de seções consideradas opcionais.*

# 1.1 *Bits* e seu armazenamento

Os computadores representam a informação por meio de padrões de *bits*. Um ***bit*** (dígito binário) pode assumir os valores 0 e 1, os quais, por enquanto, consideraremos como meros símbolos, sem significado numérico. Na verdade, veremos que o significado de um *bit* varia de uma aplicação para outra. Algumas vezes, os padrões de *bits* são usados para representar valores numéricos e, em outras, para representar caracteres ou outros símbolos; também podem representar imagens ou sons. Armazenar um *bit* em um computador exige a presença de um dispositivo que possa assumir dois estados, como, por exemplo, um interruptor (ligado ou desligado), um relé (aberto ou fechado), ou um sinalizador de bandeira (erguida ou abaixada). Um dos estados representa 0 e o outro, 1. Nosso objetivo é estudar as formas como os *bits* são armazenados nos computadores modernos.

## Portas lógicas (*gates*) e *flip-flops*

Começamos introduzindo as operações AND (e), OR (ou) e XOR (ou exclusivo), conforme esquematizado na Figura 1.1. Estas operações são semelhantes às operações aritméticas de produto e de soma, por operarem sobre um par de valores (as entradas da operação) e produzirem um terceiro valor, o resultado (saída) da operação. Entretanto, note-se que os únicos dígitos manipulados pelas operações AND, OR e XOR são 0 e 1; estas operações atuam conceitualmente sobre os valores TRUE (verdadeiro) e FALSE (falso) — 1 para verdadeiro, 0 para falso. Operações que manipulam valores verdadeiro/falso são chamadas **operações booleanas**, em homenagem ao matemático George Boole (1815-1864).

A operação booleana AND reflete a verdade ou falsidade de uma expressão formada combinando duas expressões menores por meio da conjunção *and*. Estas expressões apresentam a forma genérica

*P* AND *Q*

onde *P* representa uma expressão e *Q*, outra. Por exemplo,

> Caco é um sapo AND Senhorita Piggy é uma atriz.

Os dados de entrada para a operação AND representam a verdade ou falsidade dos componentes da expressão composta e a saída, a verdade ou falsidade da expressão composta. Como uma expressão da forma *P* AND *Q* só é verdadeira quando os seus dois componentes são verdadeiros, concluímos que 1 AND 1 resulta em 1; as demais situações resultariam em 0, de acordo com a Figura 1.1.

De modo semelhante, a operação OR está baseada em expressões compostas da forma

*P* OR *Q*

onde, novamente, *P* representa uma expressão e *Q*, outra. Tais expressões são verdadeiras quando pelo menos um dos componentes é verdadeiro, o que concorda com o conceito da operação OR, descrito na Figura 1.1.

**A operação AND**

```
  0     0     1     1
AND 0 AND 1 AND 0 AND 1
  0     0     0     1
```

**A operação OR**

```
  0     0     1     1
OR  0 OR  1 OR  0 OR  1
  0     1     1     1
```

**A operação XOR**

```
  0     0     1     1
XOR 0 XOR 1 XOR 0 XOR 1
  0     1     1     0
```

**Figura 1.1** As operações booleanas AND, OR e XOR (ou exclusivo).

Não há uma conjunção no idioma inglês* que expresse o significado da operação XOR. O resultado, ou seja, a saída da operação XOR será 1 quando uma de suas entradas for verdadeira e a outra, falsa. Por exemplo, uma declaração da forma *P* XOR *Q* significa "*P* ou *Q*, mas não ambos".

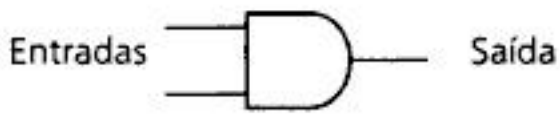
A operação NOT é outra operação booleana. Difere das operações AND, OR e XOR por apresentar uma única entrada. O valor da sua saída é o oposto da entrada: se o dado de entrada da operação NOT for verdadeiro, a saída será falsa, e vice-versa. Assim, se a entrada da operação NOT representar a veracidade** da afirmação

Fozzie é um urso,

então sua saída representará a veracidade** da afirmação oposta:

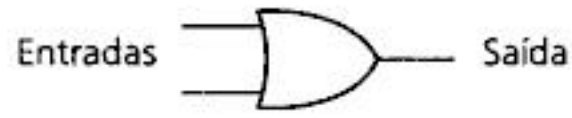
Fozzie não é um urso.

**AND**



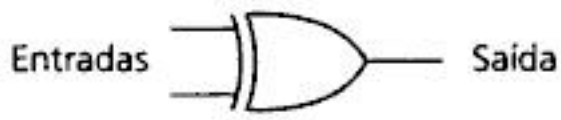

| Entradas | Saída |
|---|---|
| 0 0<br>0 1<br>1 0<br>1 1 | 0<br>0<br>0<br>1 |

**OR**



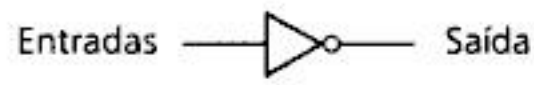

| Entradas | Saída |
|---|---|
| 0 0<br>0 1<br>1 0<br>1 1 | 0<br>1<br>1<br>1 |

**XOR**




| Entradas | Saída |
|---|---|
| 0 0<br>0 1<br>1 0<br>1 1 | 0<br>1<br>1<br>0 |

**NOT**




| Entradas | Saída |
|---|---|
| 0<br>1 | 1<br>0 |

**Figura 1.2** Representação gráfica das portas lógicas AND, OR, XOR e NOT, com os valores de suas entradas e saídas.

*N. de T. Em português, também não. Aqui cabe uma observação similar para a saída da operação NOT.

**N. de T. Neste ponto, o termo veracidade está sendo empregado no sentido de que, se a afirmação em questão for verdadeira, o valor da entrada da operação NOT será considerado TRUE; caso contrário, FALSE.

Um dispositivo que fornece a saída de uma operação booleana, a partir de suas entradas, é denominado **porta lógica**. As portas lógicas podem ser implementadas por meio de diversas tecnologias, tais como engrenagens, relés e dispositivos óticos. As portas lógicas dos computadores modernos geralmente são constituídas de pequenos circuitos eletrônicos, em que os dígitos 0 e 1 são representados por níveis de tensão elétrica. No presente estudo, todavia, não haverá necessidade de nos aprofundarmos nesses pormenores. Para os nossos propósitos, basta representar as portas lógicas na forma simbólica, conforme mostrado na Figura 1.2. Note-se que as portas lógicas das operações AND, OR, XOR e NOT são representadas por meio de diagramas diferentes, com os dados de entrada posicionados em um dos lados do diagrama, enquanto os de saída são representados no lado oposto.

Tais portas lógicas permitem a realização de blocos, com os quais são construídos os computadores. Um passo importante nesta direção está descrito no circuito representado na Figura 1.3. Este é um exemplo de uma série de circuitos do tipo *flip-flop*. Um ***flip-flop*** é um circuito cuja saída apresenta um dos dois valores binários, permanecendo assim até que um pulso temporário em sua entrada, proveniente de outro circuito, venha a forçá-lo a modificar sua saída. Em outras palavras, o valor da saída alternará entre dois valores, de acordo com a ocorrência de estímulos externos. Enquanto as duas entradas do circuito da Figura 1.3 permanecerem com seus valores em 0, a sua saída (seja ela 0 ou 1) não se alterará. Todavia, introduzindo-se temporariamente o valor 1 na entrada superior, a saída será levada a assumir o valor 1. Se, no entanto, for introduzido temporariamente o valor 1 na entrada inferior, a saída será levada a assumir o valor 0.

Analisemos com maior cuidado estes resultados. Sem conhecer a saída do circuito descrito na Figura 1.3, suponhamos que o valor nele introduzido pela sua entrada superior seja 1, enquanto o da entrada inferior permanece com o valor 0 (Figura 1.4a). Isto levará o circuito a atribuir à saída da porta lógica OR o valor 1, independentemente do valor presente em sua outra entrada. Por outro lado, os valores das duas entradas da porta lógica AND agora serão iguais a 1, já que o valor da outra entrada desta porta já se encontra em 1 (valor este obtido passando-se o dado de entrada inferior do *flip-flop* através da porta lógica NOT). O valor da saída da porta lógica AND agora será igual a 1, o que significa que o valor da segunda entrada da porta lógica OR agora será igual a 1 (Figura 1.4b). Isto garante que o valor da saída da porta lógica OR permanecerá em 1, mesmo quando o valor da entrada superior do *flip-flop* retornar para 0 (Figura 1.4c). Em suma, o valor da saída do *flip-flop* passará a ser 1, permanecendo assim mesmo depois que o valor da entrada superior retornar a 0.

De forma semelhante, colocando-se temporariamente o valor 1 na entrada inferior, o valor da saída do *flip-flop* será forçado para 0, permanecendo assim mesmo que o valor da entrada inferior volte para 0.

A importância do *flip-flop*, do nosso ponto de vista, é que ele se mostra ideal para o armazenamento de um *bit* no interior de um computador. O valor armazenado é o valor de saída do *flip-flop*. Outros circuitos podem facilmente ajustar este valor enviando pulsos para as entradas do *flip-flop*, e outros ainda podem responder ao valor armazenado usando a saída do *flip-flop* como suas entradas.

Existem, é claro, outras maneiras de se construir um *flip-flop*. Uma alternativa é mostrada na Figura 1.5. Se você experimentar com esse circuito, verá que embora apresente uma estrutura interna diferente, suas propriedades externas são as mesmas daquelas da Figura 1.3. Isto nos leva ao primeiro exemplo do papel das ferramentas abstratas. Quando projeta um *flip-flop*, um engenheiro considera as maneiras alternativas nas quais ele pode ser construído usando portas lógicas como módulos. Então, uma vez projetados os *flip-flops* e outros circuitos lógicos, o engenheiro pode usá-los como módulos para construir circuitos mais complexos. Assim, o projeto dos circuitos de um computador se faz em uma estrutura hierárquica, onde cada nível usa os componentes do nível abaixo como ferramentas abstratas.

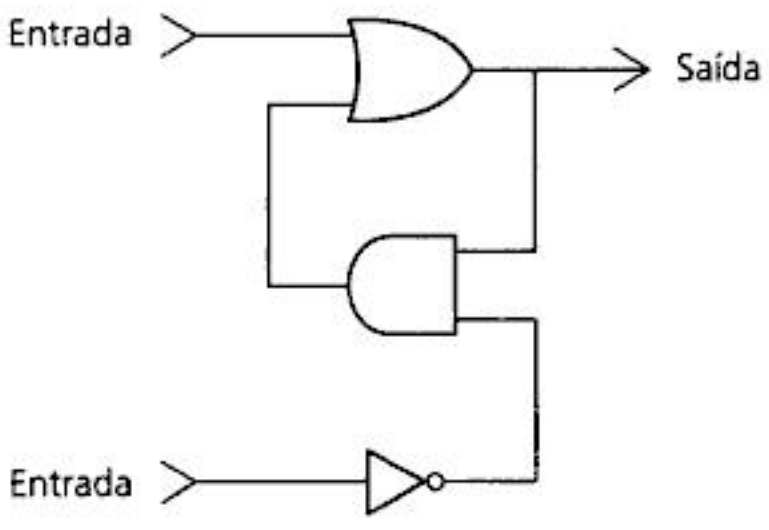



**Figura 1.3** Um circuito simples de *flip-flop*.

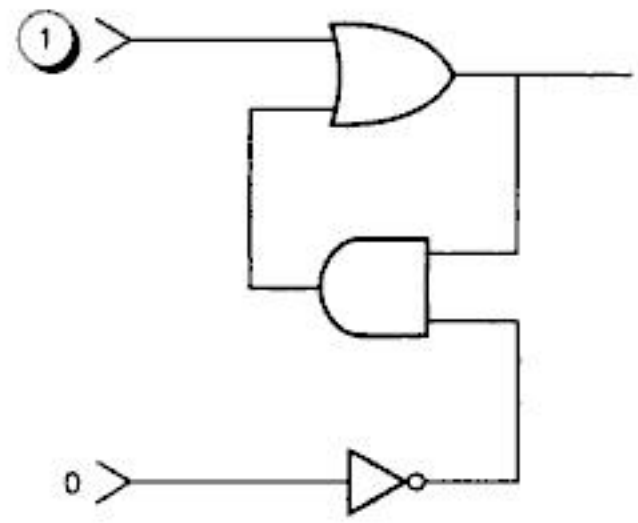
## Outras técnicas de armazenamento

Na década de 1960, era comum os computadores conterem pequeninos anéis ou argolas feitos de material magnético chamados **núcleos**, atravessados por fios elétricos muito finos. Passando uma corrente elétrica pelos fios, cada núcleo poderia ser magnetizado em uma de duas polaridades possíveis. Posteriormente, a polaridade desse campo magnético poderia ser consultada observando-se seu efeito sobre uma corrente elétrica forçada pelos fios que atravessavam o núcleo. Deste modo, os núcleos magnéticos realizavam uma forma de armazenamento de *bits* — o 1 era representado por um campo magnético em uma direção e o 0, por um campo magnético na outra. Tais sistemas são obsoletos hoje, devido ao seu tamanho e ao seu consumo de energia.

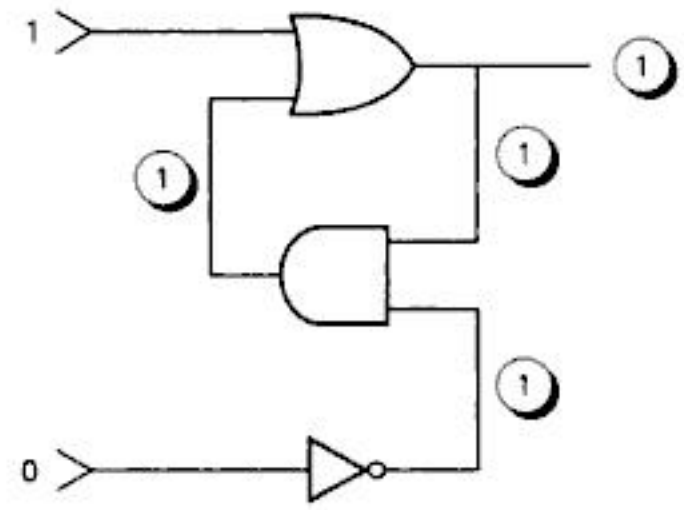
Um método mais recente para armazenar um *bit* é o capacitor, que consiste em duas pequenas placas metálicas posicionadas paralelamente mantendo uma pequena distância de separação. Se uma fonte de tensão for conectada às placas — positivo em uma placa e negativo na outra — as cargas da fonte se distribuirão nas placas. Então, quando a fonte de tensão for removida, essas cargas ficarão nas placas. Se estas forem conectadas mais tarde, uma corrente elétrica fluirá pela conexão e as cargas serão neutralizadas. Assim, um capacitor está em um de dois possíveis estados: carregado ou descarregado, um deles pode ser usado para representar o 0 e o outro, para representar o 1. A tecnologia atual é capaz de construir milhões de pequenos capacitores, bem como os seus circuitos, tudo em uma única pastilha (chamada ***chip***). Assim, os capacitores vêm se tornando uma tecnologia popular para armazenamento de *bits* dentro das máquinas.

*Flip-flops*, núcleos e capacitores são exemplos de sistemas de armazenamento com diferentes graus de volatilidade. Um núcleo retém seu campo magnético após o desligamento da máquina. Um *flip-flop* perde o dado armazenado quando a fonte de energia é desligada. Por sua vez, as cargas nesses pequenos capacitores são tão frágeis que tendem a se dissipar neles próprios, mesmo enquanto a máquina está funcionando. Assim, a carga em um capacitor deve ser restaurada regularmente por um circuito conhecido como refresco (*refresh*). Considerando essa volatilidade, a memória do computador (Seção 1.2) construída com essa tecnologia, freqüentemente é chamada de **memória dinâmica**.

**a.** 1 é forçado na entrada superior.




**b.** Isto faz com que a saída da porta lógica OR apresente o símbolo 1 e, assim, a saída da porta lógica AND passe a ser 1.




**c.** O 1 proveniente da porta lógica AND evita que a porta lógica OR altere sua saída depois que a entrada superior retornar ao valor 0.

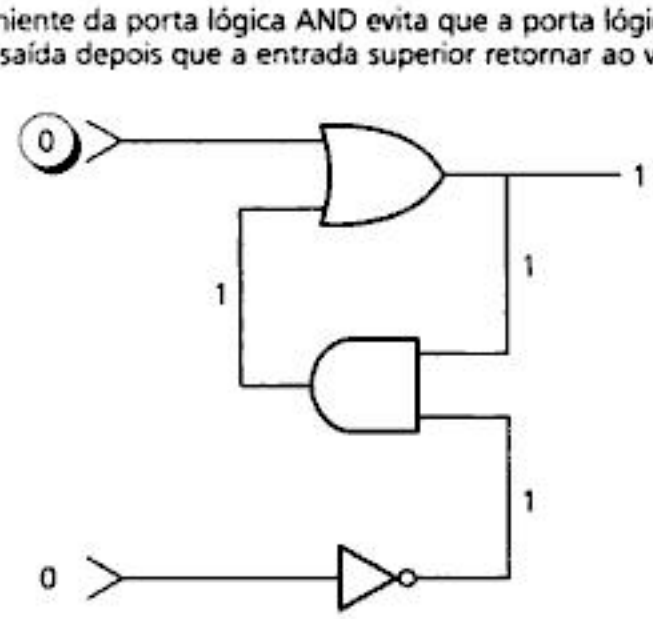



**Figura 1.4** Como forçar o valor 1 na saída do *flip-flop*.

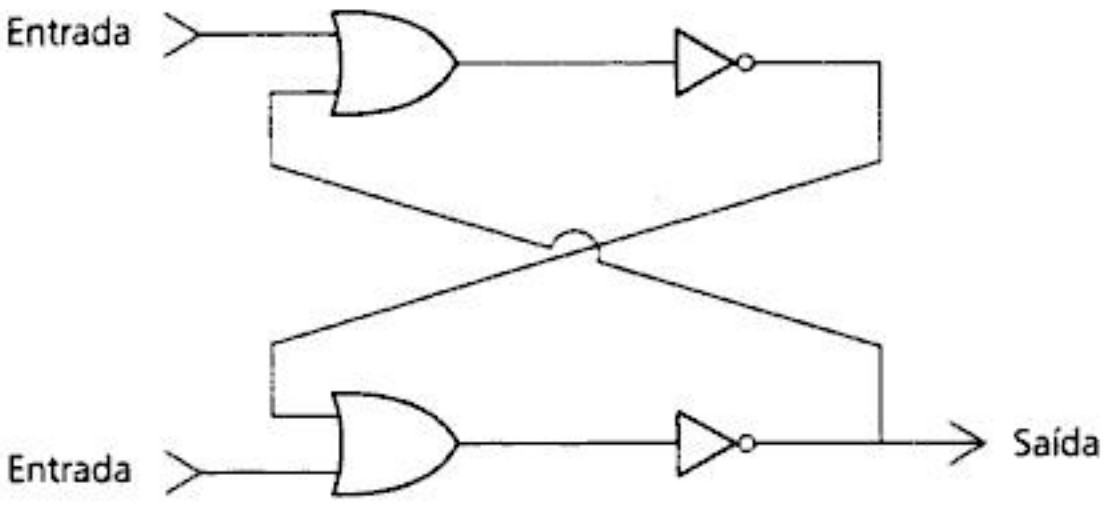



**Figura 1.5** Outra forma de construção de um *flip-flop*.

| Padrão de *bits* | Representação hexadecimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

**Figura 1.6** O sistema de código hexadecimal.

## A notação hexadecimal

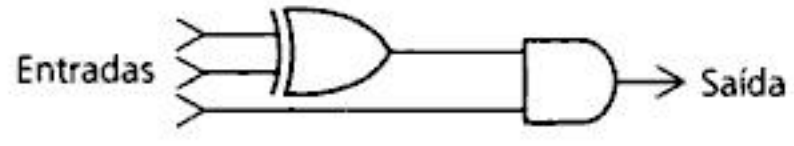
Quando consideramos as atividades internas de um computador, devemos lidar com cadeias de *bits*, algumas das quais podem ser muito longas. Essa cadeia é freqüentemente chamada corrente (***stream***). Infelizmente, a mente humana tem dificuldades para gerenciar esse nível de detalhe. A simples transcrição do padrão 101101010011 se mostra não apenas tediosa, como muito propensa a erros. Por essa razão, para simplificar a representação de padrões de *bits*, normalmente utilizamos uma notação mais compacta, conhecida como *notação hexadecimal*. Esta notação aproveita o fato de que os padrões de *bits* dentro de um computador apresentam sempre comprimentos múltiplos de quatro. Em particular, a notação hexadecimal utiliza um único símbolo para representar quatro *bits*, ou seja, uma seqüência de doze *bits* pode ser denotada com apenas três símbolos hexadecimais.

A Figura 1.6 apresenta o sistema de codificação hexadecimal. A coluna da esquerda mostra todos os possíveis padrões de *bits* de comprimento quatro, enquanto a da direita apresenta o símbolo utilizado na notação hexadecimal para representar o padrão de *bits* correspondente. Nesta notação, o padrão 10110101 é representado como B5. Isto é obtido dividindo o padrão de *bits* em subcadeias de comprimento quatro e novamente representando cada cadeia pela notação hexadecimal equivalente. Assim, 1011 é representado por B e 0101, por 5. Desta maneira, o padrão 1010010011001000, de 16 *bits*, pode ser reduzido à forma A4C8, bem mais confortável.

Utilizaremos extensivamente a notação hexadecimal no próximo capítulo, no qual se poderá constatar a sua eficácia.

### *QUESTÕES/EXERCÍCIOS*

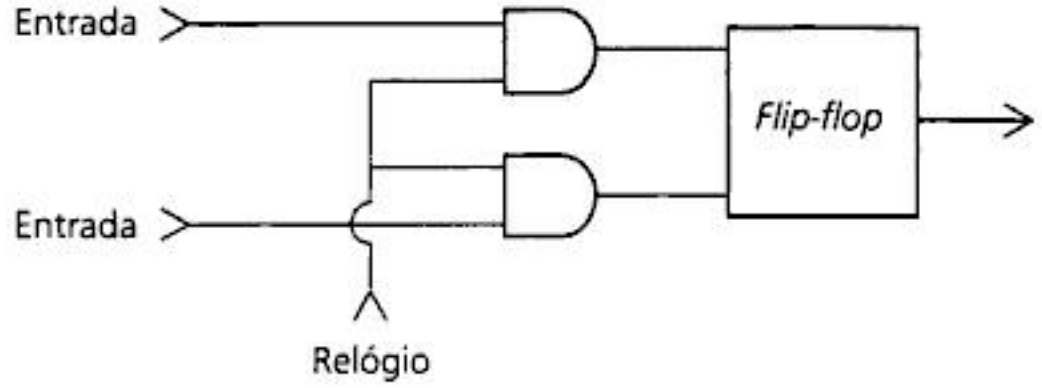
1. Que configurações de *bits* de entrada devem ser fornecidas ao circuito abaixo para que seja produzida uma saída 1?



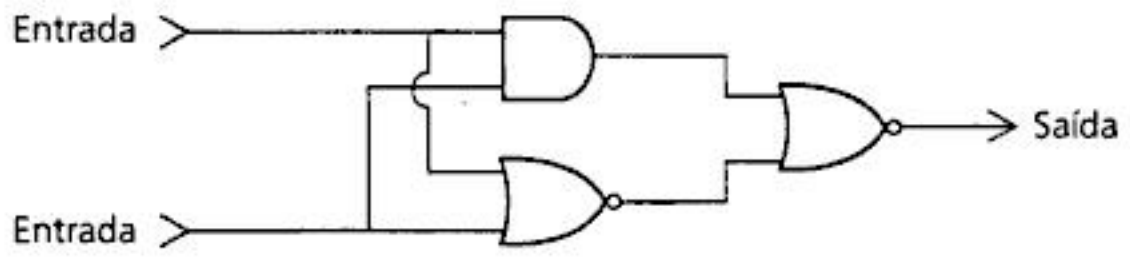

2. No texto, afirmamos que a introdução do 1 na entrada inferior do *flip-flop* da Figura 1.3 (mantendo fixo o 0 para a entrada superior) forçará a saída do *flip-flop* a ser 0. Descreva a sucessão de eventos que ocorreram neste caso, dentro do *flip-flop*.

**3.** Admitindo que as duas entradas do *flip-flop* da Figura 1.5 sejam 0, descreva a sucessão de eventos que ocorrerão quando a entrada superior for temporariamente igualada a 1.

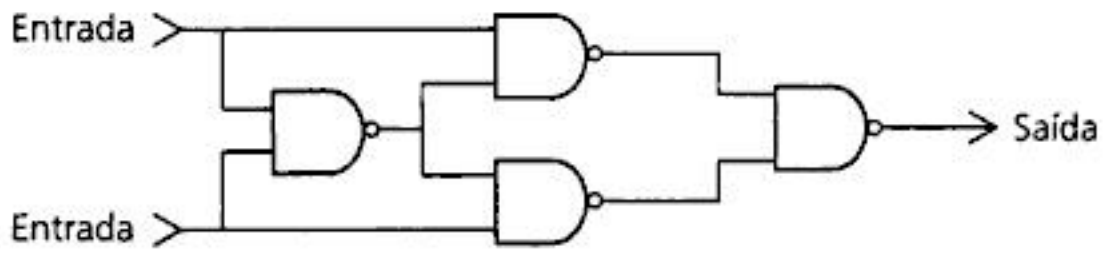
**4.** Muitas vezes é necessário coordenar atividades em várias partes de um circuito. Isso é feito acoplando um sinal pulsante (chamado *relógio)* às partes do circuito que requerem coordenação. Uma vez que o relógio alterna entre os valores 0 e 1, ele ativa os vários componentes do circuito. A seguir, está um exemplo de uma parte do circuito que envolve o *flip-flop* mostrado na Figura 1.3. Para quais valores do relógio o *flip-flop* ficará isolado dos valores de entrada do circuito? Para quais valores ele responderá aos valores de entrada do circuito?




**5.** a. Se a saída de uma porta lógica OR passar por uma porta NOT, a combinação executará a operação booleana chamada NOR, cuja saída só será 1 quando as duas entradas forem 0. O símbolo para essa porta lógica é o mesmo da porta OR, exceto em que ele possui um círculo em sua saída. Abaixo, está um circuito que contém uma porta lógica AND e duas portas NOR. Que função booleana ele computa?




b. Se a saída de uma porta lógica AND passar por uma porta NOT, a combinação executará a operação booleana chamada NAND, cuja saída só será 0 quando as duas entradas forem 1. O símbolo para essa porta lógica é o mesmo da porta AND, exceto em que ele possui um círculo em sua saída. Abaixo, está um circuito que contém portas lógicas NAND. Que função booleana ele computa?




**6.** Utilize a notação hexadecimal para representar os seguintes padrões de *bits*:
a. 0110101011110010
b. 111010000101010100010111
c. 01001000

**7.** Que padrões de *bits* são representados pelos seguintes padrões hexadecimais?
a. 5FD97 b. 610A c. ABCD d. 0100

## 1.2 Memória principal

Com a finalidade de armazenar seus dados, um computador contém um conjunto grande de circuitos, cada qual capaz de armazenar um *bit*. A este conjunto denominamos **memória principal** da máquina.

### Organização da memória

Os circuitos de armazenamento da memória principal de um computador são organizados em unidades manipuláveis denominadas **células** (ou *palavras*), geralmente com um tamanho de oito *bits* cada uma. De fato, os conjuntos de *bits* de tamanho oito ficaram tão populares que o termo ***byte*** é amplamente utilizado, nos dias de hoje, para denotá-lo. Computadores pequenos, utilizados em dispositivos domésticos, como fornos de microondas, podem ter suas memórias principais medidas em centenas de células, enquanto computadores de grande porte, utilizados para armazenar e manipular grandes quantidades de dados, podem ter bilhões de células em suas memórias principais.

Embora dentro do computador não tenham sentido os conceitos de lado esquerdo ou direito, normalmente imaginamos os *bits* de uma posição de memória organizados em linha, cuja extremidade esquerda é chamada **extremidade de alta ordem** e a direita, **extremidade de baixa ordem**. O último *bit* da extremidade de alta ordem é conhecido como o *bit* de ordem mais elevada, ou ***bit* mais significativo** em referência ao fato de que se o conteúdo da célula fosse interpretado como a representação de um valor numérico, este *bit* seria o dígito mais significativo do número. Do mesmo modo, o *bit* da extremidade direita é conhecido como o *bit* de ordem mais baixa, ou ***bit* menos significativo**. Dessa forma, podemos representar o conteúdo de uma célula de memória de um *byte* conforme mostra a Figura 1.7.

Para distinguir, na memória principal de um computador, cada célula individual, esta é identificada por um nome único, denominado **endereço**. Esse sistema de endereçamento é similar e emprega a mesma terminologia da técnica de identificação de edificações urbanas, com o auxílio de endereços. No caso das células de memória, contudo, os endereços utilizados são totalmente numéricos. Para ser mais precisos, podemos visualizar as células de memória como que dispostas em uma fila única, cujos elementos são numerados em ordem crescente, a partir do endereço com valor zero. Tal sistema de endereçamento não apenas nos fornece um meio de identificar univocamente cada célula, mas também associa a elas uma ordem (Figura 1.8), o que nos permite utilizar expressões como "a próxima célula" ou "a célula anterior".

Uma conseqüência importante da ordenação simultânea das posições da memória principal e dos *bits* dentro de cada célula é que todo o conjunto de *bits* dentro da memória principal de uma máquina fica ordenado, essencialmente, em uma fila única longa. Assim, partes dessa longa fila podem ser utilizadas para armazenar padrões de *bits* que sejam mais longos do que o comprimento de uma única célula. Em particular, se a memória estiver dividida em células com tamanho de um *byte* cada, ainda assim será possível armazenar uma cadeia de 16 *bits*, simplesmente utilizando duas células consecutivas de memória para isso.

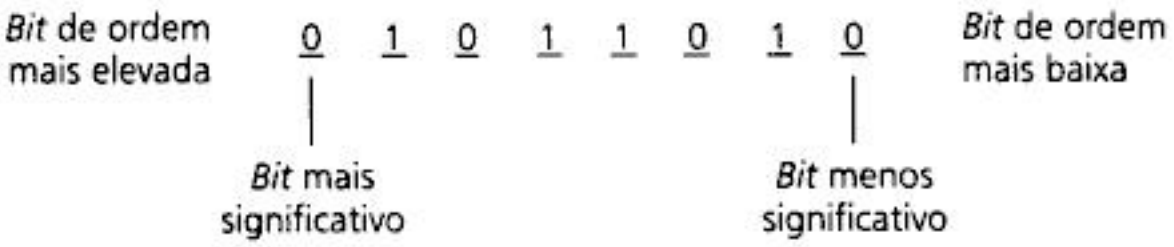



**Figura 1.7** A organização de uma célula de memória de um *byte* de comprimento.

Outra conseqüência de organizar a memória principal de uma máquina com pequenas células endereçadas é que cada uma pode ser individualmente referenciada. Por conseguinte, dados armazenados na memória principal de uma máquina podem ser processados em qualquer ordem, razão pela qual este tipo de memória é conhecido como **memória de acesso aleatório** (*random access memory* — RAM).

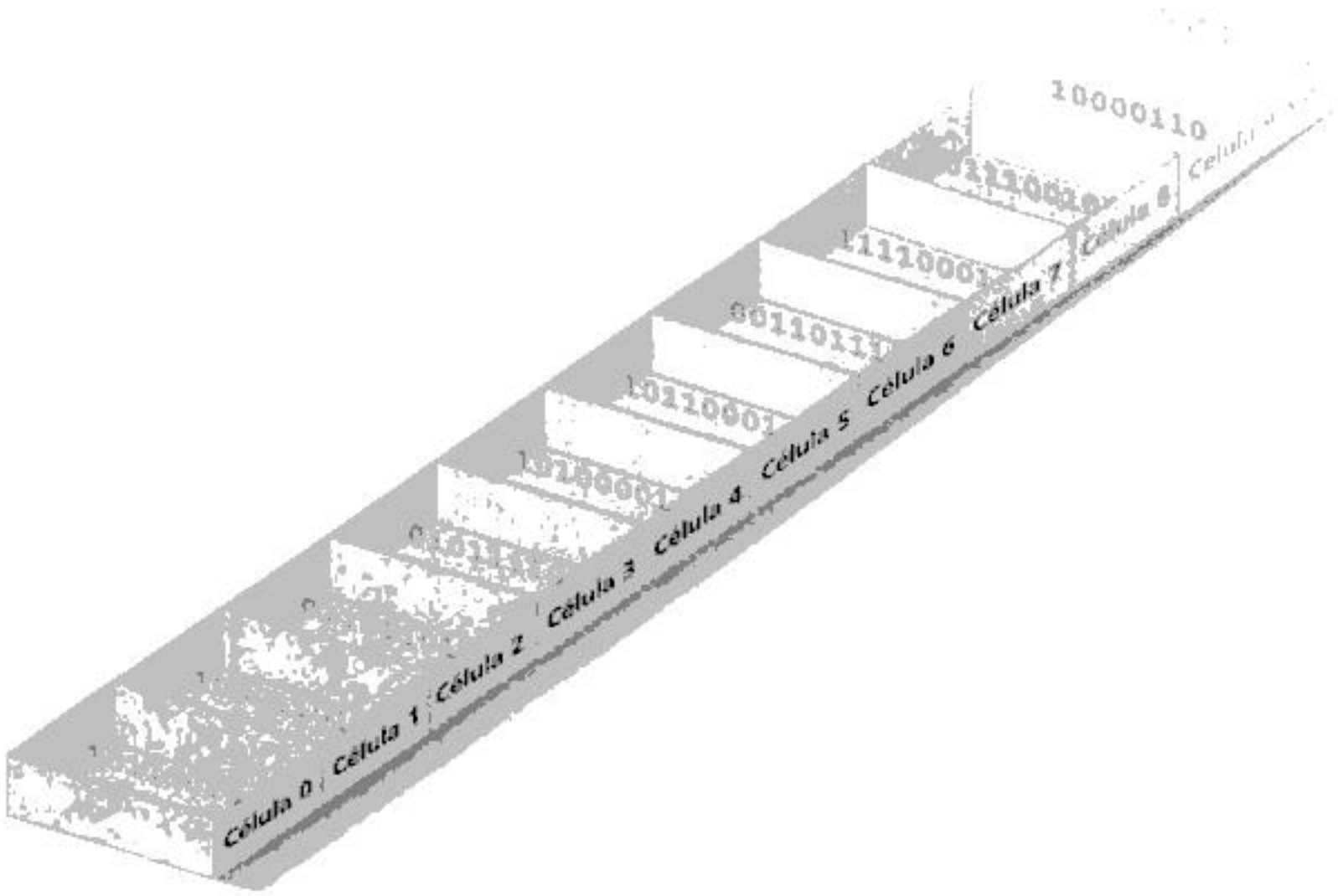



**Figura 1.8** Células de memória, ordenadas em ordem crescente de endereços.

Esse acesso aleatório a pequenas porções de dados contrasta significativamente com os sistemas de armazenamento em massa, discutidos na próxima seção, nos quais longas cadeias de *bits* devem ser manipuladas em bloco. Quando a RAM é construída com a tecnologia de memória dinâmica, freqüentemente é chamada de DRAM ("Dynamic RAM").

Para completar a memória principal de uma máquina, o sistema de circuitos que realmente armazena os *bits* é conectado com outro, encarregado de prover meios para que os demais circuitos possam armazenar dados nas células de memória e, posteriormente, reavê-los. Dessa maneira, outros circuitos ficam habilitados a consultar eletronicamente os dados da memória, recuperando, assim, o conteúdo de um determinado endereço (*read* — operação de leitura), ou então gravar informação na memória, ao solicitar que um padrão de *bits* desejado seja inserido na célula correspondente a um certo endereço (*write* — operação de gravação).

## Como medir a capacidade da memória

Como aprenderemos nos próximos capítulos, é conveniente projetar sistemas de memória principal nos quais o número total de células seja uma potência de dois. A propósito, o tamanho da memória dos primeiros computadores era comumente medido em unidades de 1024 (que é $2^{10}$) células. Como 1024 fica próximo de 1000, muitos na comunidade da computação adotaram o prefixo *kilo* em referência a essa unidade. Assim, o termo *kilobyte* (abreviado com KB) era usado para se referir a 1024 *bytes*, e dizia-se que uma máquina que tivesse 4096 células de memória tinha 4 KB de memória ($4096 = 4 \times 1024$). À medida que as memórias foram se tornando maiores, essa terminologia cresceu para incluir os prefixos

*mega*, para 1.048.576 (que é $2^{20}$), e *giga*, para 1.073.741.824 (que é $2^{30}$), e unidades como MB (*megabyte*) e GB (*gigabyte*) tornaram-se comuns.

Infelizmente, essa aplicação de prefixos representa um mau uso da terminologia, porque os prefixos já eram utilizados em outros campos para referir-se a unidades que são potências de dez. Por exemplo, quando medimos distância, *quilômetro* refere-se a 1.000 metros, e quando medimos rádio-freqüência, *megahertz* refere-se a 1.000.000 de hertz. Além disso, alguns fabricantes de equipamento de computação decidiram adotar o prefixo *mega* para referir-se a unidades de 1.024.000 (logo, um *megabyte* seria 1000 KB). Desnecessário dizer que essas discrepâncias têm levado à confusão e ao desentendimento ao longo dos anos.

Para esclarecer a matéria, uma proposta foi feita a fim de reservar os prefixos *kilo*, *mega* e *giga* para unidades que são potências de dez e introduzir os novos prefixos *kibi*. (abreviatura de *kilobinary*, abreviado com Ki), *mebi* (abreviatura de *megabinary*, abreviada com Mi), e *gibi* (abreviatura de *gigabinary*, abreviado com Gi), em referência às unidades correspondentes que são potências de dois. Nesse sistema, o termo *Kibibyte* se refere a 1024 *bytes* e *Kilobyte*, a 1000 *bytes*. Se esses prefixos se tornarão parte do vernáculo popular, só vendo. Por hora, o mau uso dos prefixos *kilo*, *mega* e *giga* continua impregnado na comunidade da computação, de modo que seguiremos essa tradição em nosso estudo. Contudo, os prefixos propostos *kibi*, *mebi* e *gibi* representam uma tentativa de resolver o problema crescente, e devemos interpretar cautelosamente no futuro termos como *kilobyte* e *megabyte*.

### QUESTÕES/EXERCÍCIOS

1. Se a posição de memória cujo endereço é 5 contém o valor 8, qual a diferença entre escrever o valor 5 na célula de número 6 e passar o conteúdo da célula de número 5 para a de número 6?

2. Suponha que se queira intercambiar os valores armazenados nas células de memória de endereços 2 e 3. O que há de errado na seqüência de passos descrita a seguir?
   *Passo 1.* Transfira o conteúdo da célula de número 2 para a de número 3.
   *Passo 2.* Transfira o conteúdo da célula de número 3 para a de número 2.
   Projete outra seqüência de passos que troque, corretamente, os conteúdos dessas duas células.

3. Quantos *bits* existem na memória de um computador com 4KB (mais precisamente KiB) de memória?

## 1.3 Armazenamento em massa

Devido à volatilidade e ao tamanho limitado da memória principal de um computador, a maioria das máquinas possui dispositivos de memória adicional chamados **sistemas de armazenamento em massa** — que incluem discos magnéticos, CDs e fitas magnéticas. As vantagens dos sistemas de armazenamento em massa em relação à memória principal incluem menor volatilidade, maior capacidade de armazenamento e, em muitos casos, a possibilidade de se retirar o meio físico no qual são gravados os dados da máquina com o propósito de arquivamento.

Os termos *on-line* e *off-line* são costumeiramente utilizados para referir-se a dispositivos que estejam, respectivamente, ligados diretamente ao computador, ou dele desconectados. ***On-line*** significa que o dispositivo ou a informação já está conectado(a) ao computador e prontamente disponível para a máquina, sem a necessidade de uma intervenção humana. ***Off-line***, pelo contrário, significa que a intervenção humana é necessária antes que seja possível ao computador fazer os acessos necessários à informação ou ao dispositivo — talvez porque o dispositivo precise ser ativado manualmente ou porque um meio físico que contenha a informação desejada deva ser fisicamente inserido em algum equipamento.

A maior desvantagem dos sistemas de armazenamento em massa é que geralmente requerem movimentação mecânica, portanto seus tempos de resposta são muito maiores quando comparados com os da memória principal da máquina, que executa todas as atividades eletronicamente.

## Discos magnéticos

Atualmente, a forma mais usual de armazenamento em massa é o disco magnético. Nele, os dados são armazenados em um fino disco giratório com revestimento de material magnético. Cabeçotes de leitura/gravação (*read/write*) ficam instalados nos dois lados do disco, de forma que, ao girá-lo, cada cabeçote percorra, na superfície superior ou inferior, uma trajetória circular, denominada **trilha** (*track*). Por meio do reposicionamento dos cabeçotes de leitura/gravação, é possível acessar as diversas trilhas concêntricas. Em muitos casos, um sistema de armazenamento em disco consiste em vários discos montados em um eixo giratório comum, um em cima do outro, com espaço suficiente para que o cabeçote de leitura/gravação se desloque entre os discos. Nesses casos, os cabeçotes se movem em uníssono. Cada vez que eles são reposicionados, um novo conjunto de trilhas — chamado **cilindro** — se torna acessível.

Considerando que cada trilha pode conter mais informação do que é necessário manipular de uma única vez, as trilhas são divididas em **setores**, nos quais a informação é registrada como cadeia contínua de *bits* (Figura 1.9). Cada trilha em um sistema de discos contém o mesmo número de setores e cada setor contém o mesmo número de *bits*. (Isso significa que os *bits* de um setor são armazenados mais compactados nas trilhas perto do centro do disco do que nas trilhas perto da borda.)

Assim, um sistema de armazenamento em disco consiste em muitos setores individuais, cada qual acessível na forma de uma cadeia independente de *bits*. O número de trilhas por superfície e o número de setores por trilha variam muito de um sistema de discos para outro. O tamanho dos setores tende a assumir valores não maiores que alguns KB; setores de 512 *bytes* ou 1024 *bytes* são comuns.

A localização física das trilhas e dos setores não constitui parte permanente da estrutura física de um disco. Ao contrário, eles são marcados magneticamente, como resultado de um processo denominado **formatação** (ou inicialização) do disco. Este processo em geral é efetuado pelo fabricante de discos, e resulta nos chamados discos formatados. A maioria dos sistemas computacionais tem a capacidade de executar essa tarefa. Assim, se a informação de formato em um disco for estragada, ele poderá ser reformatado, embora esse processo destrua toda informação que estava previamente gravada no disco.

A capacidade de um sistema de armazenamento em disco depende do número de superfícies magnéticas nele disponíveis e da densidade das suas trilhas e setores. Os sistemas de capacidade mais baixa consistem em um único disco de plástico conhecido como disquete ou, se for flexível, pelo título de disco flexível (*floppy disk*), de menor prestígio. (Os discos flexíveis atuais com diâmetro de 3½ polegadas são protegidos com capa de plástico rígido, não sendo, portanto, tão flexíveis quanto seus antigos primos com diâmetro de 5¼ polegadas e capa de papel.) Os disquetes podem ser facilmente inseridos na respectiva unidade de leitura/gravação e dela removidos, bem como facilmente armazenados. Conseqüentemente, representam um bom meio de armazenamento de informação *off-line*. O disquete usual de 3½ polegadas é capaz de armazenar 1,44 MB, mas existem disquetes com capacidades muito maiores. Um exemplo é o sistema de discos Zip da Iomega Corporation, com capacidade de centenas de MB em um único disquete rígido.

Os sistemas de disco de alta capacidade de armazenamento, capazes de

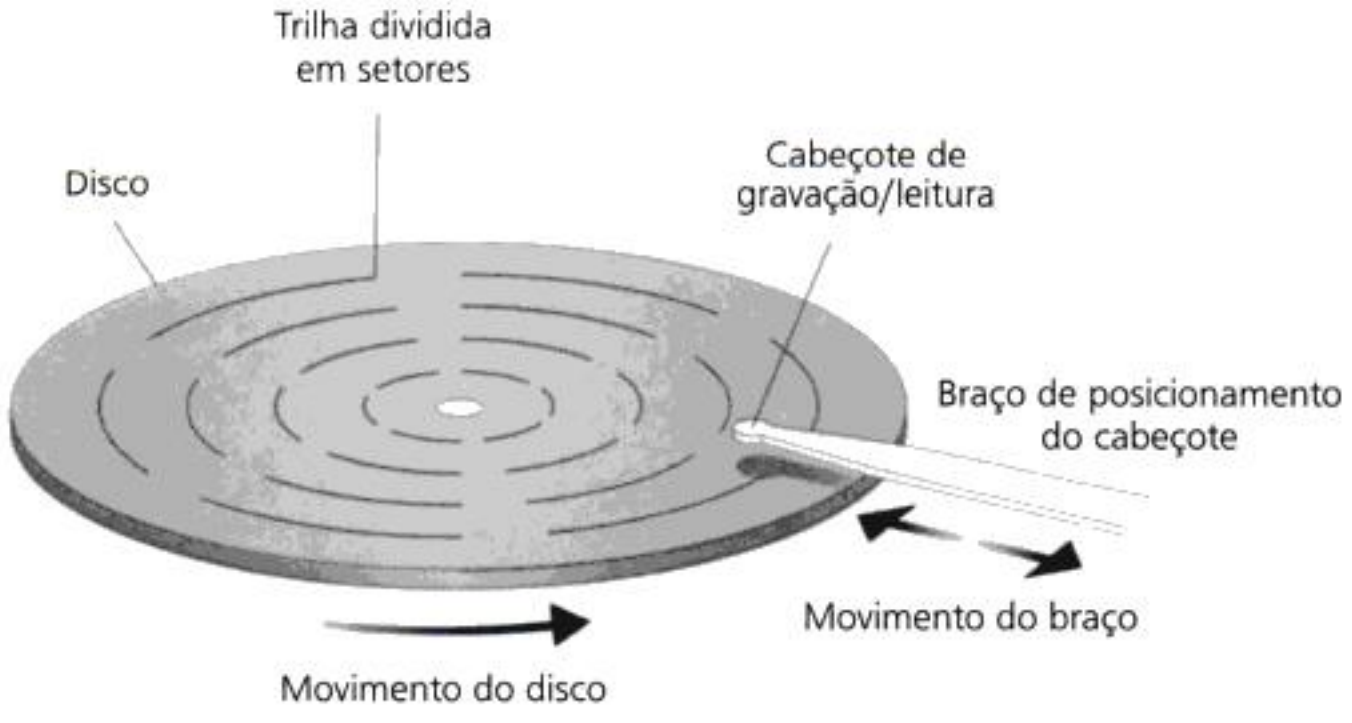



**Figura 1.9** Um sistema de armazenamento em disco.

comportar vários *gigabytes*, consistem em cinco a dez discos rígidos, montados em uma coluna comum. Pelo fato de tais discos serem feitos de material rígido, são conhecidos como sistemas de disco rígido (*hard disk*), em contraste com os flexíveis (*floppy disk*). Para permitir uma velocidade maior de rotação nesses sistemas, seu cabeçote de leitura/gravação não chega a ter contato físico com o disco, mas apenas se mantém flutuando pouco acima da superfície do disco em movimento. A distância entre o cabeçote e a superfície do disco é tão pequena que qualquer partícula de pó que se interponha entre eles causará danos a ambos (fenômeno de danificação do cabeçote, conhecido como *head crash*). Por essa razão, os sistemas de disco rígido são montados em recipientes lacrados na fábrica.

São utilizadas várias medidas para avaliar o desempenho de um sistema de disco: (1) **tempo de busca** (tempo necessário para mover os cabeçotes de leitura/gravação de uma trilha para outra); (2) **atraso de rotação** ou **tempo de latência** (metade do tempo utilizado para o disco executar uma rotação completa, que é o tempo médio necessário para os dados necessários chegarem debaixo do cabeçote de leitura/gravação, uma vez que este tenha sido posicionado na trilha desejada); (3) **tempo de acesso** (soma do tempo de busca e do atraso de rotação); (4) **tempo de transferência** (velocidade com que os dados são transferidos do disco para o computador ou vice-versa).

Em geral, os sistemas de discos rígidos apresentam características significativamente melhores que os de discos flexíveis. Considerando que os cabeçotes de leitura/gravação não tocam a superfície do disco em um sistema de disco rígido, é possível operá-lo a velocidades da ordem de 3000 a 4000 rotações por minuto, enquanto nos discos flexíveis a velocidade viável é da ordem de 300 rotações por minuto. Por conseguinte, as taxas de transferência para sistemas de disco rígido, normalmente medidas em *megabytes* por segundo, são muito maiores do que as de sistemas de disco flexível, geralmente medidas em *quilobytes* por segundo.

Considerando que a operação dos sistemas de disco requer movimento físico, tanto os discos rígidos como os flexíveis perdem para os circuitos eletrônicos quando comparados em matéria de velocidade. De fato, os tempos de espera no interior de um circuito eletrônico são medidos em nanossegundos (bilionésimos de segundo) ou menos, sendo que os tempos de busca, de latência e de acesso aos sistemas de disco são medidos em milissegundos (milésimos de segundo). Assim, o tempo necessário para obter dados de um sistema de disco pode parecer uma eternidade para um circuito eletrônico à espera de algum resultado.

## Discos óticos (*compact disks*)

Outra tecnologia popular de armazenamento em disco é o disco compacto (CD). Esses discos possuem 12 cm (aproximadamente 3 polegadas) de diâmetro e são feitos de material reflexivo, recoberto com uma camada protetora transparente. A informação é gravada neles criando variações na superfície reflexiva. Essa informação pode então ser recuperada por meio de um facho de raios *laser* que monitora as irregularidades da superfície refletiva do CD enquanto ele gira.

A tecnologia do CD foi originalmente aplicada a gravações de áudio, usando um formato de gravação conhecido como CD-DA (*compact disk-digital audio*), e os CDs usados hoje em dia para armazenamento de dados do computador usam essencialmente o mesmo formato. A informação nesses CDs específicos é gravada em uma única trilha disposta em espiral, como o sulco dos antigos discos musicais de vinil (Figura 1.10). (Diferentemente dos primeiros, a espiral em um CD vai do centro para fora). Essa trilha é dividida em unidades chamadas setores, e cada qual contém marcações de identificação e 2 KB disponíveis para os dados, que equivalem a 1/75 de segundo de música nas gravações de áudio.

Note que as distâncias ao longo da espiral são maiores perto da borda do que nas partes interiores. Para maximizar a capacidade de um CD, a informação é gravada com densidade linear uniforme na trilha inteira, o que significa que mais informação é guardada em uma volta na parte mais externa da espiral do que em uma volta na parte mais interna. Como conseqüência, mais setores são lidos em uma única rotação do disco quando o facho de *laser* está percorrendo a parte mais externa da trilha do que quando percorre a mais interna. Assim, para obter uma taxa de transferência de dados uniforme, os aparelhos de CD são projetados para variar a velocidade de rotação, dependendo da localização do facho de *laser*.

Como conseqüência dessa decisão de projeto, os sistemas de armazenamento com CD têm um melhor desempenho quando lidam com longas e contínuas cadeias de dados, como é o caso quando reproduzem música. Entretanto, quando uma aplicação exige acesso a itens de dados de uma maneira aleatória, a abordagem usada em armazenamento com discos magnéticos (trilhas individuais, concêntricas, sendo que cada uma contém o mesmo número de setores) supera em muito a abordagem em espiral usada nos CDs.


**Figura 1.10** Formato de armazenamento do CD.

Os CDs tradicionais têm capacidade de armazenamento em torno de 600 a 700 MB. Contudo, novos formatos, como o DVD (*digital versatile disk*) atingem capacidades na ordem de 10 GB. Tais CDs podem armazenar apresentações de multimídia nas quais os dados de áudio e vídeo são combinados para apresentar a informação de uma maneira mais interessante e informativa do que a obtida apenas com texto. De fato, a principal aplicação do DVD é como meio de gravação de um filme inteiro em um único CD.

## Fita magnética

Outra forma, mais antiga, de armazenamento em massa é a fita magnética (Figura 1.11). Aqui, a informação é registrada sobre uma película de material magnético, que recobre uma fita de plástico fina, enrolada sobre um carretel. Para se ter acesso aos dados, esta fita deve ser montada em um dispositivo, denominado unidade de fita, o qual tem a possibilidade de efetuar operações de leitura, gravação e rebobinamento da fita, sob controle do computador. As unidades de fita variam em tamanho, desde as pequenas unidades de cartucho, ou fitas *streaming*, que utilizam fitas semelhantes às dos gravadores de áudio, até as grandes unidades antigas, em que o transporte da fita se fazia de carretel para carretel. Embora a capacidade desses dispositivos dependa do formato utilizado na gravação, alguns podem comportar um volume de dados da ordem de vários *gigabytes*.

Os sistemas modernos dividem a fita em segmentos, cada qual magneticamente gravado por meio de um processo de formatação, semelhante ao dos dispositivos de armazenamento em disco. Cada segmento contém várias trilhas, que correm paralelamente umas às outras, ao longo da fita. O acesso a essas trilhas pode ser feito independentemente, ou seja, a fita consiste, em última instância, em numerosas e independentes cadeias de *bits*, de forma similar ao que ocorre nos setores em um disco.

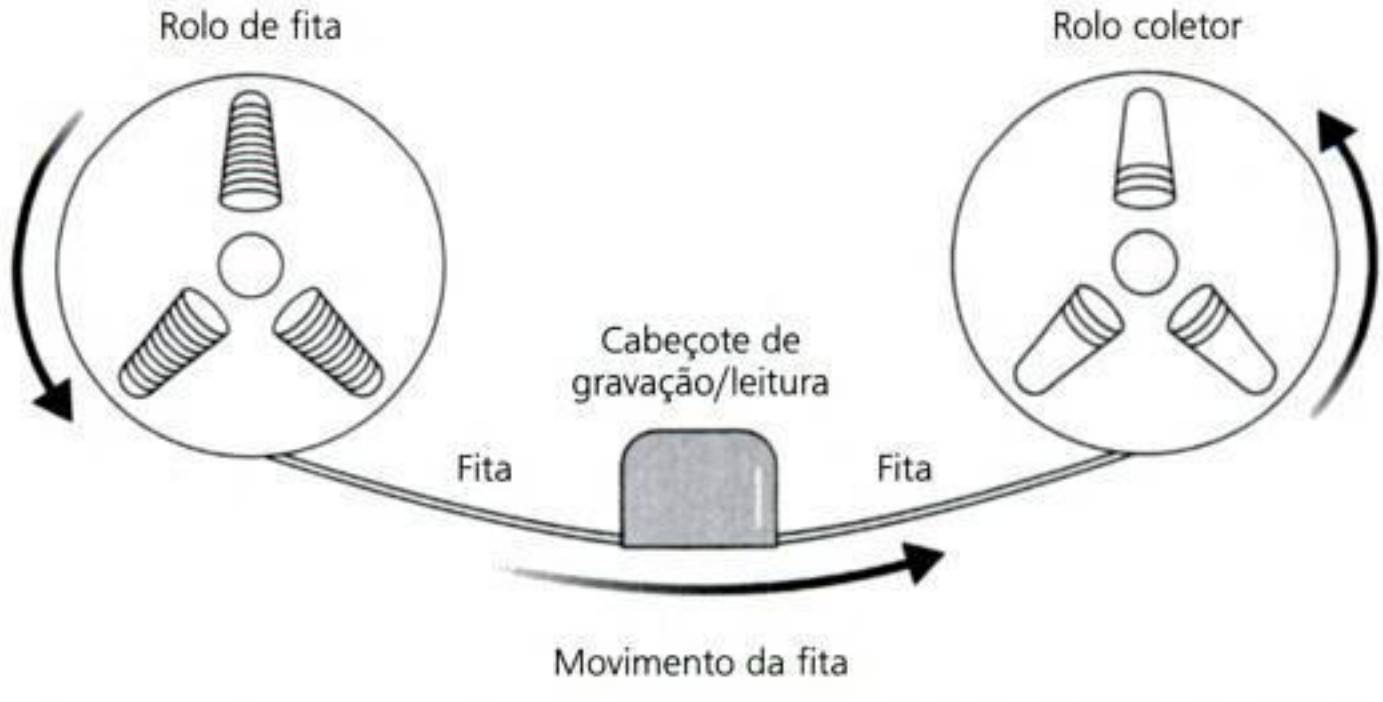



**Figura 1.11** Um mecanismo de armazenamento em fita magnética.

A maior desvantagem dos sistemas de fita é o fato de que a movimentação para posicionamento, entre posições diferentes de uma fita, pode ser muito demorada, devido à grande quantidade de fita que deve ser deslocada entre os carretéis. Assim, os sistemas de fita apresentam maiores tempos de acesso de dados do que os sistemas de disco, nos quais setores diferentes podem ser acessados por meio de pequenos movimentos do cabeçote de leitura/gravação. Assim, os sistemas de fita não são adequados para armazenamento de dados *on-line*. Contudo, quando o objetivo é o armazenamento *off-line* com o propósito de arquivamento (*back-up*), a alta capacidade e confiabilidade e o baixo custo da fita tornam essa tecnologia uma escolha popular entre os sistemas de armazenamento de dados atuais.

## Armazenamento e recuperação de arquivos

A informação é armazenada nos sistemas de armazenamento em massa em grandes unidades chamadas **arquivos**. Um arquivo típico pode consistir em um documento completo sob a forma de texto, uma fotografia, um programa, ou uma coleção de dados a respeito dos funcionários de uma companhia. As características físicas dos dispositivos de armazenamento em massa ditam que esses arquivos devem ser armazenados e recuperados em unidades com múltiplos *bytes*. Por exemplo, cada setor de um disco magnético deve ser obrigatoriamente tratado como uma longa cadeia de *bits*. Um bloco de dados dimensionado com base nas características físicas de um dispositivo de armazenamento recebe a denominação de **registro físico**. Assim, um arquivo guardado em um sistema de armazenamento em massa geralmente consiste em muitos registros físicos.

Ao contrário do particionamento dos dados em registros físicos cujos tamanhos são impostos pelas características físicas dos dispositivos de armazenamento, o arquivo a ser armazenado normalmente apresenta suas divisões naturais específicas. Assim, é conveniente que um arquivo que contenha informação relativa aos funcionários de uma companhia seja dividido em blocos, cada qual com a informação relativa a determinado funcionário. Tais agrupamentos naturais de dados, um para cada funcionário, são denominados **registros lógicos**.

Raramente os comprimentos dos registros lógicos combinam exatamente com os dos registros físicos em um dispositivo de armazenamento em massa. Por essa razão, pode-se encontrar vários registros lógicos em um único registro físico. Entretanto, também é possível que um registro lógico se divida em dois ou mais registros físicos (Figura 1.12). Por conseguinte, um certo desembaralhamento freqüentemente é associado à representação de dados provenientes de sistemas de armazenamento em massa.

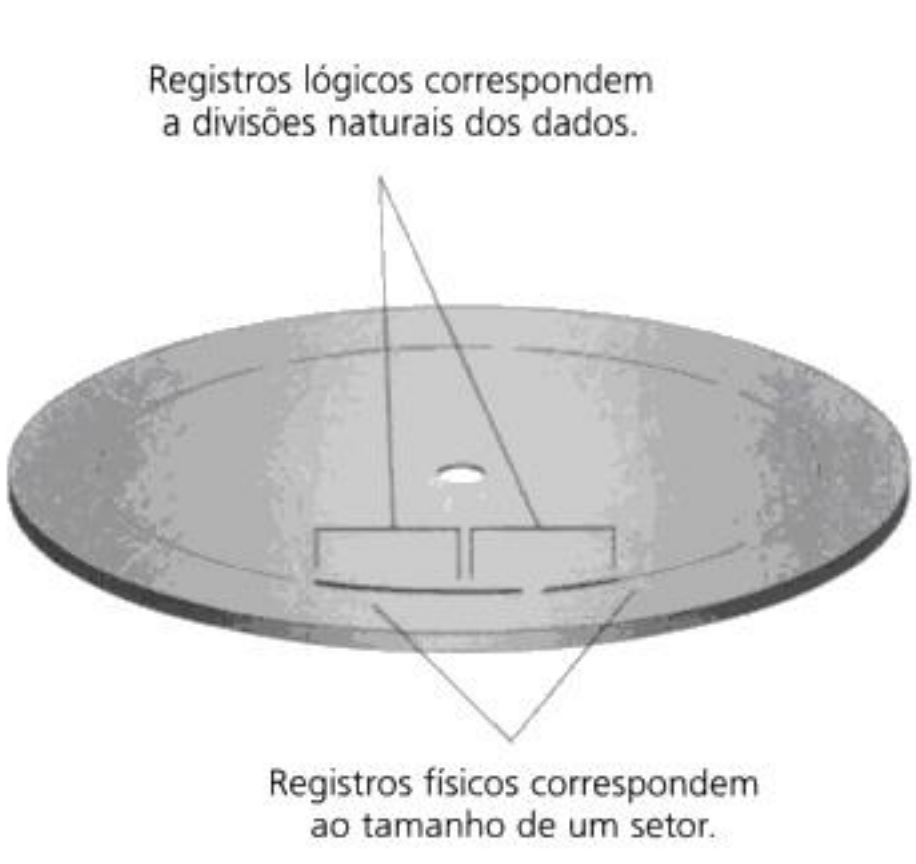



**Figura 1.12** Registros lógicos e registros físicos em um disco.

Uma solução comum a esse problema é reservar uma área de memória principal grande o suficiente para guardar vários registros físicos e usar esse espaço de memória como área de agrupamento. Em outras palavras, blocos de dados compatíveis com os registros físicos podem ser transferidos entre a memória principal e o sistema de armazenamento em massa, enquanto os dados residentes na memória principal podem ser referenciados em termos de registros lógicos.

Uma área de memória usada dessa maneira é chamada **retentor** (*buffer*). Em geral, um retentor é uma área de armazenamento para montar os dados temporariamente, em geral, durante o processo de transferência de um dispositivo para outro. Por exemplo, as impressoras modernas contêm circuitos de memória cuja maioria é usada como retentor para montar partes de um documento que foi transferido para a impressora mas ainda não foi impresso.

O uso de um retentor no contexto da transferência de dados entre um dispositivo de armazenamen-

to em massa e a memória principal de um computador exemplifica os papéis relativos da memória principal e do armazenamento em massa. A memória principal é usada para reter os dados a serem processados, enquanto o armazenamento em massa serve como depósito permanente para eles. Assim, a atualização de dados de um sistema de armazenamento em massa envolve a sua transferência para a memória principal, onde são feitas a atualização propriamente dita e a transferência dos dados atualizados de volta para o sistema.

Concluímos que a memória principal, os discos magnéticos, os CDs e as fitas magnéticas exibem graus decrescentes de acesso aleatório aos dados. O sistema de endereçamento usado na memória principal permite acesso rápido e aleatório a *bytes* individuais de dados. Os discos magnéticos permitem acesso aleatório somente a setores inteiros de dados. Além disso, a leitura de um setor envolve o tempo de busca e o atraso de rotação. Os discos óticos também permitem acesso aleatório aos setores individuais, mas os atrasos aí são maiores que os encontrados nos discos magnéticos, devido ao tempo adicional exigido para o posicionamento na trilha em espiral e o ajuste de velocidade de rotação do disco. Finalmente, as fitas magnéticas oferecem pouco em termos de acesso aleatório. Os sistemas de fita modernos marcam posições na fita de tal forma que diferentes segmentos da mesma possam ser referenciados individualmente, mas a estrutura física da fita implica tempos significativos para recuperar um segmento distante.

### *Questões/Exercícios*

1. Quais as vantagens que um sistema de disco rígido apresenta pelo fato de seus discos girarem mais rapidamente que os discos flexíveis?
2. Quando se gravam dados em um sistema de armazenamento com múltiplos discos, devemos preencher completamente uma superfície do disco antes de utilizar uma outra, ou primeiro encher um cilindro inteiro antes de iniciar a gravação em outro cilindro?
3. Por que devem ser armazenados em disco, e não em fita, os dados referentes a um sistema de reservas que seja atualizado com muita freqüência?
4. Algumas vezes, quando modificamos um documento usando um processador de textos, ao se adicionar texto, o tamanho aparente do arquivo no disco não aumenta, mas em outras vezes a adição de um único símbolo pode aumentar o tamanho aparente do arquivo em várias centenas de *bytes*. Por quê?

## 1.4 Representação da informação como padrões de *bits*

Uma vez consideradas as técnicas para armazenar *bits*, agora consideramos como a informação pode ser codificada como padrões de *bits*. Nosso estudo vai focalizar os métodos populares para codificação de texto, dados numéricos, imagens e sons. Cada um desses sistemas tem repercussões que normalmente são visíveis a um usuário típico de computador. Nosso objetivo é entender suficientemente essas técnicas, de modo a podermos reconhecer as suas reais conseqüências.

### Representação de texto

A informação na forma de texto normalmente é representada por meio de um código, no qual se atribui a cada um dos diferentes símbolos do texto (letras do alfabeto e caracteres de pontuação) um único padrão de *bits*. O texto fica então representado como uma longa cadeia de *bits*, na qual padrões sucessivos representam os símbolos sucessivos no texto original.

## American Standard National Institute

O American Standard National Institute (ANSI) foi fundado em 1918 por um pequeno consórcio formado por sociedades de engenharia e agências governamentais como organização sem fins lucrativos para coordenar o desenvolvimento de padrões espontâneos no setor privado. Atualmente, entre os membros do ANSI, incluem-se mais de 1300 empresas, organizações profissionais, associações comerciais e agências governamentais. O ANSI está localizado em Nova York e representa os EUA como membro do ISO. Seu sítio na Web está no endereço http://www.ansi.org.

Organizações similares em outros países incluem Standards Australia (Austrália), Standards Council of Canada (Canadá), Chine State Bureau of Quality and Technical Supervision (China), Deutsches Institut für Normung (Alemanha), Japanese Industrial Standards Committee (Japão), Dirección General de Normas (México), State Committee of the Russian Federation for Standardization and Metrology (Rússia), Swiss Association for Standardization (Suíça) e British Standards Institution (Reino Unido).

Nos primórdios da computação, muitos códigos diferentes foram projetados e utilizados em associação com diferentes partes do equipamento computacional, acarretando a correspondente proliferação de problemas de comunicação. Para aliviar essa situação, o **American National Standard Institute** (ANSI) adotou o **American Standard Code for Information Interchange** (ASCII, pronunciado como *asc-i-i*). Este código utiliza padrões de sete *bits* para representar letras maiúsculas e minúsculas do alfabeto, símbolos de pontuação, os dígitos de 0 a 9 e certas informações de controle, como mudanças de linha (*line feed*), posicionamento no início de uma linha (*carriage return*) e tabulações (*tab*). Hoje em dia, o código ASCII tem sido, muitas vezes, estendido para um formato de oito *bits* por símbolo, mediante a inserção de um 0 (zero) adicional, como ***bit*** mais significativo, em cada padrão de sete *bits*. Esta técnica não apenas gera um código cujos padrões se ajustam perfeitamente a uma célula de um *byte* de memória, como permite 128 padrões adicionais de *bits* (obtidos atribuindo-se o valor 1 (um) ao ***bit*** extra) e a representação de símbolos que não figuram no código ASCII original. Infelizmente, como cada fabricante tende a dar sua própria interpretação a esses padrões adicionais de *bits*, os dados em que tais padrões figuram não costumam apresentar fácil portabilidade entre programas de fabricantes diferentes.

O apêndice A mostra uma parte do código ASCII, em um formato de oito *bits* por símbolo, e a Figura 1.13 demonstra que, neste sistema, o padrão de *bits*

```
01001000 01100101 01101100 01101100 01101111 00101110
```

representa a palavra "Hello."

## ISO — A International Organization for Standardization

A International Organization for Standardization (comumente chamada ISO) foi criada em 1947 como federação mundial de organismos de padronização, cada qual de um país. Atualmente, sua sede fica em Genebra, na Suíça, e possui mais de 100 organizações-membros, bem como numerosos membros correspondentes. (Um membro correspondente geralmente é de um país que não tem um organismo de padronização reconhecido nacionalmente. Esses membros não podem participar diretamente do desenvolvimento de padrões, mas são mantidos informados das atividades da ISO.) A ISO mantém um sítio na Web no endereço http://www.iso.ch.

Embora o ASCII tenha sido o código mais utilizado durante muitos anos, estão ganhando popularidade outros códigos de maior alcance, capazes de representar documentos em diversos idiomas. Um deles, o **Unicode**, foi desenvolvido por alguns dos mais proeminentes fabricantes de *hardware* e *software* e está rapidamente ganhando apoio na comunidade da computação. Esse código usa um padrão único de 16 *bits* para representar cada símbolo. Por conseguinte, o Unicode consiste em 65.536 diferentes padrões de *bits* — o suficiente para representar os símbolos mais comuns dos idiomas chinês e japonês. Um código que provavelmente competirá com o Unicode foi desenvolvido pela **International Organization for Standardization** (também conhecida como **ISO**, em alusão à palavra grega *isos*, que significa *igual*). Utilizando padrões de 32 *bits*, esse sistema de codificação tem a capacidade de representar bilhões de símbolos.

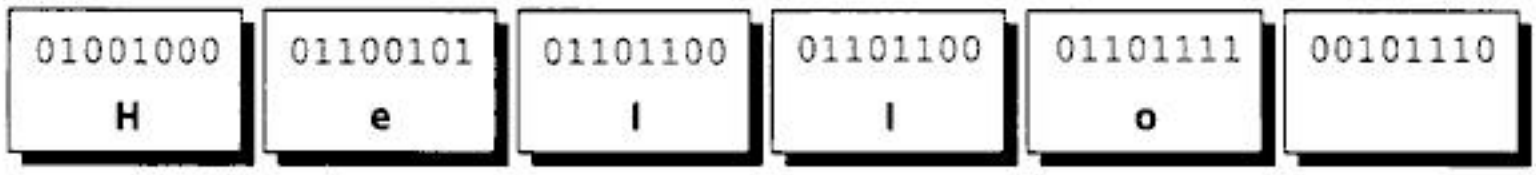



**Figura 1.13** A mensagem "Hello." em ASCII.

## Representação de valores numéricos

Embora seja muito útil o método de armazenar informação na forma de caracteres codificados, ele é ineficiente quando a informação a ser registrada é puramente numérica. Para ilustrar, suponha que desejemos armazenar o número 25. Se insistirmos em armazená-lo na forma de símbolos codificados em ASCII, utilizando um *byte* por símbolo, precisaremos de um total de 16 *bits*. Além disso, 99 é o maior número que podemos armazenar desta forma, utilizando 16 *bits*. Um modo mais eficiente de efetuar esse armazenamento seria representar em base dois, ou seja, em notação binária, o valor numérico desejado.

A forma binária permite representar valores numéricos mediante o uso exclusivo dos dígitos 0 e 1, em vez dos dez dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 do sistema decimal tradicional. Sabe-se que, na base dez, cada posição, na denotação do número, está associada com uma quantidade. Na representação numérica de 375, o dígito 5 está localizado na posição associada às unidades, o dígito 7, na posição associada às dezenas e o dígito 3, na associada às centenas (Figura 1.14a). O peso correspondente a cada posição é dez vezes maior do que o associado à posição imediatamente à sua direita. O valor representado pela expressão completa é obtido multiplicando-se o valor representado em cada dígito pela quantidade associada à sua posição relativa, e somando-se todos esses resultados intermediários. No exemplo dado, o padrão 375 representará, portanto, (3 × *cem*) + (7 × *dez*) + (5 × *um*).

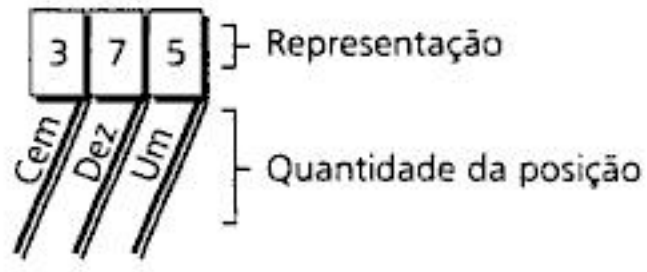
Na notação binária, a posição de cada dígito de uma representação numérica também está associada a uma quantidade. O peso associado a cada posição é o dobro do peso associado à posição imediatamente à sua direita. Mais precisamente, o dígito mais à direita está associado ao peso *um* ($2^0$); a posição imediatamente à sua esquerda ao peso *dois* ($2^1$); a seguinte, ao peso *quatro* ($2^2$); a próxima, a *oito* ($2^3$), e assim por diante. Por exemplo, seja a seqüência 1011. Em base binária, o dígito mais à direita, 1, está na posição associada ao peso *um*, o 1 à sua esquerda está na posição associada ao peso *dois*, o 0 seguinte, ao peso *quatro*, e o dígito 1 mais à esquerda, a *oito* (Figura 1.14b).

Para recuperar o valor de uma representação binária, seguimos o mesmo procedimento visto para a base dez — multiplicamos o valor de cada dígito pela quantidade associada à sua posição e somamos todos os resultados intermediários assim obtidos. Por exemplo, o valor representado por 100101 é 37, como mostra a Figura 1.15. Note-se que, como a notação binária só utiliza os dígitos 0 e 1, o processo de multiplicação e soma se reduz apenas à adição das quantidades associadas às posições ocupadas por dígitos 1 na representação. Assim, o padrão binário 1011 representa o valor onze, porque os dígitos 1 figuram nas posições associadas aos pesos um, dois e oito.

Note-se que a seqüência de representações binárias, obtida ao contar de zero a oito, é a seguinte:

```
0
1
10
```

**a. O sistema de base dez**




**b. O sistema de base dois**

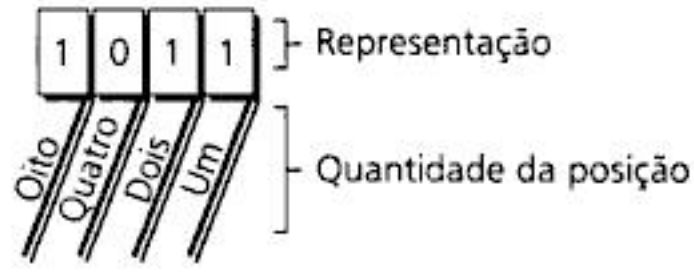



**Figura 1.14** Os sistemas decimal e binário.

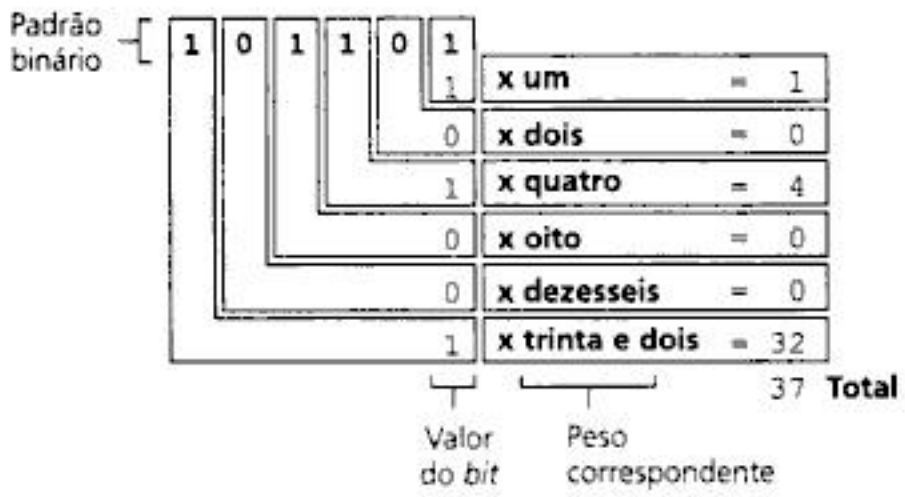



**Figura 1.15** Decodificação da representação binária 100101.

```
11
100
101
110
111
1000
```

Há numerosas formas de gerar esta seqüência e, embora não seja brilhante quanto ao seu conteúdo teórico, elas permitem a rápida obtenção da representação binária de valores numéricos pequenos. Para exemplificar, imagine-se o hodômetro de um automóvel cujo mostrador seja formado por componentes que apresentem apenas os dígitos 0 e 1. O hodômetro é inicialmente posicionado em 0 e passa a mostrar o dígito 1 conforme o automóvel se desloca. Em seguida, quando este 1 retorna novamente para 0, na posição à sua esquerda no mostrador, deverá aparecer o dígito 1, produzindo-se desta maneira o padrão 10. Em seguida, o dígito 0 mais à direita evolui para 1, produzindo 11. Prosseguindo, este dígito 1, da posição mais à direita do mostrador, volta outra vez para 0, provocando uma alteração no dígito à sua esquerda, fazendo-o passar novamente de 1 para 0. Isto provoca, de modo semelhante, o aparecimento de outro 1 na posição mais à esquerda, produzindo assim o padrão 100.

Para encontrar as representações binárias de valores grandes, é preferível adotar uma forma mais sistemática, como a descrita pelo algoritmo da Figura 1.16. Apliquemos este algoritmo para o valor treze (Figura 1.17). Primeiro, dividimos o número *treze* por *dois*, obtendo *seis* como quociente e *um* como resto. Como o quociente não é nulo, o passo seguinte é dividi-lo por *dois*, obtendo outro quociente, igual a *três*, e o novo resto, igual a *zero*. O novo quociente ainda não é *zero*, assim temos de dividi-lo novamente por *dois*, obtendo um quociente e um resto iguais a *um*. Mais uma vez, dividimos o novo quociente (*um*) por *dois* e, desta vez, obtemos um quociente igual a *zero*, e o resto igual a *um*. Agora, tendo obtido um quociente nulo, passamos ao passo 3 do algoritmo, em que concluímos que a representação binária do valor original (*treze*) é 1101.

Retomemos agora o problema de armazenar o valor 25, com o qual começamos esta subseção. Como vimos, seriam necessários dois *bytes* para armazenar o valor usando um código ASCII por *byte*, e o maior valor que poderíamos armazenar nesses dois *bytes* seria 99. Contudo, se usarmos a notação binária, poderemos armazenar qualquer número inteiro na faixa de 0 a 65.535 em apenas dois *bytes*, o que é uma melhora fantástica.

Por essas e outras razões, é comum representar a informação numérica por meio de uma forma de notação binária, em vez de símbolos codificados. Dizemos uma "forma de notação binária" porque o sistema binário simples descrito anteriormente serve como base para diversas técnicas de representação de números em computadores. Algumas dessas variantes do sistema binário são discutidas adiante, neste capítulo. Por ora, devemos notar apenas que um sistema, conhecido como notação de complemento de dois, é comum para representar números inteiros por oferecer uma forma conveniente de representação para números negativos e positivos. Para a representação de números com partes fracionárias, como 4½ ou ¾, outra técnica, chamada notação de vírgula flutuante, é utilizada. Assim, um valor específico (como 25) pode ser representado por meio de diferentes padrões de *bits* (em caracteres codificados que usem ASCII, em complemento de dois, ou em notação de vírgula flutuante, como

**Passo 1.** Divida o valor a representar por dois e armazene o resto.
**Passo 2.** Enquanto o quociente não for zero, continue dividindo o quociente mais recente por dois e armazene o resto.
**Passo 3.** Agora que o quociente é zero, a representação binária do valor original poderá ser obtida concatenando-se, da direita para a esquerda, na ordem em que foram calculados, os restos das divisões realizadas no passo 2.

**Figura 1.16** Um algoritmo para encontrar a representação binária de um inteiro positivo.

25⁰/₂); de maneira recíproca, a cada padrão de *bits*, diversas interpretações também podem ser associadas.

Encerrando esta seção, devemos mencionar um problema dos sistemas de armazenamento numérico a ser estudado adiante em maior profundidade. Independentemente do tamanho do padrão de *bits* que uma máquina possa alocar para o armazenamento e valores numéricos, sempre haverá valores excessivamente grandes ou frações demasiadamente pequenas para que possam ser representados no espaço disponível. Decorre um risco permanente da ocorrência de erros de representação, tais como estouro (*overflow*) — valores muito grandes e o truncamento (frações muito pequenas), que deve ser levado em conta para evitar que um usuário desavisado venha a se deparar com miríades de dados incorretos em seus programas.

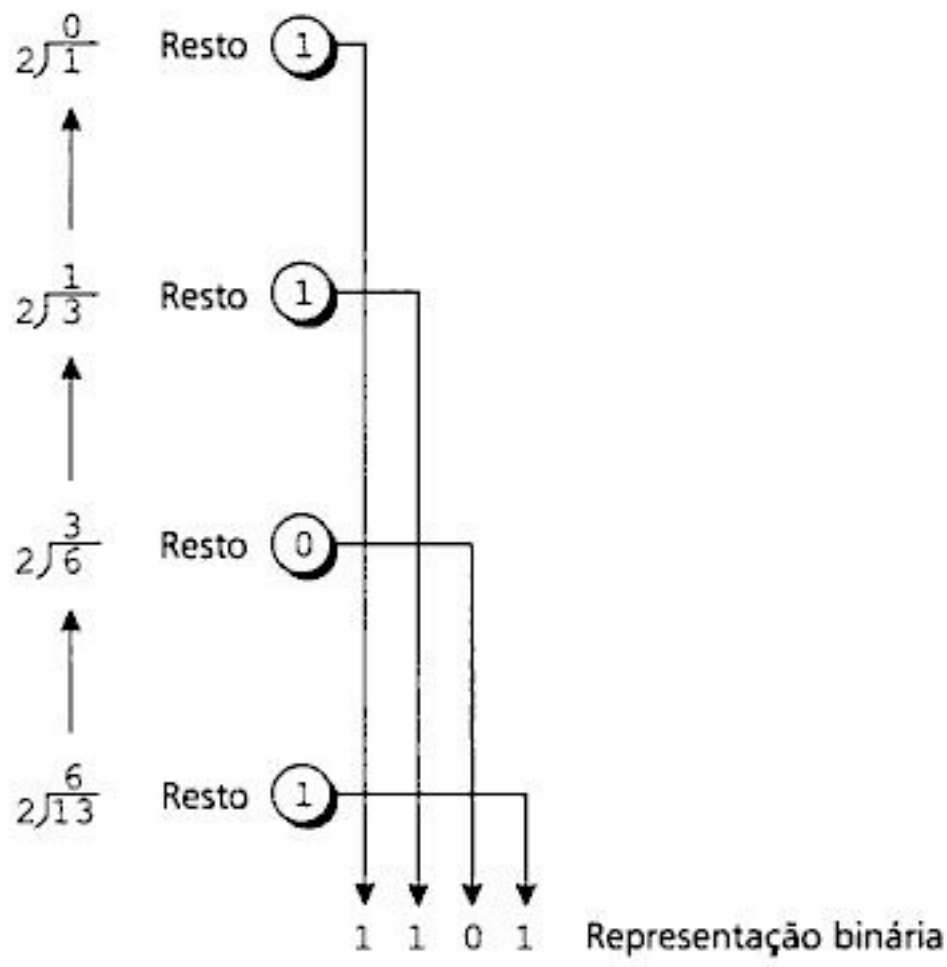



**Figura 1.17** Aplicação do algoritmo da Figura 1.15 para obter a representação binária do número 13.

## Representação de imagens

As aplicações dos computadores modernos envolvem não somente caracteres e dados numéricos, mas também figuras, áudio e vídeo. Em comparação com os sistemas de armazenamento restritos a caracteres e dados numéricos, as técnicas para a representação de dados nessas formas adicionais estão em sua infância e, portanto, ainda não suficientemente padronizadas na comunidade de processamento de dados.

As técnicas populares para representar imagens podem ser classificadas em duas categorias: **mapas de *bits*** e **vetoriais**. No caso das técnicas de mapas de *bits*, uma imagem é considerada uma coleção de pontos, cada qual chamado ***pixel***, abreviatura de *picture element* ("elemento de imagem"). Em sua forma mais simples, uma imagem é representada por uma longa cadeia de *bits*, os quais representam as linhas de *pixel* da imagem, onde cada ***bit*** é 1 ou 0, dependendo de o correspondente *pixel* ser preto ou branco. As imagens coloridas são ligeiramente mais complicadas, uma vez que cada *pixel* pode ser representado por uma combinação de *bits* que indicam a sua cor. Quando as técnicas de mapas de *bits* são usadas, o padrão de *bits* resultante é chamado mapa de *bits*, o que significa que o padrão de *bits* é pouco mais que um mapa da imagem que está sendo representada.

A maioria dos periféricos dos computadores modernos, tais como aparelhos de fax, câmeras de vídeo e digitalizadores de imagem, convertem imagens coloridas na forma de mapas de *bits*. Esses dispositivos normalmente registram a cor de cada *pixel* em três componentes — um vermelho, um verde e um azul — que correpondem às cores primárias. Um *byte* é usado para representar a intensidade de cada componente de cor. Assim, três *bytes* de armazenamento são necessários para representar um único *pixel* da imagem original. Essa abordagem de três componentes por *pixel* também corresponde à maneira como a maioria dos monitores de computador exibe imagens. Esses dispositivos exibem uma miríade de *pixels*, em que cada um consiste em três componentes — um vermelho, um verde e um azul — como pode ser observado inspecionando-se a tela de perto. (Você pode preferir usar uma lente de aumento.)

O formato dos três *bytes* por *pixel* significa que uma imagem que consiste em 1024 linhas de 1024 *pixels* (uma fotografia comum) exige vários *megabytes* para ser armazenada, o que excede a capacidade dos discos flexíveis comuns. Na Seção 1.8, consideraremos duas técnicas populares (GIF e JPEG) usadas para comprimir essas imagens em tamanhos mais manejáveis.

Uma desvantagem das técnicas de mapas de *bits* é que uma imagem não pode ser facilmente ampliada ou reduzida para qualquer tamanho. Com efeito, o único modo de ampliar uma imagem é aumentar o tamanho dos *pixels*, o que leva a uma aparência granulada — um fenômeno que também ocorre nas fotografias baseadas em filme. As técnicas vetoriais proporcionam os meios de resolver esses

problemas de escala. Em tais sistemas, uma imagem é representada por uma coleção de linhas e curvas. Essa descrição deixa os detalhes de como as linhas e curvas são desenhadas para o dispositivo que efetivamente produz a imagem, em vez de insistir em que o dispositivo reproduza um padrão particular de *pixels*. As várias fontes de caracteres disponíveis nas impressoras atuais e nos monitores normalmente são codificadas dessa maneira para permitir maior flexibilidade no tamanho do caractere, o que resulta em **fontes em escala** (*scalable fonts*). Por exemplo, o TrueType (desenvolvido pela Microsoft e pela Apple Computer) é um sistema projetado para descrever a maneira como os símbolos utilizados nos textos podem ser desenhados. Do mesmo modo, o Postscript (desenvolvido pela Adobe Systems) fornece meios de descrever não apenas caracteres, mas também outros dados pictóricos mais gerais. As representações vetoriais também são populares nos sistemas de projeto assistido por computador (CAD), onde os desenhos das linhas correspondentes a objetos em três dimensões são exibidos e manipulados na tela do computador. Contudo, as técnicas vetoriais não conseguem a qualidade fotográfica que se obtém com os mapas de *bits*. Isso explica porque as técnicas de mapas de *bits* são utilizadas nas câmeras digitais modernas.

## Representação de som

O modo mais geral de codificar a informação de áudio para armazená-la e manipulá-la no computador é tirar uma amostragem da amplitude da onda de som em intervalos regulares e registrar a série de valores obtidos. Por exemplo, a série 0; 1,5; 2,0; 1,5; 2,0; 3,0; 4,0; 3,0; 0 pode representar uma onda de som que cresce em amplitude, cai brevemente, aumenta para um nível mais alto e cai para zero. (Figura 1.18). Essa técnica, que usa uma taxa de amostragem de 8.000 amostras por segundo, tem sido usada há anos nas comunicações telefônicas de longa distância. A voz, em uma extremidade da linha, é codificada sob a forma de valores numéricos que representam a amplitude do som a cada oito milésimos de segundo. Esses valores numéricos são então transmitidos pela linha à extremidade receptora, onde são usados para reproduzir o som da voz.

Embora 8.000 amostras por segundo possa parecer uma taxa rápida, ela não é suficiente para a gravação musical com alta fidelidade. Para obter a qualidade de reprodução do som conseguida nos CDs atuais, é usada uma taxa de 44.100 amostras por segundo. Os dados obtidos em cada amostra são representados com 16 *bits* (32 *bits* para gravações estereofônicas). Conseqüentemente, cada segundo de música gravada em estéreo exige mais de um milhão de *bits*.

Uma alternativa, o sistema de codificação mais econômico, conhecido como *Musical Instrument Digital Interface* (MIDI, pronuncia-se "midi") é largamente utilizado nos sintetizadores encontrados nos teclados eletrônicos, nos equipamentos de jogos e nos efeitos sonoros que acompanham os sítios na Web. Codificando as diretrizes para produzir música em um sintetizador em vez de codificar o som produzido, o MIDI evita a grande quantidade de armazenamento necessária à técnica de amostragem. Mais precisamente, o MIDI codifica qual instrumento deve tocar qual nota por quanto tempo, o que significa que uma clarineta tocando a nota Ré por dois se-

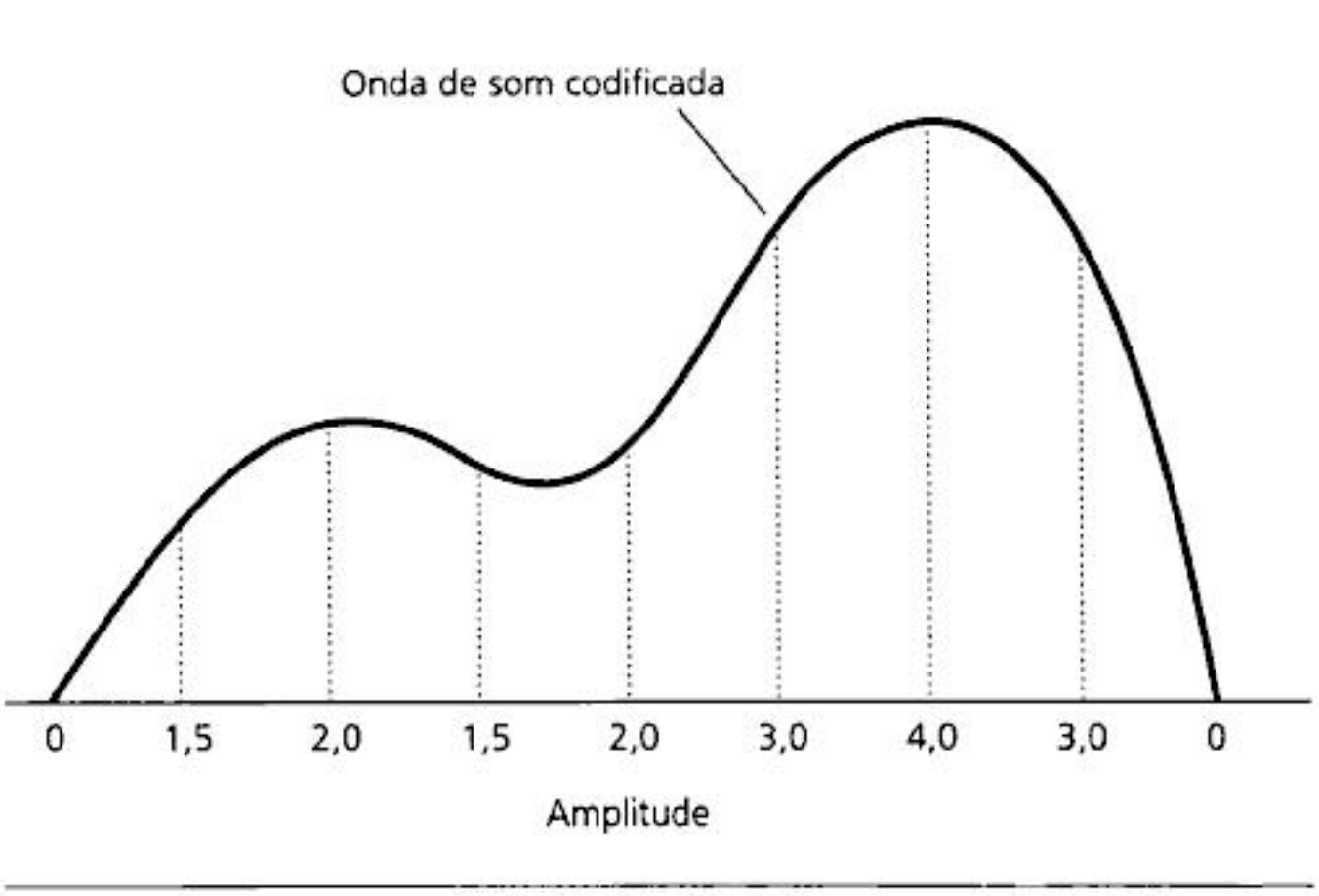



**Figura 1.18** A onda de som representada pela seqüência.

gundos pode ser codificada em três *bytes* em vez de mais de dois milhões de *bits* necessários se ela for gravada com taxa de 44.100 amostras por segundo.

Em resumo, o MIDI pode ser entendido como um meio de codificar a partitura de uma música em vez de codificar a música. Assim, uma "gravação" MIDI pode ser significativamente diferente quando tocada em sintetizadores diferentes.

## QUESTÕES/EXERCÍCIOS

**1.** O que está escrito na mensagem abaixo, codificada em ASCII, com oito *bits* por símbolo?

```
01000011 01101111 01101101 01110000 01110101 01110100
01100101 01110010 00100000 01010011 01100011 01101001
01100101 01101110 01100011 01100101
```

**2.** No código ASCII, qual a relação entre o código de uma letra maiúscula e o da mesma letra, porém minúscula?

**3.** Codifique as sentenças abaixo em ASCII:
   a. *Where are you?**
   b. *"How?" Cheryl asked.***
   c. 2 + 3 = 5.

**4.** Descreva um dispositivo do cotidiano que possa assumir um de dois estados, como, por exemplo, uma bandeira em um mastro, que pode estar hasteada ou arriada. Associe o símbolo 1 a um dos estados e 0 ao outro e mostre o aspecto do código ASCII para a letra *b* quando representado com tais *bits*.

**5.** Converta os seguintes códigos binários na forma decimal:
   a. `0101` b. `1001` c. `1011` d. `0110`
   e. `10000` f. `10010`

**6.** Converta as seguintes representações decimais em código binário:
   a. 6 b. 13 c. 11 d. 18
   e. 27 f. 4

**7.** Qual o maior valor numérico que poderá ser representado com três *bytes* se cada dígito for codificado na forma de um padrão ASCII por *byte*? E se for utilizada a notação binária?

**8.** Uma alternativa para a notação hexadecimal, na representação de padrões de *bits*, é a *notação decimal pontuada* (*dotted decimal notation*), na qual cada *byte* do padrão é representado pelo seu equivalente em base dez. Estes códigos de um *byte* estão, por sua vez, separados por pontos. Por exemplo, 12.5 representa o padrão 0000110000000101 (o *byte* 00001100 é representado por 12 e o 00000101, por 5), e o padrão 100010000001010000000000111 é representado por 136.16.7. Represente os seguintes padrões de *bits* nesta notação.
   a. `0000111100001111` b. `00110011000000010000000`
   c. `0000101010100000`

**9.** Qual é a vantagem de representar imagens por meio de técnicas vetoriais em relação à representação por mapas de *bits*? E qual é a desvantagem?

**10.** Suponha que uma gravação estereofônica de uma hora de música tenha sido codificada usando uma taxa de 44.100 amostras por segundo, como discutido no texto. Compare o tamanho da versão codificada com a capacidade de armazenamento de um CD.

---

*N. de T. *Onde você está?*
**N. de T. *"Como?", Cheryl perguntou.*

# 1.5 O sistema binário

Antes de analisar as técnicas de armazenamento numérico utilizadas nas máquinas atuais, precisamos de maiores detalhes sobre o sistema de representação binária.

## Adição binária

Para somar dois valores representados em notação binária, iniciamos da mesma maneira como se ensina nas escolas de ensino fundamental para a base dez, memorizando a tabuada de adição (Figura 1.19). Usando essa tabuada, somamos dois valores como se segue: primeiro, somamos os dois dígitos da coluna mais à direita; escrevendo o dígito menos significativo desta soma logo abaixo desta coluna, transportamos o dígito mais significativo desta soma parcial (se houver) para cima da coluna imediatamente à esquerda e prosseguimos somando esta coluna. Por exemplo, para efetuar a conta:

```
  00111010
+ 00011011
```

Começamos somando os dígitos da coluna mais à direita (0 e 1), obtendo 1, o qual escrevemos sob esta coluna. Agora somamos os dígitos 1 e 1 da próxima coluna e obtemos 10. Escrevemos o 0 deste 10 sob esta coluna e levamos o 1 para o topo da coluna seguinte. Neste ponto, nossa solução fica assim:

```
    1
  00111010
+ 00011011
       01
```

Somamos o 1, o 0 e o 0 da próxima coluna, obtendo 1, e escrevemos o 1 sob esta coluna. Os dígitos 1 e 1 da coluna seguinte totalizam 10. Escrevemos o 0 sob esta coluna e levamos o 1 para a próxima coluna. Neste ponto, a solução se torna:

```
    1
  00111010
+ 00011011
      0101
```

Os dígitos 1, 1 e 1 na coluna imediata totalizam 11. Escrevemos o 1 da direita sob esta coluna e levamos o outro 1 ao topo da coluna seguinte. Somamos este 1 com o 1 e o 0 que já estavam naquela coluna, obtendo 10. Novamente, escrevemos o 0 menos significativo e levamos o 1 para a coluna seguinte. Temos então:

```
   1
  00111010
+ 00011011
     010101
```

Agora somamos os dígitos 1, 0 e 0 da penúltima coluna, obtendo 1, o qual escrevemos sob esta coluna, nada havendo para levar para a próxima. Finalmente, somamos os dígitos da última coluna, que resulta em 0, que escrevemos sob esta última coluna. Nossa solução fica, então:

```
  00111010
+ 00011011
  01010101
```

```
  0      1      0      1
+ 0    + 0    + 1    + 1
  0      1      1     10
```

**Figura 1.19** As tabuadas de adição binária.

## Frações de números binários

Para estender a notação binária de forma que acomode a representação de frações, também utilizamos uma **vírgula*** (*radix point*), que funciona do mesmo modo que a vírgula da notação decimal. Em outras palavras, os dígitos à esquerda da vírgula representam a parte inteira do valor, sendo interpretados exatamente como no sistema binário previamente discutido. Os dígitos à direita da vírgula representam a parte fracionária do valor, sendo interpretados de modo semelhante aos outros *bits*, exceto em que às suas posições são associadas quantidades fracionárias: à primeira posição à direita da vírgula, é associada a quantidade $^1/_2$; à posição seguinte, a quantidade $^1/_4$; à próxima, $^1/_8$, e assim por diante. Note-se que isto é apenas uma extensão da regra previamente estabelecida: cada posição é associada a uma quantidade que é o dobro da quantidade associada à posição à sua direita. Com estes pesos associados às posições dos *bits*, é possível decodificar uma representação binária que contenha uma vírgula usando exatamente o mesmo procedimento empregado no caso em que não existe esta vírgula. Em particular, multiplicamos o valor correspondente a cada *bit* pela quantidade associada à posição por ele ocupada na representação do número. Para ilustrar, a representação binária 101.101 é decodificada em $5^5/_8$, como mostra a Figura 1.20.

### *Alternativas ao sistema binário*

Os antigos computadores não aproveitavam o sistema de notação binária. De fato, a maneira como os valores numéricos deveriam ser representados nas máquinas computacionais era tema de ativos debates em fins da década de 1930 e durante a de 1940. Um candidato era o sistema biquinário, onde cada dígito na representação de base dez de um número era substituído por dois – um tinha o valor 0,1,2,3, ou 4 e o outro era 0 ou 5 – de tal forma que a soma equivalesse ao dígito original. Este foi o sistema usado no ENIAC. Outro candidato era a notação em base oito. No artigo "Binary Calculation", que apareceu no *Journal of the Institute or Actuaries*, em 1936, E. W. Phillips escreveu: "A meta final é persuadir o mundo civilizado inteiro a abandonar a numeração decimal e em seu lugar usar a numeração octal; parar de contar em dezenas e sim em oitavas".

### *Analógico* versus *digital*

Um debate nos primórdios da computação era se os dispositivos computacionais deveriam ser baseados na tecnologia digital ou analógica. Em um sistema digital, um valor é representado por uma coleção de dispositivos; cada qual pode representar um número limitado de dígitos distintos (tais como 0 e 1). Em um sistema analógico, um valor é representado por um único dispositivo que pode conter qualquer valor em uma faixa contínua.

Vamos comparar as duas abordagens em termos de um recipiente d'água. Para simular um sistema digital, poderíamos convencionar que um recipiente vazio representasse o dígito 0 e um cheio, o dígito 1. Então armazenaríamos um valor numérico em uma carreira de recipientes usando a forma da notação binária. Em oposição a isso, poderíamos simular um sistema analógico enchendo parcialmente um único recipiente até o ponto em que o nível da água correspondesse ao valor numérico representado. A princípio, o sistema analógico pode parecer mais preciso, uma vez que não está sujeito a erros (tais como truncamento) inerentes ao sistema digital. Contudo, qualquer movimento no recipiente no sistema analógico pode causar erros de leitura do nível d'agua, enquanto uma perturbação significativa teria de ocorrer no sistema digital para que a distinção entre o recipiente cheio e o vazio não pudesse ser feita. Assim, o sistema digital é menos sensível a erros do que o analógico. Essa robustez é a principal razão pela qual muitas aplicações originalmente baseadas na tecnologia analógica (tais como comunicação telefônica, gravação de áudio e televisão) estão se deslocando para a digital.

*N. de T. Este ponto (ponto decimal) é substituído por vírgula em muitos países, inclusive o Brasil.

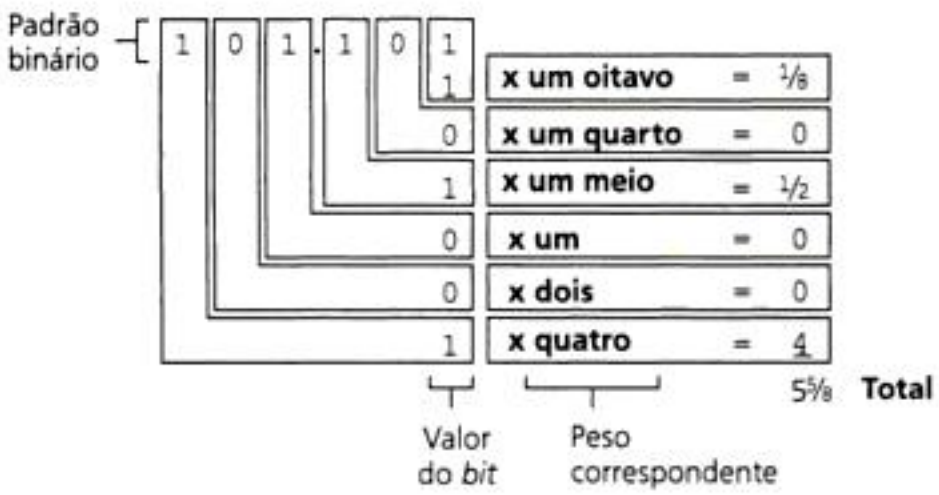



**Figura 1.20** Decodificação da representação binária 101.101.

Quanto à adição, as técnicas aplicadas no sistema decimal também são aplicáveis no sistema binário. Assim, para somarmos duas representações binárias que contenham vírgula, basta alinharmos as vírgulas e aplicarmos o mesmo processo de adição estudado anteriormente. Por exemplo, 10,011 somado a 100,11 produz 111,001, conforme mostrado a seguir:

```
  10,011
+ 100,110
  111,001
```

### *QUESTÕES/EXERCÍCIOS*

**1.** Converta as seguintes representações binárias em sua forma decimal equivalente:
a. 101010 b. 100001 c. 10111 d. 0110
e. 11111

**2.** Converta as seguintes representações decimais em sua forma binária equivalente:
a. 32 b. 64 c. 96 d. 15
e. 27

**3.** Converta as seguintes representações binárias nas suas representações equivalentes em base dez:
a. 11,01 b. 101,111 c. 10,1 d. 110,011
e. 0,101

**4.** Expresse os seguintes valores na notação binária:
a. $4^1/_2$ b. $2^3/_4$ c. $1^1/_8$ d. $^5/_{16}$ e. $5^5/_8$

**5.** Efetue as seguintes adições em notação binária:

```
a.  11011      b.  1010,001      c.  11111      d.  111,11
  + 1100         +    1,101        +     1        +   0,01
```

## 1.6 A representação de números inteiros

Há muito tempo, os matemáticos se interessam pelos sistemas de notação numérica, e muitas de suas idéias mostraram-se compatíveis com o projeto de circuitos digitais. Nesta seção, consideramos dois desses sistemas: notação de complemento de dois e notação de excesso, usadas para representar valores inteiros nos equipamentos de computação. Esses sistemas são baseados no sistema binário apresentado na Seção 1.5, mas apresentam propriedades adicionais que fazem com que os sistemas fiquem mais compatíveis com o projeto de computadores. Junto com as vantagens, porém, surgem as desvantagens. Nossa meta é entender essas propriedades e seu efeito no uso dos computadores.

### A notação de complemento de dois

O sistema mais conhecido para a representação interna de inteiros nos computadores modernos é a **notação de complemento de dois**. Este sistema emprega um número fixo de *bits* para

representar cada valor numérico. Nos equipamentos atuais, é comum o uso da notação de complemento de dois na qual cada valor é representado por um padrão de 32 *bits*. Esse sistema é grande e permite uma variedade de números a serem representados, mas é impróprio para fins de demonstração. Assim, para estudar as propriedades dos sistemas de complemento de dois, nos concentraremos nos sistemas menores.

A Figura 1.21 ilustra dois sistemas completos de complemento de dois — um baseado em padrões de *bits* de comprimento três e o outro, em padrões de *bits* de comprimento quatro. Tais sistemas são construídos iniciando-se com uma cadeia de zeros com o comprimento adotado e então contando em binário até que o padrão seja formado por um 0 seguido de 1s. Estes padrões representam os valores 0, 1, 2, 3,... Os padrões que representam valores negativos são obtidos começando com uma cadeia de 1s de comprimento apropriado e contando em binário, em ordem decrescente, até que o padrão obtido seja formado de um 1 seguido de 0s. Estes padrões representarão os valores –1, –2, –3,... (Se houver dificuldade para contar em binário, em ordem decrescente, pode-se começar pelo final da tabela, com o padrão formado por um 1 seguido de 0s, e contar em ordem crescente até que seja obtido o padrão formado só de 1s.)

Note-se que, em um sistema de complemento de dois, o *bit* mais à esquerda do padrão indica o sinal do valor representado. Assim, ele freqüentemente é chamado ***bit* de sinal**. Em um sistema de complemento de dois, os valores negativos são representados por padrões cujo *bit* de sinal é 1; valores não-negativos são representados por padrões cujo *bit* de sinal é 0.

Em um sistema de complemento de dois, existe uma relação conveniente entre padrões que representam números positivos e negativos de mesma magnitude. Estes padrões são idênticos quando lidos da direita para a esquerda, até a ocorrência do primeiro 1, inclusive. Desta posição em diante, os padrões são o complemento um do outro. (O complemento de um padrão é obtido mudando-se todos os 0s para 1s e todos os 1s para 0s; assim, 0110 e 1001 são complementos um do outro.) Por exemplo, no sistema de quatro *bits* da Figura 1.21, os padrões que representam 2 e –2 apresentam dígitos finais 10, porém o padrão que representa 2 começa com 00, enquanto o que representa –2 começa com 11. Isto conduz a um algoritmo de conversão de padrões de *bits* para representar números positivos e negativos de mesma magnitude. O algoritmo consiste em copiar o padrão original da direita para a esquerda até o aparecimento do primeiro *bit* 1, substituindo-se, em seguida, os demais *bits* originais pelos seus complementos, ou seja, trocando-se todos os demais 1s por 0s e 0s por 1s (Figura 1.22). (Note que o maior valor negativo em um sistema de complemento de dois não tem um correspondente positivo dentro do sistema.)

**a. Utilização de padrões de comprimento três**

| Padrão de *bits* | Valor representado |
|---|---|
| 011 | 3 |
| 010 | 2 |
| 001 | 1 |
| 000 | 0 |
| 111 | -1 |
| 110 | -2 |
| 101 | -3 |
| 100 | -4 |

**b. Utilização de padrões de comprimento quatro**

| Padrão de *bits* | Valor representado |
|---|---|
| 0111 | 7 |
| 0110 | 6 |
| 0101 | 5 |
| 0100 | 4 |
| 0011 | 3 |
| 0010 | 2 |
| 0001 | 1 |
| 0000 | 0 |
| 1111 | -1 |
| 1110 | -2 |
| 1101 | -3 |
| 1100 | -4 |
| 1011 | -5 |
| 1010 | -6 |
| 1001 | -7 |
| 1000 | -8 |

**Figura 1.21** Duas tabelas de conversão para a notação de excesso de oito.

A compreensão dessas propriedades básicas da notação de complemento de dois leva à construção de um algoritmo para a decodificação de representações numéricas nesta notação. Se o padrão a ser decodificado apresentar um *bit* de sinal igual a 0, simplesmente leremos o seu valor como se o padrão estivesse de-

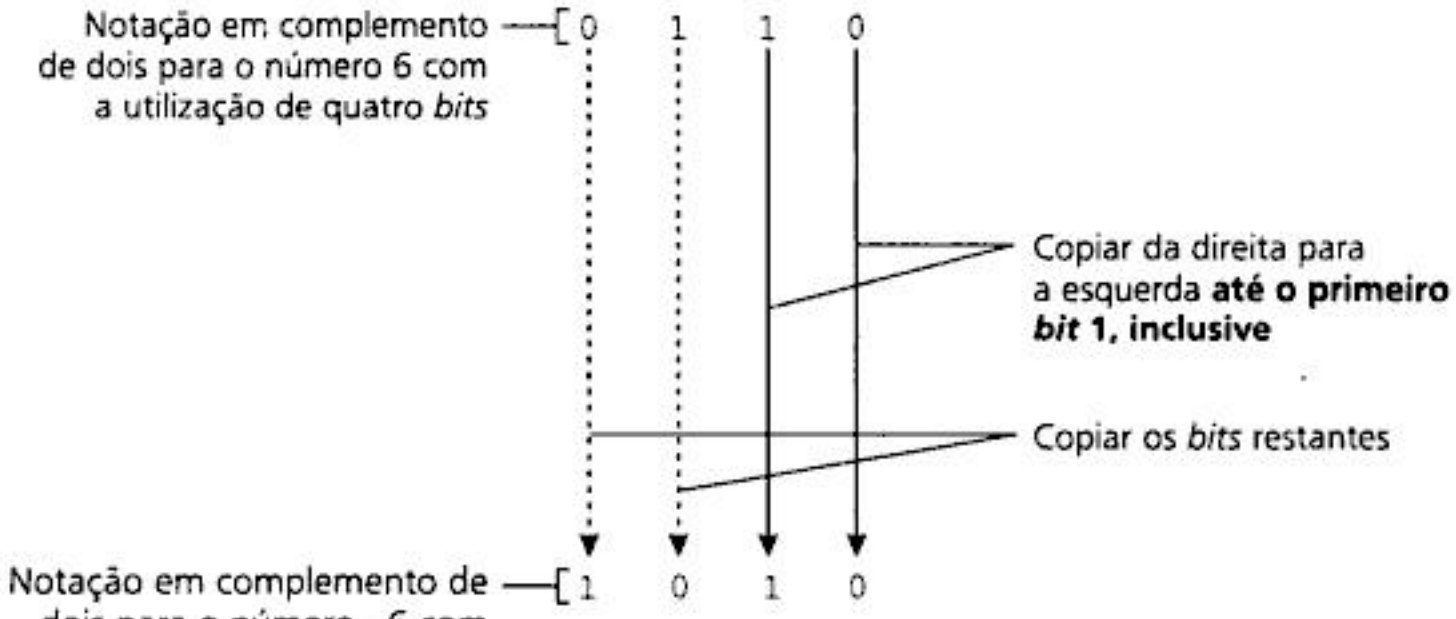



**Figura 1.22** Codificação, em quatro *bits*, do valor –6, em notação de complemento de dois.

notado em código binário. Por exemplo, 0110 representa o número 6, porque 110 significa 6 em notação binária. Se o padrão a ser decodificado tiver 1 como *bit* de sinal, saberemos que o valor representado é negativo, e tudo o que restará a fazer será determinar a magnitude do número. Isto é feito copiando, da direita para a esquerda, o padrão original, até o aparecimento do primeiro dígito 1, complementando os demais *bits* para finalmente decodificarmos o padrão resultante, como se fosse uma representação binária.

Por exemplo, para decodificar o padrão 1010, verificamos primeiro que o *bit* de sinal é 1, logo o valor representado é negativo. Em seguida, convertemos o padrão para 0110 e verificamos que isto representa 6, logo concluímos que o padrão original representa –6.

**Adição na notação de complemento de dois** Para somar valores representados em complemento de dois, aplicamos o mesmo algoritmo empregado na adição binária, desde que todos os padrões de *bits*, inclusive o resultado, sejam do mesmo comprimento. Isto significa que quando se efetua uma soma em complemento de dois, deve ser descartado qualquer *bit* extra, gerado no último passo da operação de adição, à esquerda do resultado. Assim, a soma de 0101 com 0010 resulta 0111, e a de 0111 com 1011 resulta 0010 (0111 + 1011 = 10010, que é truncado para 0010).

Compreendido esse método, considerem-se as três contas de adição mostradas na Figura 1.23. Em cada caso, traduzimos o problema para a notação de complemento de dois (usando padrões de comprimento quatro), executamos o processo de adição descrito previamente e decodificamos o resultado, retornando à notação decimal usual.

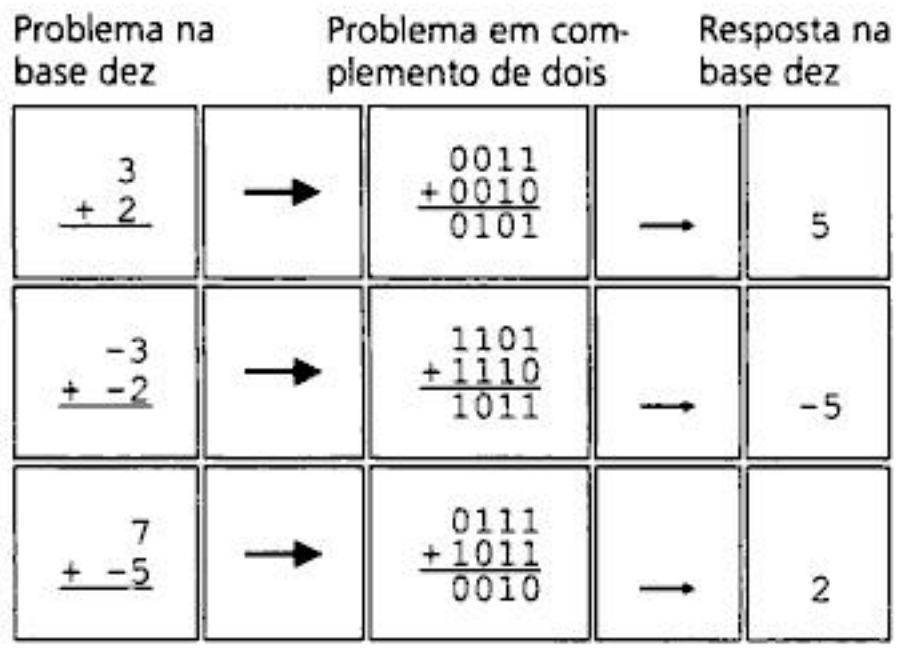



**Figura 1.23** Problemas de adição, convertidos na notação de complemento de dois.

Observe-se que se fossem usadas as técnicas tradicionais ensinadas nas escolas de ensino fundamental, a terceira conta (subtração) exigiria um procedimento completamente diferente das outras duas. Por outro lado, ao expressar tais operações na notação de complemento de dois, podemos obter o resultado correto em todos os casos, aplicando o mesmo algoritmo. Esta é, então, a vantagem da notação de complemento de dois: a adição de qualquer combinação de números, positivos e negativos, pode ser efetuada usando-se um mesmo algoritmo e, portanto, o mesmo circuito.

Em contraste com o que ocorre com um estudante do ensino fundamental, que aprende primeiro a somar para depois subtrair, uma máquina que usa a notação de complemento de dois precisa apenas ser capaz de somar e complementar. Por exemplo, o problema de subtração 7 – 5 equivale à adição 7 + (–5).

Por conseguinte, se for solicitado a uma máquina que subtraia 5 (armazenado como 0101) de 7 (armazenado como 0111), ela primeiro mudará 5 para –5 (representado como 1011), e então executará o algoritmo da adição para efetuar 0111 + 1011, obtendo o resultado 0010, que representa 2, conforme demonstrado a seguir:

```
  7       0111       0111
- 5  →   - 0101   →  + 1011
                      0010  →  2
```

Vemos então que, quando a notação de complemento de dois é usada para representar valores numéricos, um circuito somador, combinado com um circuito para alterar o sinal do número, é o suficiente para resolver os problemas de adição e subtração. (Tais circuitos são mostrados e explicados no Apêndice B.)

**O problema do estouro** Um problema que evitamos tratar nos exemplos anteriores é que, em qualquer sistema de complemento de dois, existe sempre um limite para o tamanho dos números a serem representados. Quando usamos complemento de dois, com padrões de quatro *bits*, ao valor 9 não está associado padrão algum e, por isso, não conseguimos obter uma resposta correta para a soma 5 + 4. De fato, o resultado apareceria como –7. Este erro é chamado **estouro**, o problema que ocorre quando o valor a ser representado cai fora da faixa permitida. Quando usado o complemento de dois, isto pode ocorrer ao se adicionar dois valores positivos ou dois negativos. Nos dois casos, a condição pode ser detectada verificando-se o *bit* de sinal do resultado. Em outras palavras, um estouro será indicado se a adição de dois valores positivos resultar em um padrão de valor negativo, ou se a adição de dois valores negativos resultar em um número positivo.

Evidentemente, uma vez que a maioria das máquinas manipula padrões de *bits* maiores do que os usados em nossos exemplos, valores maiores podem ser manipulados sem causar estouro. Hoje é comum usar 32 *bits* para armazenar valores em complemento de dois, permitindo valores positivos até 2.147.483.647 antes que ocorra estouro. Se forem necessários valores maiores, será possível usar padrões de *bits* mais longos ou talvez mudar as unidades de medida. Por exemplo, encontrar uma solução em milhas em vez de polegadas resulta no uso de números menores que podem ainda proporcionar a precisão necessária.

O fato é que os computadores podem errar. Assim, a pessoa que usa a máquina deve estar a par dos perigos envolvidos. Um problema é que os programadores e usuários tornam-se complacentes e ignoram o fato de que pequenos valores podem ser acumulados e produzir grandes números. Por exemplo, no passado, era comum usar padrões de 16 *bits* para representar valores na notação de complemento de dois, o que significa que os estouros não ocorreriam até que o valor $2^{15} = 32.768$ fosse alcançado. No dia 19 de setembro de 1989, o sistema computacional de um hospital falhou após anos de serviços confiáveis. Uma inspeção cuidadosa revelou que esta data era 32.768 dias após o 1º de janeiro de 1900. O que você acha que ocorreu?

## A notação de excesso

Outro método para codificar números inteiros é a **notação de excesso**. Neste sistema, cada número é codificado como um padrão de *bits*, de comprimento convencionado. Para estabelecer um sistema de excesso, primeiro escolhemos o comprimento do padrão a ser empregado; em seguida, escrevemos todos os diferentes padrões de *bits* com este comprimento, na ordem em que seriam gerados se estivéssemos contando em binário. Logo, observamos que o primeiro desses padrões, que apresenta um dígito 1 como seu *bit* mais significativo, figura aproximadamente no centro dessa lista. Escolhemos este padrão para representar o valor *zero*; os padrões que o seguem serão utilizados para representar 1, 2, 3...; os que o precedem serão adotados para a representação dos inteiros negativos –1, –2, –3,... O código resultante, para padrões de quatro *bits* de comprimento, é mostrado na Figura 1.24, na qual podemos observar que o valor 5 é representado pelo padrão 1101 e –5, por 0011. (Note

| Padrão de *bits* | Valor representado |
|---|---|
| 1111 | 7 |
| 1110 | 6 |
| 1101 | 5 |
| 1100 | 4 |
| 1011 | 3 |
| 1010 | 2 |
| 1001 | 1 |
| 1000 | 0 |
| 0111 | −1 |
| 0110 | −2 |
| 0101 | −3 |
| 0100 | −4 |
| 0011 | −5 |
| 0010 | −6 |
| 0001 | −7 |
| 0000 | −8 |

**Figura 1.24** Uma tabela de conversão para a notação de excesso de oito.

| Padrão de *bits* | Valor representado |
|---|---|
| 111 | 3 |
| 110 | 2 |
| 101 | 1 |
| 100 | 0 |
| 011 | −1 |
| 010 | −2 |
| 001 | −3 |
| 000 | −4 |

**Figura 1.25** Um sistema de notação de excesso, com padrões de três *bits* de comprimento.

que a diferença entre um sistema de excesso e um de complemento de dois é que os *bits* de sinal são trocados.)

O sistema apresentado na Figura 1.24 é conhecido como a notação de excesso de oito. Para entender o porquê desta denominação, interpretemos primeiramente cada um dos padrões codificados, utilizando para isso o sistema binário tradicional, e então comparemos os resultados desta observação com os valores representados no código de excesso. Em cada caso, poderemos constatar que a interpretação binária é maior do que a interpretação do código de excesso de oito. Por exemplo, o padrão 1100 normalmente representa o valor 12, mas em nosso sistema de excesso, representa 4; 0000 representa 0, mas no sistema de excesso, representa –8. Do mesmo modo, um sistema de excesso baseado em padrões de comprimento cinco seria chamado de notação de excesso de 16, porque o padrão 10000, por exemplo, seria usado para representar o zero, em vez do seu valor habitual de 16. Do mesmo modo, podemos confirmar que o sistema de excesso que utiliza padrões de três *bits* seria conhecido como notação de excesso de quatro (Figura 1.25).

## QUESTÕES/EXERCÍCIOS

**1.** Converta em decimais as seguintes representações em complemento de dois:

a. 00011 b. 01111 c. 11100 d. 11010
e. 00000 f. 10000

**2.** Converta as seguintes representações decimais na notação de complemento de dois em padrões de oito *bits*:

a. 6 b. –6 c. –17 d. 13
e. –1 f. 0

**3.** Suponha que os seguintes padrões de *bits* representem valores em notação de complemento de dois. Encontre a representação, em complemento de dois, do negativo de cada valor:

a. 00000001 b. 01010101 c. 11111100
d. 11111110 e. 00000000 f. 01111111

**4.** Suponha um computador que represente números na notação de complemento de dois. Quais os maiores e menores números representáveis utilizando padrões com os seguintes comprimentos?

a. quatro b. seis c. oito

**5.** Nas seguintes contas, cada padrão de *bits* representa um valor em notação de complemento de dois. Encontre a resposta para cada uma utilizando o processo de adição descrito no texto. A seguir, confira os resultados, resolvendo o mesmo problema em notação decimal.

a. 0101 + 0010 b. 0011 + 0001 c. 0101 + 1010 d. 1110 + 0011 e. 1010 + 1110

**6.** Faça as seguintes contas na notação de complemento de dois, mas desta vez preste atenção à ocorrência de estouro e indique quais resultados resultam incorretos devido a tal ocorrência.

a. `0100` + `0011`
b. `0101` + `0110`
c. `1010` + `1010`
d. `1010` + `0111`
e. `0111` + `0001`

**7.** Converta as seguintes contas, da notação decimal para a notação de complemento de dois, usando padrões de *bits* de comprimento quatro. Converta-as então à forma de uma adição equivalente (como um computador faria) e finalmente efetue a adição. Confira suas respostas retornando-as à notação decimal.

a. 6 − (−1)
b. 3 − 2
c. 4 − 6
d. 2 − (−4)
e. 1 − 5

**8.** Quando se somam números na notação de complemento de dois, pode ocorrer estouro quando um valor é positivo e o outro é negativo? Explique sua resposta.

**9.** Converta as seguintes representações de excesso de oito em seu equivalente decimal sem consultar a tabela do texto:

a. `1110` b. `0111` c. `1000`
d. `0010` e. `0000` f. `1001`

**10.** Converta as seguintes representações decimais na notação de excesso de oito sem consultar a tabela do texto:

a. 5 b. −5 c. 3 d. 0 e. 7 f. −8

**11.** O valor 9 pode ser representado na notação de excesso de oito? E quanto a representar o número 6 em notação de excesso de quatro? Explique sua resposta.

## 1.7 A representação de frações

Em contraste com o que ocorre no caso dos números inteiros, a representação de valores com parte fracionária requer não apenas o armazenamento do padrão de 0s e 1s que representa sua notação binária, mas também deve conter uma informação sobre a posição da vírgula*. Uma notação muito usada, que permite atingir tal objetivo, inspira-se na notação científica e é conhecida como **notação de vírgula flutuante**.

### A notação de vírgula flutuante

Vamos explicar a notação de vírgula flutuante por meio de um exemplo que emprega somente um *byte* de memória. Embora os computadores normalmente usem padrões mais longos, este exemplo é representativo dos casos reais e serve para demonstrar os principais conceitos sem as dificuldades trazidas pelos padrões longos de *bits*.

Primeiramente, escolhemos o *bit* mais significativo do *byte* para ser o *bit* de sinal do número. Novamente, um 0 neste *bit* significa que o valor representado é não-negativo, enquanto um 1 indica que é negativo. Em seguida, dividimos os sete *bits* restantes do *byte* em dois grupos, ou campos, o **campo de expoente** e o **campo de mantissa**. Vamos atribuir aos três *bits* que seguem o *bit* de sinal a função de campo de expoente, e aos quatro *bits* restantes, a de campo de mantissa. Assim, o *byte* fica dividido, conforme ilustra a Figura 1.26.

*N. de T. Vírgula binária, que separa as partes inteira e não-inteira dos números, similar à vírgula decimal dos números reais em base dez. Em inglês, usa-se o ponto.

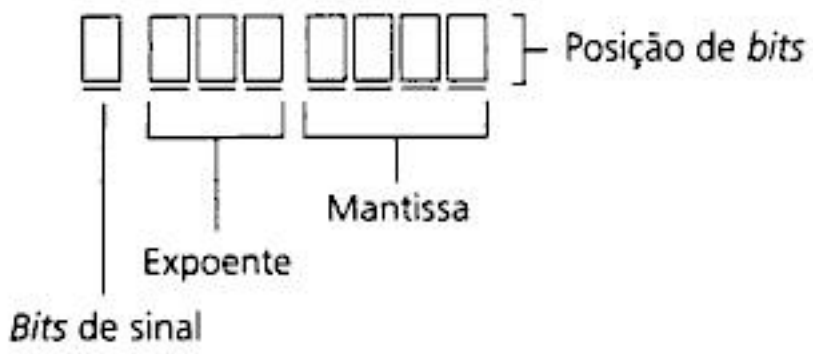



**Figura 1.26** Componentes da notação de ponto flutuante.

Podemos explicar o significado desses campos com a ajuda do exemplo seguinte. Suponha que um *byte* contenha o padrão de *bits* 01101011. Interpretando este padrão no formato que acabamos de definir, constatamos que o *bit* de sinal é 0, o expoente é 110, e a mantissa, 1011. Para decodificar o *byte*, extraímos primeiro a mantissa e colocamos a vírgula binária à sua esquerda, obtendo:

```
,1011
```

Em seguida, extraímos o conteúdo do campo do expoente (110) e o interpretamos como um inteiro codificado em três *bits* pelo método de representação de excesso (ver Figura 1.25). Assim, o padrão contido no campo de expoente do nosso exemplo representa o número positivo 2. Isto indica que devemos deslocar a vírgula binária dois *bits* à direita. (Um expoente negativo indica um deslocamento para a esquerda.) Como resultado, obtemos:

```
10,11
```

que representa 2¾. Em seguida, notamos que o *bit* de sinal do nosso exemplo é 0; assim, o valor representado é não-negativo. Concluímos que 2¾ é o número representado pelo *byte* 01101011 nesta notação.

Como outro exemplo, consideremos o *byte* 10111100. Extraímos a mantissa, obtendo:

```
,1100
```

e deslocamos a vírgula binária para a esquerda, uma vez que o campo de expoente (011) representa o valor –1. Então, temos:

```
,01100
```

que representa $^3/_8$. Uma vez que o *bit* de sinal, no padrão original, é 1, o valor codificado é negativo. Concluímos que o padrão 10111100 representa o valor $-^3/_8$.

Para armazenar um valor usando a notação de vírgula flutuante, revertemos o processo anterior. Por exemplo, para codificar $1^1/_8$, primeiramente expressamos este valor na notação binária, obtendo 1,001. Em seguida, copiamos o padrão de *bits* no campo da mantissa, da esquerda para a direita, iniciando com o primeiro *bit* não-nulo da representação binária. Neste momento, o *byte* assume o aspecto:

```
_ _ _ _ 1 0 0 1
```

Agora, devemos preencher o campo de expoente. Para tanto, imaginamos o conteúdo do campo da mantissa com uma vírgula à sua esquerda e determinamos o número de posições e o sentido de deslocamento da vírgula, necessários à obtenção do número binário original. No nosso exemplo, verificamos que a vírgula no padrão ,1001 deve ser deslocada de um *bit* à direita para obtermos 1,001. Como, agora, o expoente deve ser um 1 positivo, colocamos 101 (que representa o valor 1 positivo na notação de excesso de quatro) no campo de expoente. Finalmente, preenchemos o *bit* de sinal com 0, porque o valor a ser codificado é não-negativo. Assim, o *byte* final fica:

```
0 1 0 1 1 0 0 1
```

Há um ponto sutil que talvez tenha passado despercebido ao se preencher o campo de mantissa. A regra é copiar o padrão de *bits* que aparece na representação binária da esquerda para a direita, começando com o 1 mais à esquerda. Para esclarecer, considere o processo de armazenar o valor $^3/_8$, que é ,011 na notação binária. Neste caso, a mantissa será

```
_ _ _ _ 1 1 0 0
```

Ela não será

_ _ _ _ 0 1 1 0

Isto porque preenchemos o campo de mantissa *começando com o 1 mais à esquerda* que aparece na representação binária. Essa regra elimina a possibilidade de múltiplas representações para o mesmo valor. Também implica que a representação de todos os valores diferentes de zero terá mantissa cujo primeiro *bit* seja 1. Essa representação está na **forma normalizada**. Note que o valor zero deve ser um caso especial; sua representação em vírgula flutuante é um padrão de *bits* 0s.

## Erros de truncamento

Consideremos o problema incômodo de representar o número $2^5/_8$ no sistema de vírgula flutuante de um *byte*. Primeiramente, escrevemos $2^5/_8$ em binário, obtendo 10,101. Entretanto, ao copiarmos este código no campo da mantissa, não haverá espaço suficiente, e o último 1 (o qual representa a última parcela $^1/_8$) se perderá (Figura 1.27). Se ignorarmos este problema por ora e continuarmos a preencher o campo de expoente e do *bit* de sinal, teremos o padrão de *bits* 01101010, que representa o valor $2^1/_2$, e não $2^5/_8$. O fenômeno assim observado é denominado **erro de truncamento** ou **erro de arredondamento**, ou seja, que parte do valor que está sendo armazenado se perdeu porque o campo de mantissa não é grande o suficiente.

A conseqüência de tais erros pode ser reduzida usando-se uma mantissa maior. De modo semelhante ao que ocorre com a representação de inteiros, é comum o uso de 32 *bits* na notação de vírgula flutuante em vez dos oito aqui utilizados. Esta abordagem também permite que o campo de expoente seja estendido, tornando-se mais longo. Entretanto, mesmo com estes formatos mais longos, há ocasiões em que se exige uma precisão maior ainda.

Outra fonte de erros de truncamento, muito comum na notação decimal, é o problema das dízimas periódicas, tais como a que ocorre, por exemplo, quando se tenta exprimir $^1/_3$ em decimal. Alguns valores não podem ser expressos com precisão absoluta, independentemente do número de dígitos utilizados para representá-los.

A diferença entre a notação decimal usual e a binária é a existência de um número muito maior de valores, na notação binária, cujas representações são dízimas. Por exemplo, o valor $^1/_{10}$ é uma dízima quando denotado em binário. Imagine os problemas com que se defronta um usuário incauto que se utilize da notação de vírgula flutuante para representar e manipular valores financeiros. Em particular, se a quantia de um real for utilizada como unidade de medida, nem sequer o valor de uma moeda de dez centavos poderá ser representado com precisão. Uma solução para este caso consiste em converter tais valores em centavos, de forma que todos os valores representados sejam inteiros, podendo assim ser representados com precisão mediante a codificação em complemento de dois.

Os erros de truncamento e os problemas deles decorrentes são uma preocupação constante de pessoas que trabalham com análise numéri-

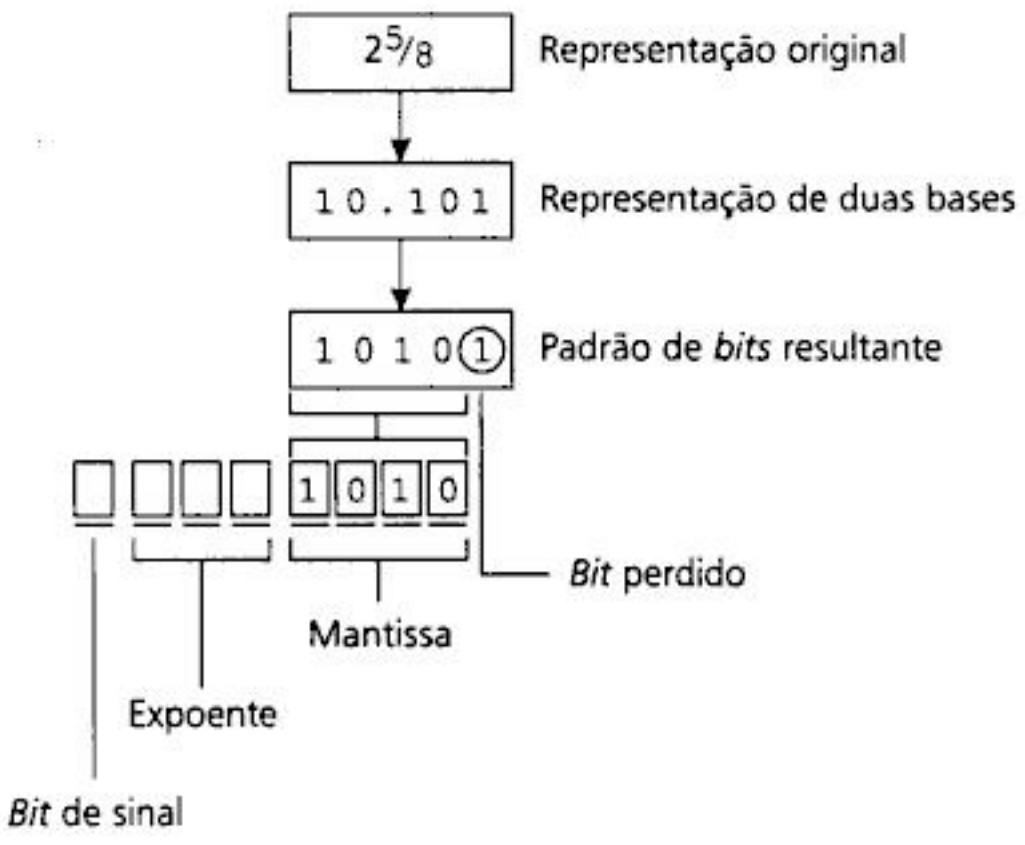



**Figura 1.27** Codificação do valor de $2^5/_8$.

ca. Este ramo da matemática enfrenta tais problemas ao processar dados reais, geralmente volumosos e que exigem uma precisão considerável.

Finalizamos esta seção com um exemplo capaz de entusiasmar qualquer analista numérico. Suponha que se deseje somar os três seguintes valores, utilizando a notação de vírgula flutuante de um *byte* definida anteriormente:

$$2^1/_2 + {}^1/_8 + {}^1/_8$$

Se somarmos os valores na ordem em que se encontram, primeiro somaremos $2^1/_2$ e $^1/_8$ e obteremos $2^5/_8$, o qual em binário é 10,101. Infelizmente, um erro de truncamento ocorre quando armazenamos este valor, então o resultado deste primeiro passo é igual a $2^1/_2$ (que é igual a uma das parcelas). O próximo passo é somar tal resultado com o último $^1/_8$. Aqui ocorre mais uma vez o mesmo erro de truncamento, o que resulta em um valor final incorreto, igual a $2^1/_2$.

Somemos agora os valores na ordem oposta. Primeiro, somemos $^1/_8$ com $^1/_8$ e obteremos $^1/_4$, que em binário seria ,01; assim, o resultado deste primeiro passo é armazenado em um *byte* como 00111000, que está correto. Somemos, a seguir, este ¼ com o próximo valor da lista, $2^1/_2$, e obteremos $2^3/_4$, o qual também pode ser armazenado com precisão em um *byte* como 01101011, gerando-se desta maneira a resposta correta.

Em resumo, quando se somam números em notação de vírgula flutuante, a ordem em que são somados pode ser importante. O problema é que se um número muito grande for somado com um muito pequeno, o pequeno poderá ser truncado. Assim, a regra geral para somar muitos valores é primeiramente somar os pequenos na esperança de que o valor acumulado seja significativo quando somado aos valores grandes. Este foi o fenômeno constatado no exemplo precedente.

Os projetistas dos pacotes comerciais atuais fazem um bom trabalho ao evitar que um usuário incauto sofra esse tipo de problema. Em uma planilha eletrônica típica, as respostas corretas são obtidas a menos que os valores somados difiram em tamanho por $10^{16}$ ou mais. Assim, se for necessário somar um ao valor

```
10.000.000.000.000.000
```

você pode obter a resposta

```
10.000.000.000.000.000
```

em vez de

```
10.000.000.000.000.001
```

Tais problemas são significativos em aplicações (como sistemas de navegação) nos quais pequenos erros podem ser combinados em computações adicionais, produzindo graves conseqüências, mas para um usuário comum de computador pessoal esse grau de exatidão é suficiente.

## QUESTÕES/EXERCÍCIOS

**1.** Decodifique os seguintes padrões de *bits* utilizando a notação de vírgula flutuante estudada no texto:

a. `01001010` b. `01101101` c. `00111001`
d. `11011100` e. `10101011`

**2.** Codifique os seguintes valores utilizando a notação de vírgula flutuante estudada no texto. Indique as ocorrências de erros de truncamento.

a. $2^3/_4$ b. $5^1/_2$ c. $^3/_4$ d. $-3^1/_2$ e. $-4^3/_8$

3. Na notação de vírgula flutuante discutida no texto, qual dos padrões 01001001 e 00111101 representa o maior valor? Descreva um procedimento simples para determinar o maior dentre os valores associados a dois padrões fornecidos.
4. Na notação de vírgula flutuante descrita no texto, qual o maior valor que pode ser representado? E qual o menor valor positivo que pode ser representado?

## 1.8 Compressão de dados

Com a finalidade de armazenar e transferir dados, freqüentemente é útil (e às vezes obrigatório) reduzir o seu tamanho. A técnica para isso se chama **compressão de dados**. Iniciamos esta seção considerando alguns métodos genéricos de compressão de dados; em seguida, apresentamos algumas abordagens projetadas especificamente para comprimir imagens.

### Técnicas genéricas de compressão

Numerosas técnicas de compressão de dados têm sido desenvolvidas, cada qual com seu cenário de melhor e pior caso. O método chamado **codificação de tamanho de seqüência** (*run-length encoding*) produz ótimo resultado quando os dados que estão sendo comprimidos consistem em longas seqüências com o mesmo valor. Com efeito, a codificação de tamanho de seqüência é o processo de substituir tais seqüências por um código indicativo do valor repetido e do número de vezes que ele ocorre na seqüência. Por exemplo, é necessário menos espaço para indicar que um padrão de *bits* consiste em 253 uns seguidos por 118 zeros seguidos por 87 uns, do que para listar os 458 *bits*.

Em alguns casos, a informação envolvida consiste em blocos de dados, em que cada um difere ligeiramente do anterior. Um exemplo seria os quadros consecutivos de um filme. Nesses casos, as técnicas que usam a **codificação relativa** são úteis. A abordagem consiste em registrar as diferenças entre blocos consecutivos em vez dos blocos inteiros, isto é, cada bloco é codificado em termos de sua relação com o bloco precedente.

Outra abordagem para reduzir o tamanho dos dados é a **codificação dependente da freqüência**, que é um sistema no qual o tamanho do padrão de *bits* usado para representar um item de dados é inversamente proporcional à freqüência de uso do item. Esses códigos são exemplos dos **códigos de comprimento variável**, o que significa que os itens são representados por padrões de tamanho diferente, em oposição a códigos como o Unicode, no qual todos os símbolos são representados usando 16 *bits*. Mais precisamente, na língua inglesa, as letras *e*, *t*, *a* e *i* são usadas mais freqüentemente do que as letras *z*, *q* e *x*. Assim, quando se constrói um código para texto em inglês, o espaço pode ser economizado usando pequenos padrões de *bits* para representar as primeiras letras e longos padrões para as últimas. O resultado é um código no qual um texto em inglês teria representação menor do que a que seria obtida com códigos de comprimento uniforme como o ASCII e o Unicode. David Huffman foi o descobridor de

### *Compressão do som*

Como aprendemos na Seção 1.4, um segundo de música estéreo codificada com uma taxa de 44.100 amostras por segundo necessita mais de um milhão de *bits* para ser armazenado. Essas exigências de espaço são aceitáveis para gravações musicais distribuídas em CDs, mas desafiam as capacidades da tecnologia quando combinadas com vídeo para produzir gravação de cinema. Assim, o *Motion Picture Expert Group* do ISO tem desenvolvido técnicas de compressão que reduzem significativamente a necessidade de espaço de armazenamento para áudio. Uma delas é conhecida como MP3 (MPEG-1 Audio Layer-3) que pode obter taxas de compressão de 12 para 1. Usando MP3, as gravações de música podem ser reduzidas para um tamanho que pode ser economicamente transmitido através da Internet – uma possibilidade que ameaça revolucionar a indústria de gravações musicais.

um algoritmo amplamente utilizado no desenvolvimento de códigos dependentes de freqüência, e é comum referir-se aos códigos desenvolvidos dessa maneira como **Códigos de Huffman**. De fato, a maioria dos códigos dependentes de freqüência atualmente em uso é código de Huffman.

Embora tenhamos introduzido a codificação de tamanho de seqüência, a codificação relativa e a codificação dependente de freqüência como técnicas genéricas de compressão, cada uma tende a ter seu próprio domínio de aplicação. No entanto, os sistemas baseados na **codificação de Lempel-Ziv** (assim chamada por causa de seus criadores Abraham Lempel e Jacob Ziv) são verdadeiramente genéricos. Com efeito, os usuários da Internet provavelmente já viram e talvez usaram programas que utilizam as técnicas Lempel-Ziv para comprimir arquivos, conhecidos como arquivos *zip*.

Os sistemas de codificação Lempel-Ziv são exemplos da **codificação adaptável de dicionário**. Aqui, o termo *dicionário* refere-se a uma coleção de módulos a partir da qual a mensagem a ser comprimida é construída. Se quisermos comprimir um texto em inglês, os módulos poderão ser os caracteres do alfabeto. Se quisermos comprimir dados já codificados como uma cadeia de 0s e 1s, os módulos poderão ser os dígitos 0 e 1. Em um sistema de codificação adaptável de dicionário, o dicionário pode mudar durante o processo de codificação. Por exemplo, no caso de texto em inglês, após codificar parte da mensagem, podemos decidir adicionar *ing* e *the* ao dicionário. Assim, qualquer nova ocorrência de *ing* e *the* pode ser codificada com uma única referência ao dicionário, em vez de três. Os sistemas de codificação Lempel-Ziv usam maneiras inteligentes e eficientes para adaptar o dicionário durante o processo de codificação (compressão).

Como exemplo, vamos considerar como poderíamos comprimir uma mensagem usando um sistema Lempel-Ziv particular, conhecido como LZ77. Iniciamos tomando a parte inicial da mensagem. Então representamos o restante da mesma como seqüências de triplas (que consistem em dois números e um símbolo da mensagem), e cada uma descreve como a próxima parte da mensagem deve ser construída em função das partes precedentes. (Assim, o dicionário a partir do qual a mensagem foi construída consiste na própria mensagem).

Por exemplo, considere a mensagem comprimida

xyxxyzy(5, 4, x)

que consiste no segmento inicial *xyxxyzy* seguido pela tripla (5, 4, x). A cadeia *xyxxyzy* é a parte da mensagem que já está na forma descompactada. Para descompactar o resto da mensagem, devemos decodificar a tripla (5, 4, x) como especificado na Figura 1.28. O primeiro número da tripla diz-nos até que ponto devemos contar retroativamente na cadeia descompactada. No caso, contamos retroativamente cinco símbolos, o que nos leva ao segundo *x* da esquerda para a direita da cadeia descompactada. Agora acrescentamos ao final desta os símbolos encontrados nesta posição. O segundo número da tripla nos diz quantos símbolos consecutivos devem ser acrescentados. No caso, este número é 4; assim, acrescentamos os símbolos *xxyz* ao fim da cadeia descompactada e obtemos

xyxxyzyxxyz

Finalmente, a última parte da tripla deve ser colocada no final da cadeia estendida. Isso produz

xyxxyzyxxyzx

que é a mensagem descompactada.

Agora suponha que a versão compactada da mensagem fosse

xyxxyzy (5, 4, x) (0, 0, w) (8, 6, y)

Começaríamos descompactando a primeira tripla como antes para obter

xyxxyzyxxyzx (0, 0, w) (8, 6, y)

Então decodificamos a segunda tripla para obter

xyxxyzyxxyzxw (8, 6, y)

Note que a tripla (0, 0, w) foi usada porque o símbolo *w* ainda não havia aparecido na mensagem. Finalmente, decodificando a terceira tripla, teríamos a mensagem descompactada

xyxxyzyxxyzxwzyxxyzy

Para comprimir uma mensagem usando o LZ77, podemos inicialmente tomar um segmento inicial da mensagem e então procurar o maior segmento no padrão tomado que coincida com o restante da mensagem a ser comprimida. Este será o padrão referenciado na primeira tripla. As outras triplas são formadas por um processo similar.

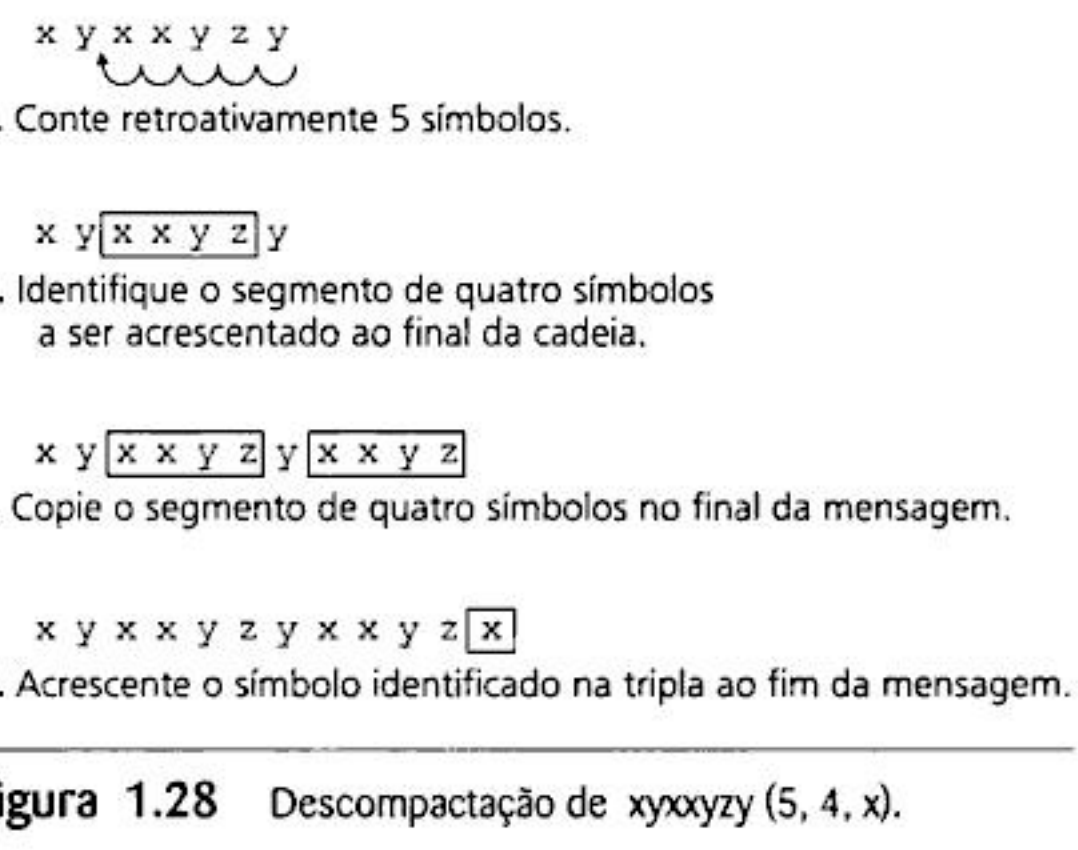



**Figura 1.28** Descompactação de xyxxyzy (5, 4, x).

Finalmente, você deve ter notado que os nossos exemplos não refletem muita compressão, uma vez que as triplas envolvidas representam segmentos pequenos. Quando aplicado a padrões longos de *bits*, contudo, é razoável que longos segmentos sejam representados por triplas — o que resulta em uma compressão de dados significativa.

## Compressão de imagens

Na Seção 1.4, vimos que os mapas de *bits* produzidos pelos digitalizadores atuais tendem a representar as imagens em um formato de três *bytes* por *pixel*, o que leva a mapas de *bits* grandes e intratáveis. Muitos esquemas de compressão têm sido desenvolvidos para reduzir esta necessidade de armazenamento. Um sistema conhecido como GIF (abreviatura de Graphic Interchange Format e pronunciado "Guif" por uns e "Jif" por outros) foi desenvolvido pela CompuServe. Ele aborda o problema reduzindo o número de cores que podem ser atribuídas a um *pixel* a apenas 256; assim, o valor de cada *pixel* é representado em um único *byte* em vez de três. Cada um dos 256 valores possíveis é associado a uma combinação de vermelho, verde e azul por meio de uma tabela conhecida como palheta. Ao mudar uma palheta associada a uma imagem, podemos mudar as cores que aparecem nesta.

A uma das cores na palheta GIF normalmente é atribuído o valor "transparente", o que significa que o fundo é mostrado através de qualquer região que tenha essa "cor". Essa opção, combinada com a relativa simplicidade do sistema GIF, faz dele uma escolha lógica em jogos de ação computadorizados nos quais múltiplas imagens se movem na tela.

Outro sistema de compressão de imagens coloridas é o JPEG (pronuncia-se j-peg, em português, ou *jei-pig* em inglês). É um padrão desenvolvido pelo Join Photographic Expert Group (daí o nome do padrão) um grupo da ISO. O JPEG demonstrou ser um padrão efetivo para representar fotografias coloridas. Com efeito, ele é adotado pelos fabricantes das câmeras digitais atuais e promete um grande impacto no contexto das imagens digitais nos anos vindouros.

O padrão JPEG na verdade engloba vários métodos para representar imagens, cada qual com seus objetivos. Por exemplo, nas situações em que se exige o máximo em precisão, o JPEG provê o modo "menos perda", cujo nome implica que nenhuma informação é perdida no processo de codificação da imagem. Nesse modo, o espaço é economizado ao armazenar a diferença entre *pixels* consecutivos em vez das intensidades dos *pixels* — a teoria é que, na maioria dos casos, a quantidade pela qual *pixels* adjacentes diferem pode ser representada por menos padrões de *bits* do que os usados para representar os valores dos *pixels*. (Este é um exemplo de codificação relativa.) Essas diferenças são então codificadas usando códigos de tamanho variável para reduzir o espaço de armazenamento.

Infelizmente, o uso do modo "menos perda" do JPEG não leva a mapas de *bits* com tamanhos manejáveis pela tecnologia atual, e por isso raramente é usado. Em seu lugar, a maioria das aplicações

usa o padrão JPEG básico, que reduz o tamanho de uma imagem codificada distinguindo o brilho da cor de cada *pixel*.

O propósito de distinguir entre luminosidade e cor é que o olho humano é mais sensível a mudança de brilho do que de cor. Considere, por exemplo, dois fundos azuis idênticos, exceto em que um deles contenha um pequeno ponto brilhante, enquanto o outro, um pequeno ponto verde com o mesmo brilho do fundo azul. Seu olho encontraria mais prontamente o ponto brilhante do que o verde. O padrão básico do JPEG tira vantagem desse fenômeno, codificando cada componente de brilho, mas dividindo a imagem em blocos de quatro *pixels* e registrando apenas a cor média de cada bloco. Assim, a representação final preserva mudanças repentinas no brilho, mas tende a obscurecer mudanças repentinas de cor. O benefício é que cada bloco de quatro *pixels* é representado por apenas seis valores (quatro valores de brilho e dois de cor) em vez de 12 valores que seriam necessários em um sistema de três *bytes* por *pixel*.

Economiza-se mais espaço ao registrar dados que indicam como os vários componentes de brilho e cor mudam, em vez de seus valores. Aqui, como no modo "menos perda" do JPEG, a motivação é que à medida que a imagem vai sendo percorrida, as diferenças entre valores de *pixels* próximos podem ser registradas usando menos *bits* do que seria necessário se os próprios valores fossem registrados. (Na realidade, essas diferenças são codificadas aplicando-se uma técnica matemática conhecida como transformação discreta de cossenos, cujos detalhes não nos interessam aqui.) O padrão final de *bits* é comprimido a seguir, aplicando um sistema de codificação de comprimento variável.

Em resumo, o padrão básico do JPEG pode codificar imagens coloridas de alta qualidade usando padrões de *bits* que estão na faixa de um vigésimo do tamanho necessário ao formato de três *bytes* por *pixel*, usado pela maioria dos digitalizadores. Isso explica por que ele está crescendo em popularidade. Outras técnicas, contudo, têm vantagens em certas aplicações. GIF, por exemplo, faz um serviço melhor ao representar imagens que consistem em blocos de cores uniformes com bordas estreitas (como os desenhos coloridos) do que o JPEG.

Para encerrar, devemos notar que a pesquisa em compressão de dados representa um campo largo e ativo. Discutimos apenas duas de muitas técnicas para comprimir imagens. Além disso, existem muitas estratégias para comprimir áudio e vídeo. Por exemplo, técnicas similares às usadas no padrão básico do JPEG têm sido adotadas pelo Motion Picture Expert Group (MPEG) da ISO com a finalidade de estabelecer padrões para codificar (comprimir) filmes de cinema. A idéia subjacente é iniciar a seqüência de quadros com uma imagem similar ao padrão básico do JPEG e então representar o resto da seqüência usando técnicas de codificação relativa.

## QUESTÕES/EXERCÍCIOS

1. Abaixo está uma mensagem que foi comprimida usando o LZ77. Qual é a cadeia descompactada?
   `101101011 (7,5,0) (12,10,1) (18,13,0)`
2. Embora não tenhamos nos concentrado no algoritmo para comprimir dados baseado no LZ77, tente comprimir a mensagem
   `bbabbbaababaababaababaaa`
3. Nesta seção, afirmamos que o GIF é melhor do que o JPEG para representar desenhos coloridos. Explique por quê.
4. No máximo, quantos *bytes* seriam necessários para representar uma imagem de 1024 por 1024 *pixels* usando GIF? E quantos se for usado o padrão básico do JPEG?
5. Qual característica do olho humano foi explorada pelo padrão básico do JPEG?
7. Identifique um fenômeno complicador que é comum quando se codifica informação numérica, imagens e sons com padrões de *bits*.

## 1.9 Erros de comunicação

Quando há troca de informação de um lado para outro entre as partes de um computador, ou transmissão da Terra para a Lua e vice-versa, ou quando essa informação é simplesmente armazenada, existe uma possibilidade de que, ao final desse processo, os padrões de *bits* recuperados não sejam exatamente idênticos aos originais. Partículas de sujeira ou de gordura sobre a superfície magnética do meio de armazenamento, ou um circuito com falhas de funcionamento, podem causar erros de gravação ou de leitura. Além disso, no caso de algumas tecnologias, a radiação de fundo pode alterar padrões armazenados na memória principal de um computador.

Para solucionar tais problemas, foram desenvolvidas diversas técnicas de codificação para permitir a detecção e até mesmo a correção de tais erros. Atualmente, por serem extensivamente empregadas na estrutura interna dos componentes de um sistema computacional, essas técnicas acabam passando despercebidas aos usuários. Todavia, a sua presença é muito importante e representa uma contribuição significativa para a pesquisa científica. É razoável, portanto, a investigação de algumas dessas técnicas, nas quais se apóia a confiabilidade dos equipamentos modernos.

### *Bits* de paridade

Um método simples para a detecção de erros se baseia no princípio de que, se cada padrão correto de *bits* tiver um número ímpar de 1s, então, se for encontrado um padrão com um número par de 1s, isto indicará a presença de um erro.

Para utilizar este princípio, precisamos de um sistema em que cada padrão correto contenha sempre um número ímpar de 1s. Isto é fácil de obter, acrescentando-se primeiramente um *bit*, denominado ***bit* de paridade**, para cada padrão de um sistema de codificação já disponível (em geral, à esquerda do *bit* mais significativo). (Assim, o código ASCII de oito *bits* será convertido em um de nove *bits*, ou um padrão de dezesseis *bits*, na notação de complemento de dois, se tornará um padrão de dezessete.) Nos dois casos, atribuímos os valores 1 ou 0 para este novo *bit* de tal forma que o padrão resultante tenha sempre um número ímpar de 1s. Como a Figura 1.29 mostra, o código ASCII para a letra A é 101000001 (*bit* de paridade 1), e o código ASCII para a letra F é 001000110 (*bit* de paridade 0). Embora o padrão de oito *bits* para a letra A tenha um número par de 1s e aquele para a letra F tenha um número ímpar de 1s, os dois padrões de nove *bits* apresentam um número ímpar de 1s. Uma vez que o nosso sistema de codificação tenha sido modificado de acordo com o esquema descrito anteriormente, a detecção de um padrão com um número par de 1s será sintoma da presença de um erro no padrão de *bits* em questão.

Este sistema de paridade que acabamos de descrever é conhecido como **paridade ímpar**, pois cada padrão correto contém um número ímpar de *bits*. Uma alternativa é a utilização da **paridade par**, na qual cada padrão é construído de forma que contenha um número par de 1s. Nesse caso, um erro é sinalizado pela detecção de um padrão de *bits* que contenha um número ímpar de 1s.

Atualmente, não é raro encontrarmos *bits* de paridade utilizados na memória principal de um computador. Embora tais computadores sejam tidos por seus usuários como máquinas cujas posições de memória têm oito *bits* cada uma, na realidade, essas memórias podem ter células de nove *bits*, um

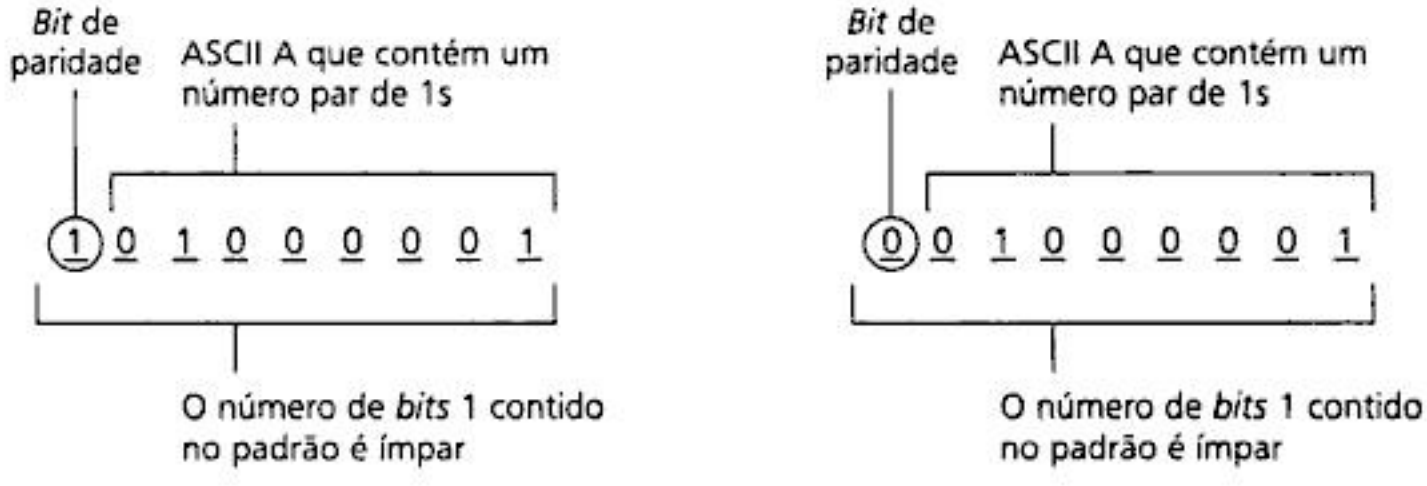



**Figura 1.29** Os códigos ASCII das letras A e F, adaptados para paridade ímpar.

dos quais opera como *bit* de paridade. Todas as vezes que um padrão de oito *bits* é fornecido ao circuito de memória para armazenamento, este lhe acrescenta um *bit* de paridade adequado, armazenando o resultado como padrão de nove *bits*. Mais tarde, quando esse padrão for recuperado, os circuitos de leitura irão conferir a paridade do padrão de nove *bits*. Se não for indicado um erro, o *bit* de paridade será removido, e o padrão resultante voltará a ser de oito *bits*. Caso contrário, a memória devolverá, juntamente com os oito *bits*, um aviso de que o padrão recuperado talvez não seja idêntico ao originalmente apresentado à memória para ser armazenado.

Longos padrões de *bits* freqüentemente são acompanhados por um conjunto de *bits* de paridade, que formam um **byte de verificação**. Cada *bit* desse *byte* é um *bit* de paridade associado a um conjunto de *bits* espalhados ao longo do padrão. Por exemplo, um *bit* de paridade pode estar associado ao conjunto de todos os oitavos *bits* contados a partir do primeiro *bit* do padrão, enquanto um outro talvez esteja associado a todos os oitavos *bits*, contados a partir do segundo *bit*. Desta forma, provavelmente será mais fácil encontrar vários erros, concentrados em uma área do padrão original, dado que estarão no escopo de vários *bits* de paridade. Algumas variações deste conceito de *byte* de verificação para a detecção de erros dão origem a esquemas de detecção de erros, conhecidos como verificação de soma (*checksum*) e como códigos de redundância cíclica (*cyclic redundancy codes* — CRC).

## Códigos de correção de erros

Embora seja possível detectar a presença de um erro com um único *bit* de paridade, ele não fornece informações suficientes para corrigir o erro. Muitas pessoas se surpreendem ao saber que é possível projetar **códigos de correção** de forma que eventuais erros sejam não só descobertos como também corrigidos. Afinal de contas, a intuição diz que não podemos corrigir os erros de uma mensagem incorretamente recebida a menos que já tenhamos conhecimento da informação nela contida. Contudo, a Figura 1.30 apresenta um código simples que exibe essa propriedade corretiva.

Para entender como este código funciona, definimos primeiro a **distância de Hamming** (assim denominada em homenagem a R. W. Hamming, pioneiro na busca de códigos de correção de erros, motivado pela falta de confiabilidade das primeiras máquinas dos anos 1940) entre dois padrões de *bits* como o número de *bits* diferentes existente entre eles. Por exemplo, a distância de Hamming entre A e B no código da Figura 1.30 é quatro e entre B e C, três. A principal característica do código é que dois padrões quaisquer estão separados por uma distância de Hamming de pelo menos três. Se um único *bit* for alterado em conseqüência de um mau funcionamento de um dispositivo, o erro será detectado, uma vez que o resultado obtido não será um padrão legal. (Devemos alterar pelo menos três *bits* de qualquer padrão legal antes que seja aceito como outro padrão legal.)

Se tiver ocorrido somente um erro no padrão da Figura 1.30, será possível descobrir qual era o seu padrão original. De fato, o padrão modificado estará a uma distância de Hamming de um único *bit* da sua forma original, mas, no mínimo, a dois *bits* de quaisquer outros padrões legais. Para decodificar uma mensagem, simplesmente comparamos cada padrão recebido com os padrões da tabela de códigos, até encontrar um que esteja a uma distância de um *bit* do padrão recebido. Este será o símbolo correto, obtido por meio da decodificação. Por exemplo, suponha o padrão de *bits* 010100. Se o compararmos com os padrões da tabela de códigos, obteremos a tabela da Figura 1.31. Assim, concluiremos que o símbolo transmitido deve ter sido um D, por apresentar o padrão mais próximo do original.

| Símbolo | Código |
|---|---|
| A | 000000 |
| B | 001111 |
| C | 010011 |
| D | 011100 |
| E | 100110 |
| F | 101001 |
| G | 110101 |
| H | 111010 |

**Figura 1.30** Um código de correção de erros.

Percebe-se que, utilizando esta técnica com os códigos da Figura 1.30, é possível descobrir até dois erros por padrão e corrigir um deles. Se projetássemos o código de forma que cada padrão tivesse, pelo menos,

uma distância de Hamming de cinco *bits* dos demais padrões, seria possível descobrir até quatro erros por padrão e corrigir dois deles. Logicamente, projetar códigos eficientes associados a distâncias de Hamming grandes não é uma tarefa fácil. De fato, constitui uma parte do ramo da matemática denominado teoria de codificação algébrica, que é uma subárea da álgebra linear e da teoria de matrizes.

As técnicas de correção de erros são usadas extensivamente para aumentar a confiabilidade dos equipamentos de computação. Por exemplo, elas freqüentemente são usadas nos dispositivos de discos magnéticos de alta capacidade para reduzir a possibilidade de um defeito na superfície magnética corromper os dados. Além disso, a principal distinção entre o formato original dos CDs usados como discos de áudio e o formato mais recente, usado como armazenamento para dados de computador, é o grau de correção de erros envolvido. O formato CD-DA incorpora características de correção de erros que reduzem a taxa para apenas um erro em dois CDs. Isto é adequado para gravações de áudio, mas uma companhia que usa CDs para fornecer *software* a seus clientes consideraria que erros em 50% dos discos são intoleráveis. Assim, características adicionais de correção de erros são empregadas em CDs usados para armazenamento de dados, reduzindo a possibilidade de erros para 1 em 20.000 discos.

| Símbolo | Distância entre o padrão recebido e o símbolo em análise |
|---|---|
| A | 2 |
| B | 4 |
| C | 3 |
| D | (1) *menor distância* |
| E | 3 |
| F | 5 |
| G | 2 |
| H | 4 |

**Figura 1.31** Decodificação do padrão 010100 utilizando a tabela da Figura 1.30.

## QUESTÕES/EXERCÍCIOS

**1.** Os *bytes* seguintes foram originalmente codificados com paridade ímpar. Em quais deles é possível afirmar que existe algum erro?
a. `10101101` b. `10000001` c. `00000000` d. `11100000`
e. `11111111`

**2.** Nos *bytes* da questão 1, pode haver erros imperceptíveis? Explique sua resposta.

**3.** Suas respostas para as questões 1 e 2 seriam diferentes se a paridade utilizada fosse par e não ímpar?

**4.** Codifique as sentenças abaixo em ASCII, usando paridade ímpar, por meio da inserção de um *bit* de paridade como *bit* mais significativo de cada código dos caracteres:
a. *Where are you?*
b. *"How"? Cheryl asked.*
c. 2+3 = 5

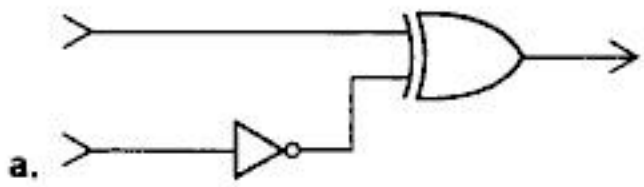
**5.** Usando o código de correção de erros apresentado na Figura 1.30, decodifique as seguintes mensagens:
a. `001111 100100 001100`
b. `010001 000000 001011`
c. `011010 110110 100000 011100`

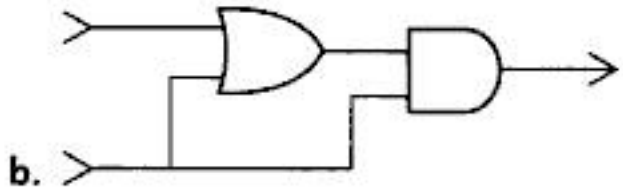
**6.** Construa códigos para os caracteres A, B, C e D usando padrões de *bits* de comprimento cinco, de modo que a distância de Hamming entre dois padrões quaisquer seja no mínimo três.

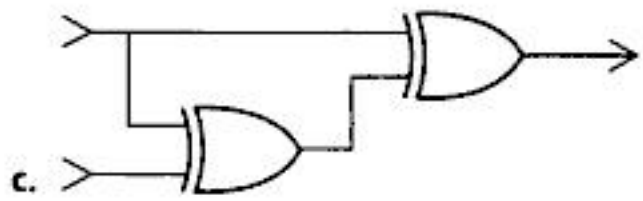
## *Problemas de revisão de capítulo*

(Os problemas marcados com asterisco se referem às seções opcionais.)

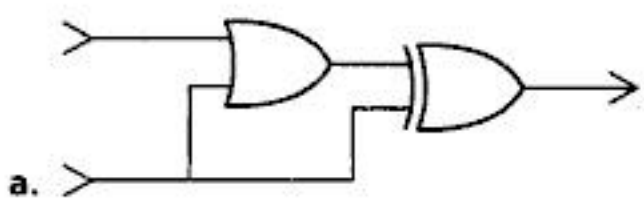
**1.** Determine a saída dos seguintes circuitos pressupondo que a entrada superior é 1 e a inferior é 0.

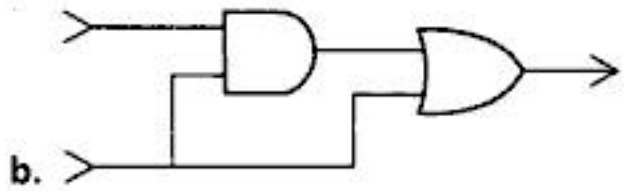


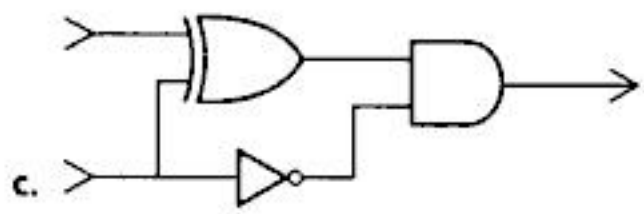



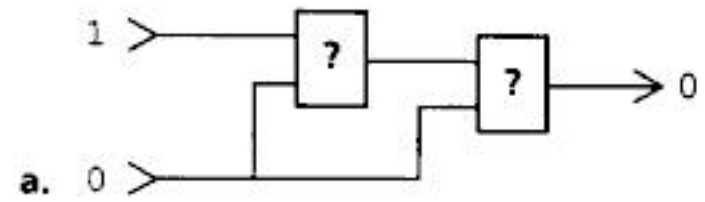

**2.** Nos seguintes circuitos, identifique a combinação de entradas que produzirá uma saída igual a 1.

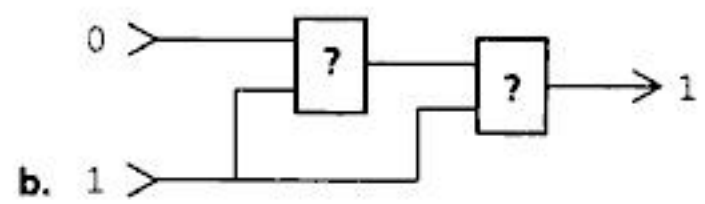


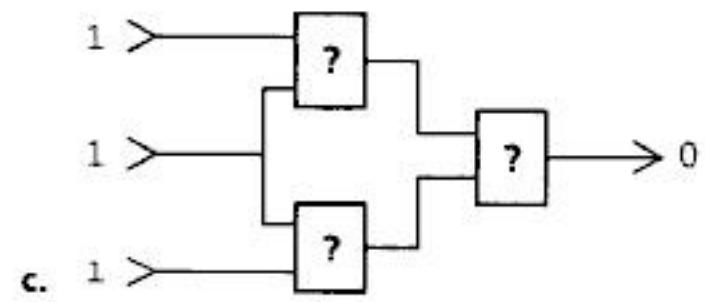



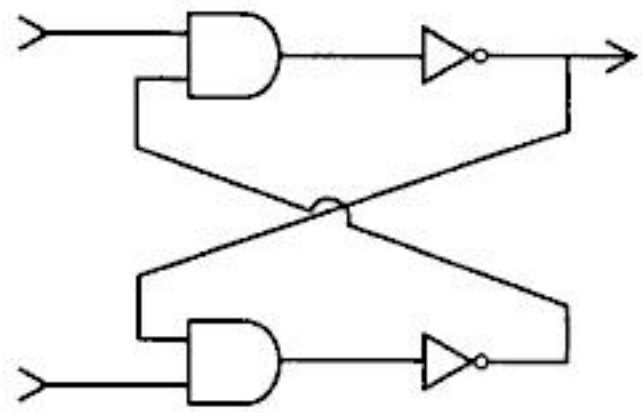

**3.** Em cada circuito abaixo, o retângulo representa o mesmo tipo de porta lógica. Baseado na informação dada de entrada e saída, identifique se a porta lógica envolvida é AND, OR ou XOR.




**4.** Suponha que as duas entradas do circuito abaixo sejam 1. Descreva o que ocorrerá se a entrada superior for alterada temporariamente para 0. Ou se a entrada inferior for alterada temporariamente para 0. Redesenhe o circuito usando portas NAND.



**5.** A seguinte tabela representa, em notação hexadecimal, os endereços e conteúdos de algumas posições da memória principal de um computador. A partir desta configuração inicial de memória, execute a seqüência de instruções indicada abaixo, e registre os resultados finais de cada posição de memória:

| Endereço | Conteúdo |
|---|---|
| 00 | AB |
| 01 | 53 |
| 02 | D6 |
| 03 | 02 |

Passo 1. Transfira o conteúdo da posição de memória de endereço 03 para a posição de endereço 00.
Passo 2. Coloque o valor 01 na posição de memória de endereço 02.
Passo 3. Transfira o valor armazenado no endereço 01 para a posição de endereço 03.

**6.** Quantas posições existem na memória principal de um computador se o endereço de qualquer dessas posições pode ser representado por três dígitos hexadecimais?

**7.** Que padrões de *bits* são representados pelas seguintes notações hexadecimais?
a. BC b. 67 c. 9A d. 10
e. 3F

**8.** Qual é o valor do *bit* mais significativo dos padrões de *bits* representados pelas seguintes notações hexadecimais?
a. FF b. 7F c. 8F d. 1F

**9.** Expresse os seguintes padrões de *bits* na notação hexadecimal:
a. `101010101010`
b. `110010110111`
c. `000011101011`

**10.** Suponha que uma tela de monitor apresente 24 linhas e cada uma contenha 80 caracteres. Se o conteúdo da tela fosse armazenado na memória, por meio da codificação dos caracteres em ASCII (um caractere por *byte*), quantos *bytes* de memória seriam necessários para apresentar a imagem completa?

**11.** Suponha que uma figura seja representada na tela de um monitor através de uma matriz retangular de 1024 colunas e 768 linhas de *pixels*. Se forem necessários oito *bits* para codificar a cor e a intensidade de cada *pixel*, quantos *bytes* de memória serão precisos para guardar a figura toda?

**12.** a. Identifique duas vantagens que a memória principal apresenta em relação ao armazenamento em disco.

b. Identifique duas vantagens que o armazenamento em disco apresenta em relação à memória principal.

**13.** Suponha que se deseje utilizar um computador pessoal para escrever um ensaio com 40 páginas, digitadas em espaço duplo. O computador tem uma unidade de disco com disquetes de 3¼ polegadas (1 polegada = 2,54 cm) com uma capacidade de 1,44 MB por disquete. Seu ensaio completo caberá em um desses discos? Neste caso, quantos ensaios podem ser armazenados em um disco? Se não, quantos discos serão necessários?

**14.** Suponha que apenas 100 MB dos 10 GB do disco rígido de seu computador pessoal estejam livres e que você esteja a ponto de substituí-lo por um outro de 30 GB. Você deseja guardar temporariamente todas as informações armazenadas no antigo disco rígido em disquetes de 3¼" enquanto efetua a conversão. É prático proceder dessa maneira? Explique a sua resposta.

**15.** Se cada setor de um disco comporta 512 *bytes*, quantos setores serão necessários para conter uma página de texto, digitada em espaço duplo? Cada símbolo ocupa um *byte*.

**16.** Um disquete comum de 3¼" tem uma capacidade de 1,44MB. Relacione essa capacidade com o tamanho de um romance de 400 páginas, sendo que cada página contém 3500 caracteres.

**17.** Se um disquete, com 16 setores por trilha e 512 *bytes* por setor, gira à razão de 300 revoluções por minuto, a que taxa aproximada, expressa em *bytes* por segundo, os dados passam pelo cabeçote de leitura/gravação?

**18.** Se um microcomputador, utilizando o disquete do Problema 17, executar dez instruções por microssegundo (milionésimo de segundo), quantas instruções poderá executar durante o período transcorrido entre as passagens de *bytes* consecutivos pelo cabeçote de leitura /gravação?

**19.** Se um disquete girar a 300 rotações por minuto e o computador executar dez instruções por microssegundo (milionésimo de segundo), quantas instruções ele poderá executar durante o tempo de latência do disco?

**20.** Compare o tempo de latência de um disquete comum, como o do Problema 19, a um dispositivo de disco rígido que gire a 60 rotações por segundo.

**21.** Qual é o tempo médio de acesso de um disco rígido de 60 revoluções por segundo, com um tempo de busca igual a 10 milissegundos?

**22.** Suponha que um digitador, trabalhando continuamente dia e noite digite a uma velocidade constante de 60 palavras por minuto. Quanto tempo ele levará para preencher um CD cuja capacidade seja de 640 MB? Pressuponha que uma palavra seja formada por cinco caracteres e que cada um ocupe um *byte*.

**23.** Decodifique a mensagem abaixo, escrita em ASCII:

```
01010111  01101000  01100001
01110100  00100000  01100100
01101111  01100101  01110011
00100000  01101001  01110100
00100000  01110011  0110001
01111001  00111111
```

**24.** A seguinte mensagem, inicialmente codificada em ASCII com caracteres de um *byte*, foi expressa abaixo, em notação hexadecimal. Qual é a mensagem?

6865786164656369 6D616C

**25.** Codifique as seguintes sentenças, em código ASCII com caracteres de um *byte*:

a. $^{100}/_{5} = 20$
b. Ser ou não ser?
c. O custo total é de R$ 7,25.

**26.** Expresse as respostas do Problema 25 em notação hexadecimal.

**27.** Liste as representações binárias dos inteiros de 6 a 16.

**28.** a. Escreva o número 13, codificando em ASCII os dígitos 1 e 3.
b. Escreva o número 13, usando codificação binária.

**29.** Que números exibem, na sua representação binária, somente um dos seus *bits* igual a 1? Liste a representação binária dos seis menores números que apresentam esta propriedade.

***30.** Expresse os textos abaixo em código ASCII, utilizando um *byte* por símbolo. Use o *bit* mais significativo de cada *byte* como *bit* de paridade (empregue a paridade ímpar).

a. $^{100}/_{5} = 20$
b. Ser ou não ser?
c. O custo total é de R$ 7,25.

***31.** A seguinte mensagem foi transmitida originalmente com paridade ímpar em cada uma das suas cadeias menores de *bits*. Em quais dessas pequenas cadeias é possível garantir a presença de erros?

```
11011 01011 10110 00000 11111 10101
10001 00100 01110
```

***32.** Seja um código de 24 *bits* gerado representando-se cada símbolo como três cópias sucessivas de sua representação ASCII (por exemplo, o símbolo A seria representado pela cadeia de *bits* 010000010100000101000001). Analise este novo código quanto a suas propriedades referentes à correção de erros.

***33.** Utilizando o código de correção de erros descrito na Figura 1.30, decodifique as seguintes palavras:

a. 111010 110110
b. 101000 100110 001100
c. 011101 000110 000000 010100
d. 010010 001000 001110 101111 000000 110111 100110
e. 010011 000000 101001 100110

***34.** Converta as seguintes representações binárias em sua forma decimal equivalente:

| | | |
|---|---|---|
| a. 111 | b. 0001 | c. 11101 |
| d. 10001 | e. 10111 | f. 000000 |
| g. 100 | h. 1000 | i. 10000 |
| j. 11001 | k. 11010 | l. 11011 |

***35.** Converta as seguintes representações decimais em sua forma binária equivalente:

a. 7 b. 12 c. 16 d. 15 e. 33

***36.** Converta na forma decimal as seguintes representações, em excesso de 16:

| | | |
|---|---|---|
| a. 10000 | b. 10011 | c. 01101 |
| d. 01111 | e. 10111 | |

***37.** Converta na notação de excesso de quatro os seguintes números decimais:
a. 0 b. 3 c. –3
d. –1 e. 1

***38.** Converta em decimais as seguintes representações em complemento de dois:
a. `10000` b. `10011` c. `01101`
d. `01111` e. `10111`

***39.** Converta as seguintes representações decimais na notação de complemento de dois com padrões de sete *bits*:
a. 12 b. –12 c. –1
d. 0 e. 8

***40.** Execute as seguintes adições, pressupondo que as cadeias de *bits* representam valores na notação de complemento de dois. Identifique os casos em que a resposta esteja incorreta devido à ocorrência de estouro:

a. `00101` + `01000`
b. `01111` + `00001`
c. `11111` + `00001`
d. `10111` + `11010`
e. `00111` + `00111`
f. `00111` + `01100`
g. `11111` + `11111`
h. `01010` + `10101`
i. `01000` + `01000`
j. `01010` + `00011`

***41.** Resolva as contas abaixo, convertendo inicialmente os valores na notação de complemento de dois (usando padrões de 5 *bits*), substituindo qualquer operação de subtração por uma equivalente de adição e executando tal adição. Confira o resultado reconvertendo-o na notação decimal. (Mantenha-se atento à ocorrência de estouro).

a. 7 + 1
b. 7 − 1
c. 12 − 4
d. 8 − 7
e. 12 + 4
f. 4 + 11

***42.** Converta em decimais as seguintes representações binárias:
a. 11,001 b. 100,1101 c. ,0101
d. 1,0 e. 10,01

***43.** Expresse em notação binária os seguintes valores numéricos:
a. $5^3/_4$ b. $^1/_{16}$ c. $7^7/_8$
d. $1^1/_4$ e. $6^5/_8$

***44.** Decodifique os seguintes padrões de *bits* utilizando a notação de vírgula flutuante discutida na Seção 1.7:
a. `01011100` b. `11001000`
c. `00101010` d. `10111001`

***45.** Codifique os seguintes valores utilizando a notação de vírgula flutuante discutida na Seção 1.7. Aponte os casos em que ocorrerem erros de truncamento:
a. $^1/_2$ b. $7^1/_2$ c. $-3^3/_4$
d. $^3/_{32}$ e. $^{31}/_{32}$

***46.** Qual a melhor aproximação da raiz quadrada de 2 que possa ser expressa na notação de vírgula flutuante descrita na Seção 1.7? Que resultado se obterá caso tal valor aproximado seja elevado ao quadrado por um computador que utilize a notação de vírgula flutuante?

***47.** Qual a melhor aproximação, para o valor *um décimo*, que pode ser codificada utilizando a notação de vírgula flutuante descrita na Seção 1.7?

***48.** Explique como podem ocorrer erros quando medidas que usam o sistema métrico são gravadas na notação de vírgula flutuante. Por exemplo, o que acontece se 110 cm for gravado na unidade metro?

***49.** Usando o formato de vírgula flutuante com oito *bits* descrito na Seção 1.7, qual seria o resultado da seguinte soma $^1/_8 + ^1/_8 + ^1/_8 + 2^1/_2$, quando resolvida da esquerda para a direita? E quando resolvida da direita para a esquerda?

***50.** Que resposta seria obtida para as seguintes contas de adição executadas por uma máquina usando o formato de vírgula flutuante com 8 *bits* descrito na Seção 1.7?
a. $1\,^1/_2 + ^3/_{16} =$
b. $3\,^1/_4 + 1\,^1/_8 =$
c. $2\,^1/_4 + 1\,^1/_8 =$

***51.** Nas seguintes contas de adição, interprete os padrões de *bits* utilizando o formato de vírgula flutuante com 8 *bits* discutido na Seção 1.7, some os valores representados e codifique a resposta no mesmo formato de vírgula flutuante. Identifique os casos em que ocorrerem erros de truncamento.

a. `01011100` + `01101000`
b. `01101010` + `00111000`

c.
```
  01111000
+ 00011000
```

d.
```
  01011000
+ 01011000
```

***52.** Um destes padrões de *bits* (01011 ou 11011) representa um valor codificado na notação de excesso de 16 e o outro, o mesmo valor, codificado na notação de complemento de dois.
   a. O que é possível determinar acerca deste valor comum?
   b. Usando o mesmo número de *bits* para as duas representações, qual é a relação existente entre o padrão que representa um valor na notação de complemento de dois e aquele que o representa em notação de excesso?

***53.** Os três padrões de *bits* 01101000, 10000010 e 00000010 são representações do mesmo valor na notação de complemento de dois, na de excesso e no formato de vírgula flutuante com oito *bits*, discutidos na Seção 1.7, mas não necessariamente nesta ordem. Qual é o valor comum representado? Qual notação corresponde a cada padrão?

***54.** Nos casos a seguir, as diferentes cadeias de *bits* representam o mesmo valor, mas em diferentes sistemas, já discutidos, de codificação numérica. Identifique cada número representado e os sistemas de codificação associados a cada representação.
   a. `11111010 0011 1011`
   b. `11111101 01111101 11101100`
   c. `1010 0010 01101000`

***55.** Quais dos seguintes padrões de *bits* não são representações válidas, na notação de excesso de 16?
   a. `01001` b. `101` c. `010101`
   d. `00000` e. `1000` f. `000000`
   g. `1111`

***56.** Quais dos seguintes valores não podem ser representados com precisão no formato de vírgula flutuante introduzido na Seção 1.7?
   a. $6^1/_2$ b. 9 c. $^{13}/_{16}$
   d. $1^7/_{32}$ e. $^{15}/_{16}$

***57.** Duplicando de quatro para oito *bits* o comprimento das cadeias de *bits* utilizadas para representar inteiros em binário, como se modifica o valor do maior inteiro representável nessa notação? E na notação de complemento de dois?

***58.** Qual será a representação hexadecimal do endereço mais alto de uma memória de 4 MB, se cada posição de memória comportar um só *byte*?

***59.** Usando portas lógicas, projete um circuito com quatro entradas e uma só saída, de modo que esta seja 1 ou 0, dependendo de o padrão formado pelas suas quatro entradas ter paridade ímpar ou par, respectivamente.

***60.** A seguinte mensagem foi comprimida usando o LZ77. Faça a sua descompactação:
   abr (3, 1, c) (2, 1, d) (7, 3, a)
Por que a mensagem comprimida é maior do que a original?

***61.** A seguinte mensagem foi comprimida usando o LZ77. Faça a sua descompactação:
   0100101 (4, 3, 0) (8, 7, 1) (17, 9, 1) (8, 6, 1)

***62.** Aqui está parte de uma mensagem que foi comprimida usando o LZ77. Baseando-se na informação dada, qual é o tamanho da mensagem original?
   xyz (_, 3, y) (_, 6, z)

***63.** Escreva um conjunto de diretrizes que expliquem como comprimir uma mensagem usando o LZ77.

***64.** Escreva um conjunto de diretrizes que expliquem como descompactar uma cadeia usando o LZ77.

## *Questões sociais*

As seguintes questões procuram auxiliar o leitor no entendimento de alguns assuntos éticos, sociais e legais no campo da computação. O objetivo não é meramente o de fornecer respostas para tais questões; o leitor também deve justificá-las e verificar se as justificativas preservam sua consistência de uma questão para outra.

1. Suponha a ocorrência de um erro de truncamento em uma situação crítica, causando enormes danos e perda de vidas. Quem é o responsável, se houver: o projetista do *hardware*? o projetista do *software*? o programador que escreveu esta parte do programa? a pessoa que decidiu aplicar o *software* naquela situação em particular? E se o *software* já tivesse sido corrigido pela companhia que originalmente o desenvolveu, mas o *software* de atualização (*upgrade*) do programa não tivesse sido adquirido, tendo sido empregada a versão antiga nessa aplicação crítica? E se o *software* tiver sido pirateado?

2. É aceitável que um indivíduo ignore a possibilidade de erros de truncamento durante o desenvolvimento de suas aplicações?

3. Foi ético desenvolver *software* na década de 1970 usando apenas dois dígitos para representar o ano (com usar 76 para representar 1976) ignorando-se o fato de que o *software* se tornaria incorreto na virada do século? Atualmente, é ético usar três dígitos para representar o ano? (como 982 para 1982 e 015 para 2015). E se forem usados quatro dígitos?

4. Muitos defendem o ponto de vista de que a codificação dilui ou distorce a informação, uma vez que ela, em sua essência, força a quantificação da informação. Argumentam que codificar opiniões acerca de determinados assuntos por meio de questionários, em uma gradação de 1 a 5, é inerentemente um método que acaba por truncar a informação. Até que ponto a informação é quantificável? Podem ser quantificados os argumentos pró e contra acerca do local de instalação de uma fábrica de reciclagem de lixo? O debate sobre a energia nuclear e o "lixo" que produz é quantificável? É perigoso tomar decisões baseadas em médias e outras análises estatísticas? É ético que as agências de notícias informem o resultado das apurações das respostas aos questionários sem mencionar o teor exato das perguntas? É possível quantificar o valor de uma vida humana? É aceitável que uma companhia deixe de investir na melhoria de um produto, mesmo sabendo que tal investimento reduziria as possibilidades da ocorrência de morte relacionadas com o uso desse produto?

5. Com o desenvolvimento de câmeras digitais, a possibilidade de alterar ou produzir fotografias é colocada à disposição do público em geral. Que mudanças isso traz à sociedade? Quais as questões éticas e legais que poderão surgir?

6. Deve haver uma distinção nos direitos de coletar e disseminar dados em função de sua forma? Em outras palavras, o direito de coletar e disseminar fotografia, áudio e vídeo é o mesmo direito de coletar e disseminar texto?

7. Intencionalmente ou não, em geral, a reportagem de um jornalista reflete a sua opinião. Freqüentemente a mudança de algumas poucas palavras pode dar uma conotação positiva ou negativa a um relato. (Compare "A maioria dos observadores não acreditava que..." com "Uma parte significativa dos observadores concordava que..."). Existe diferença entre alterar uma história (deixando de lado alguns pontos ou selecionado cuidadosamente as palavras) e alterar uma fotografia?

8. Suponha que o uso de um sistema de compressão de dados resulte na perda de itens de informação sutis, mas significativos. Que responsabilidades podem ser questionadas? Como elas devem ser resolvidas?

## *Leituras adicionais*

Gibs, S. J., and D. C. Tsichritzis, *Multimidia Programming.* Reading, MA: Addison-Wesley, 1995.
Hamacher, V. C., Z. G. Vranesic, and S. G. Zaky. *Computer Organization*, 4th ed. New York: McGraw-Hill, 1996.
Kay, D. C. and J. R. Levine. *Graphics File Formats* 2nd ed. New York, McGraw-Hill, 1995.
Knuth, D. E. *The Art of Computer Programming I*, vol. 2, 3rd ed. Reading MA: Addison-Wesley Longman, 1998.
Miano, J. *Compressed Image File Formats.* New York: ACM Press, 1999.
Patterson, D. A., and J. L. Hennessy. *Computer Organization and Design*. 2nd ed. San Francisco: Morgan Kaufmann, 1997.
Sayood, K. *Introduction to Data Compression* 2nd ed. San Francisco: Morgan Kaufmann, 2000.

CAPÍTULO 2

# *Manipulação de dados*

No Capítulo 1, estudamos conceitos relativos ao armazenamento de dados e à memória de um computador. Além de armazenar dados, o computador deve ser capaz de manipulá-los de acordo com um algoritmo. Tal manipulação exige que a máquina seja dotada de mecanismos capazes de executar operações com dados e coordenar a seqüência das mesmas. Essas tarefas são executadas por um mecanismo denominado Unidade Central de Processamento (UCP). Esta unidade e os tópicos a ela relacionados são o objeto de estudo deste capítulo.



**Os asteriscos indicam sugestões de seções consideradas opcionais.*

## 2.1 Arquitetura de computadores

Os circuitos de um computador que executam operações com dados (como adição e subtração) não são diretamente conectados às células de armazenamento da memória principal do computador, ficando isolados em uma parte conhecida como **unidade central de processamento** ou **UCP** (*central processing unit*) ou, simplesmente, processador. Esta unidade consiste em duas partes: a **unidade aritmética e lógica**, que contém os circuitos que manipulam os dados, e a **unidade de controle**, que contém os circuitos que coordenam as atividades da máquina.

Para armazenamento temporário de informação, a UCP contém células ou **registradores**, que são semelhantes às posições da memória principal. Tais registradores podem ser classificados como de **propósito geral** ou **propósito específico**. Conheceremos alguns dos registradores de propósito específico na Seção 2.3. Por ora, estudaremos a função dos registradores de propósito geral.

Os registradores de propósito geral funcionam como posições temporárias de armazenamento para dados que estão sendo manipulados pela UCP. Esses registradores guardam os dados de entrada da unidade aritmética e lógica e proporcionam um local de armazenamento para os seus resultados. Para executar uma operação sobre dados guardados na memória principal, é responsabilidade da unidade de controle transferir os dados da memória para os registradores de propósito geral, informar a unidade aritmética e lógica, cujos registradores armazenam os dados, ativar o circuito apropriado contido nesta unidade e informá-la sobre qual registrador receberá o resultado.

Para poder transferir padrões de *bits* entre o processador de um computador e a memória principal, tais elementos são interconectados por um conjunto de fios chamado **via** (*bus*) (Figura 2.1). Através desta via, o processador pode extrair ou ler dados da memória principal e, para tanto, ele fornece o endereço da posição de memória correspondente juntamente com um comando de leitura. Do mesmo modo, o processador pode colocar ou escrever dados na memória, fornecendo, para isso, o endereço da posição de destino e os dados a serem armazenados, acompanhados de um sinal de gravação.

Com este mecanismo em mente, verificamos que executar uma operação de adição com dados armazenados na memória principal é mais do que efetuar uma mera execução desta operação. O processo envolve esforços combinados entre a unidade de controle, que coordena a transferência de informação entre os registradores e a memória principal, e a unidade aritmética e lógica, que efetua a operação de adição propriamente dita, quando for instruída para isso pela unidade de controle.

O processo completo da adição de dois números armazenados na memória poderia ser dividido em uma seqüência de passos, conforme ilustra a Figura 2.2. Em resumo, os dados devem ser transferidos da memória principal para o processador, onde os valores são somados, e o resultado deve ser armazenado em uma célula de memória.

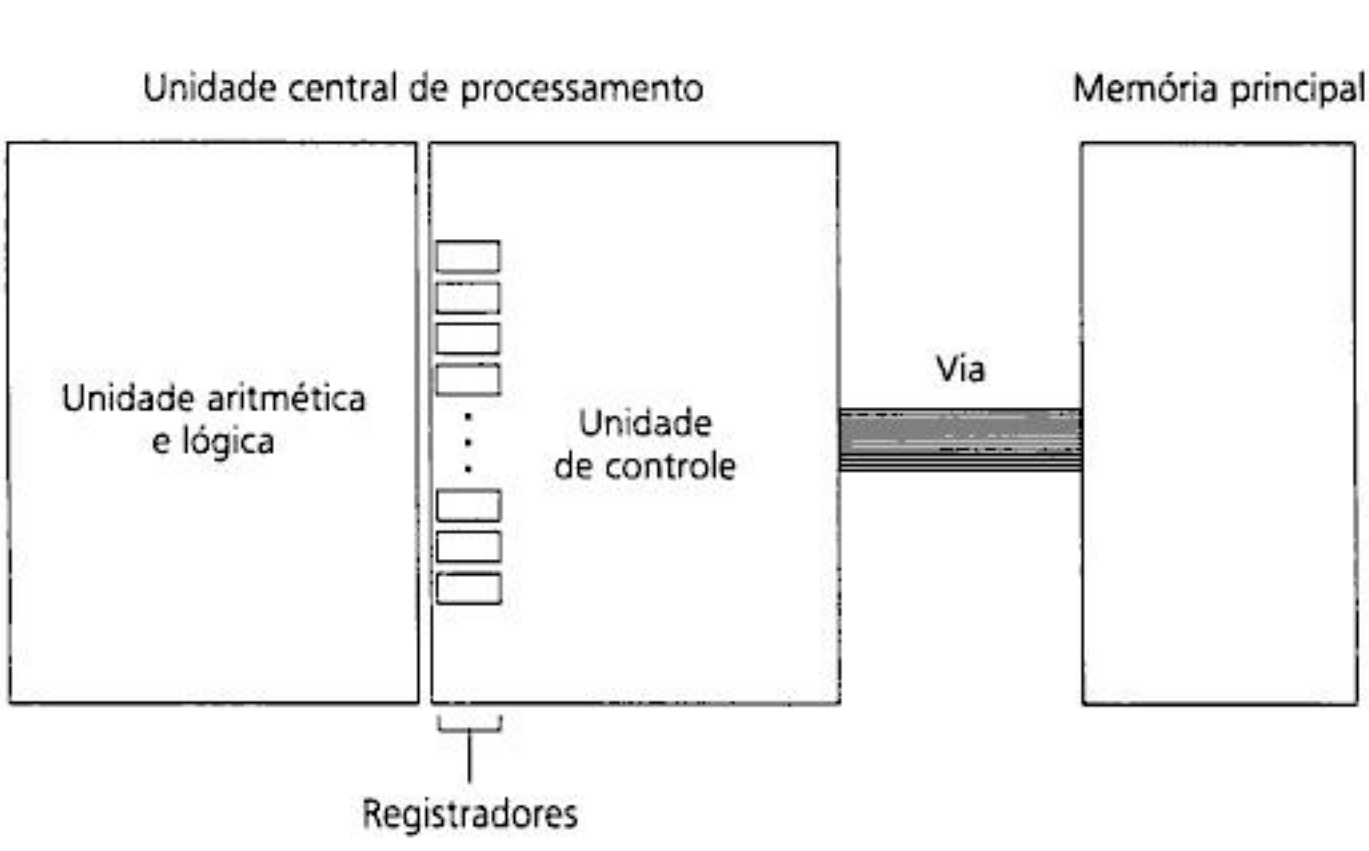



**Figura 2.1** UCP e memória principal ligadas por uma via.

Os primeiros computadores não eram conhecidos por sua flexibilidade — os passos que cada dispositivo executava estavam embutidos na unidade de controle, fazendo parte da máquina. Isto é semelhante a uma caixa de música que sempre toca a mesma melodia quando o que se deseja realmente é uma flexibilidade como a exibida

**Passo 1.** Obter da memória um dos valores a somar e guardá-lo em um registrador.
**Passo 2.** Obter da memória a outra parcela e armazená-la em outro registrador.
**Passo 3.** Acionar o circuito da adição, tendo os registradores utilizados nos passos 1 e 2 como entradas, e escolher outro registrador para armazenar o resultado.
**Passo 4.** Armazenar o resultado na memória.
**Passo 5.** Finalizar.

**Figura 2.2** Um roteiro para somar valores armazenados na memória.

por equipamentos de troca automática de CDs. Para aumentar a flexibilidade, alguns computadores eletrônicos da época foram projetados de maneira a se poder alterar a fiação da unidade de controle. Isso foi conseguido com um painel de controle semelhante às antigas centrais telefônicas, nas quais as extremidades dos cabos eram inseridas nos orifícios correspondentes à conexão desejada.

Uma inovação (creditada, talvez incorretamente, a John von Neumann) surgiu a partir da constatação de que o programa, da mesma forma que os dados, poderia ser codificado e armazenado na memória principal. Se a unidade de controle fosse projetada com capacidade de extrair o programa da memória, decodificar as instruções e executá-las, um programa de computador poderia ser modificado, simplesmente alterando o conteúdo da memória, em vez de refazer a fiação da unidade de controle.

## *Quem inventou o quê?*

Atribuir a um único indivíduo o crédito por uma invenção é sempre um empreendimento duvidoso. Thomas Edison é considerado o inventor da lâmpada incandescente, mas outros pesquisadores estavam desenvolvendo lâmpadas similares e, até certo ponto, Edison teve sorte de ser o primeiro a obter a patente. Os irmãos Wright são considerados os inventores do avião, mas se beneficiaram das pesquisas de outros e, em algum grau, foram influenciados por Leonardo da Vinci, que brincava com a idéia de máquinas voadoras já no século XVII. Entretanto, os projetos de Leonardo eram aparentemente baseados em idéias mais antigas. Obviamente, nesses casos, o inventor designado ainda tem reivindicações legítimas quanto ao crédito concedido. Em outros casos, a História parece ter conferido o crédito inapropriadamente – um exemplo é o conceito de programa armazenado. Sem dúvida, John von Neumann foi um cientista brilhante que merece crédito por muitas contribuições. Contudo, a contribuição pela qual a História popular escolheu dar-lhe o crédito, o conceito de programa armazenado, foi aparentemente desenvolvida por pesquisadores liderados por J. P. Eckert, da Moore School of Electrical Engineering, University of Pennsylvania. John von Neumann foi meramente o primeiro a publicar um trabalho sobre a idéia e, assim, os eruditos da computação o selecionaram como inventor.

O **conceito de programa armazenado** tornou-se a abordagem padrão usada atualmente — na realidade, tão padronizada que parece óbvia. O que originalmente a tornava difícil é que todo mundo pensava nos programas e dados como entidades diferentes. Os dados eram armazenados na memória; os programas eram parte da unidade de controle. O resultado era um exemplo primordial de não conseguir perceber a floresta por estar observando as árvores. É fácil cair e ficar preso nessas valetas, e o desenvolvimento da ciência da computação pode muito bem permanecer em muitas delas sem que nos apercebamos disto. Sem dúvida, uma parte empolgante da ciência é que novas percepções estão constantemente abrindo as portas para novas teorias e novas aplicações.

### *QUESTÕES/EXERCÍCIOS*

1. Qual a seqüência de eventos que o computador necessita para copiar o conteúdo de uma para outra posição de memória?
2. Quais são as ordens que o processador deve dirigir aos circuitos da memória principal para que seja guardado um número em uma certa posição de memória?
3. O armazenamento em massa, a memória principal e os registradores de propósito geral são três sistemas de armazenamento. Qual é a diferença quanto à maneira como são usados?

# 2.2 Linguagem de máquina

Para aplicar o conceito de programa armazenado, os processadores são projetados para reconhecer as instruções codificadas como padrões de *bits*. A coleção de instruções, juntamente com o sistema de codificação, é denominada **linguagem de máquina**. Uma instrução expressa nessa linguagem é chamada instrução em nível de máquina, ou mas comumente uma **instrução de máquina**.

## O repertório de instruções

É surpreendente que a lista de instruções de máquina que uma UCP comum deve decodificar e executar é bem pequena. Sem dúvida, um dos aspectos fascinantes da ciência da computação é que se uma máquina pode executar certas tarefas elementares, mas bem escolhidas, ao adicionar novos recursos a ela, isso não aumenta as suas capacidades teóricas. Em outras palavras, além de um certo ponto, as novas características podem aumentar certas facilidades, como a conveniência, mas não acrescentam coisa alguma às habilidades básicas da máquina. A decisão em relação ao aproveitamento desse fato tem levado a duas filosofias de projeto para a UCP. Uma delas é que a UCP deve executar um conjunto mínimo de instruções de máquina. A abordagem leva ao chamado **computador com conjunto mínimo de instruções** (*reduced instruction set computer* — RISC). O argumento em favor da arquitetura RISC é que essas máquinas são eficientes e rápidas. No entanto, argumenta-se a favor de UCPs com capacidade para executar muitas instruções complexas, ainda que muitas delas sejam tecnicamente redundantes. O resultado dessa abordagem é conhecido como **computador com conjunto de instruções complexas** (*complex instruction set computer* — CISC). O argumento em favor da arquitetura CISC é que a UCP mais complexa é mais fácil de programar, uma vez que uma única instrução é capaz de desempenhar uma tarefa que necessitaria uma seqüência de múltiplas instruções em um projeto RISC.

Os processadores CISC e RISC estão disponíveis comercialmente. A série Pentium de processadores, desenvolvida pela Intel, é exemplo de arquitetura CISC; as séries de processadores PowerPC, desenvolvidas pela Apple Computer, IBM e Motorola são exemplos de arquitetura RISC. Para facilitar a apresentação, enfocaremos a abordagem RISC neste capítulo.

Ao se discutir as instruções no repertório de uma máquina, é útil reconhecer que elas podem ser classificadas em três categorias: (1) o grupo de transferência de dados, (2) o grupo aritmético/lógico e (3) o grupo de controle.

**Transferência de dados** O primeiro grupo consiste em instruções que promovem a movimentação de dados de um local para outro. Os passos 1, 2 e 4 da Figura 2.2 pertencem a esta categoria. Como acontece na memória principal, não é comum os dados que estão sendo transferidos de um local para outro serem apagados no local original. Uma instrução de transferência está mais associada à idéia da cópia dos dados do que à da mudança física de posição. Logo, os termos *transferência* ou *mudança* geralmente empregados na verdade não são muito apropriados; termos como *cópia* ou *clone* descreveriam melhor tal operação.

### *Memória* cache *(escondida)*

É instrutivo considerar os registradores de propósito geral da UCP como recursos globais de memória de um computador. Os registradores são usados para manter os dados imediatamente aplicáveis à manipulação do momento; a memória principal é usada para manter os dados que serão necessários em um futuro próximo, e o armazenamento em massa, para manter os dados que dificilmente serão necessários no futuro próximo. Muitas máquinas são projetadas com um nível adicional de memória, chamada **memória *cache***, uma porção (às vezes vários KB) de memória de alta velocidade com tempo de resposta comparável ao dos registradores da UCP, normalmente localizada no próprio processador. Nessa área especial de memória, a máquina procura manter uma cópia de parte da memória principal que é de interesse corrente. Com isso, as transferências de dados que normalmente seriam entre registradores e memória principal são feitas entre os registradores e a memória *cache*. Quaisquer alterações são coletivamente transferidas para a memória em um momento mais oportuno. O resultado é uma máquina cujo ciclo pode ser executado mais rapidamente, o que aumenta a velocidade de execução das tarefas.

Ainda com relação à terminologia, deveríamos mencionar a existência de termos específicos referentes a transferências de dados entre a UCP e a memória principal. Um pedido para preencher um registrador de propósito geral com os dados de uma célula de memória é comumente chamado *carga* (LOAD); de maneira recíproca, um pedido para transferir os dados de um registrador para uma posição de memória é denominado *armazenamento* (STORE). Na Figura 2.2, os passos 1 e 2 são instruções de carga, e o passo 4 é uma instrução de armazenamento.

Um grupo importante de instruções da categoria de transferência de dados refere-se aos comandos de comunicação com dispositivos externos ao par processador — memória principal (impressora, teclado, monitor, unidade de disco etc.). Já que tais instruções manipulam ações de entrada ou saída do computador, são classificadas como instruções de E/S e, às vezes, consideradas uma categoria à parte. No entanto, na Seção 2.5, demonstraremos como as operações de E/S freqüentemente são manipuladas pelas mesmas instruções que solicitam transferências de dados entre o processador e a memória principal. Assim, consideraremos as instruções de E/S como parte do grupo de transferência de dados.

**Instruções aritméticas/lógicas** O grupo de instruções aritméticas/lógicas consiste em instruções que dizem à unidade de controle para desencadear atividades na unidade aritmética/lógica. O passo 3 da Figura 2.2 está nesse grupo. Como o próprio nome sugere, a unidade aritmética/lógica destina-se a executar operações outras além das operações aritméticas básicas. Algumas destas operações adicionais são as operações lógicas AND, OR e XOR, que introduzimos no Capítulo 1 e discutiremos com mais detalhes neste capítulo. Em geral, essas operações são utilizadas para manipular *bits* independentemente em um registrador de propósito geral, sem interferir no restante do registrador. Outro conjunto de operações, disponível na maioria das unidades aritméticas e lógicas, permite deslocar o conjunto dos *bits* dos registradores para a direita ou para a esquerda. Essas operações são conhecidas como operações de deslocamento simples (SHIFT) se, em decorrência da execução da operação, forem descartados os *bits* que "caírem para fora" do registrador. São chamadas de operações de rotação (ROTATE) quando os *bits* são reutilizados para preencher as lacunas deixadas na extremidade oposta do registrador como resultado da execução desta operação.

**Controle** O grupo de controle consiste em instruções que tratam da execução do programa, em vez da manipulação de dados. Embora seja extremamente elementar, o passo 5 da Figura 2.2 ilustra esta categoria de instruções. O grupo de instruções de controle engloba muitas das instruções mais interessantes disponíveis em um computador, como a família de instruções de desvio (JUMP ou BRANCH), utilizadas para orientar a unidade de controle a executar uma instrução que não seja a próxima da lista. Essas instruções podem ser de dois tipos: desvios incondicionais e desvios condicionais. Um exemplo do primeiro seria "Desvie para a instrução de número 5", e um do segundo, "Se o valor obtido for 0, então desvie para a instrução de número 5". A diferença é que um desvio condicional resultará em uma alteração no rumo do programa se — e somente se — uma certa condição for satisfeita. Como exemplo, a seqüência de instruções da Figura 2.3 representa um algoritmo de divisão de dois números, no qual o passo 3 é um desvio condicional, cuja finalidade é evitar a possibilidade de ocorrência de uma divisão por zero.

| | |
|---|---|
| **Passo 1.** | CARREGUE um registrador com um valor da memória. |
| **Passo 2.** | CARREGUE outro registrador com outro valor da memória. |
| **Passo 3.** | Se o segundo valor for zero, DESVIE para o passo 6. |
| **Passo 4.** | Divida o conteúdo do primeiro registrador pelo segundo e guarde o resultado em um terceiro registrador. |
| **Passo 5.** | GUARDE na memória o conteúdo do terceiro registrador. |
| **Passo 6.** | PARE. |

**Figura 2.3** Roteiro para a divisão de dois números armazenados na memória.

## Uma linguagem de máquina ilustrativa

Verifiquemos de que maneira podem ser codificadas as instruções de uma linguagem de máquina comum. O computador que usaremos para a nossa discussão está resumido na Figura 2.4 e descrito mais detalhadamente


**Figura 2.4** A arquitetura do computador usado no Apêndice C.

no Apêndice C. Possui 16 registradores de propósito geral e 256 células na memória principal, cada qual com capacidade de oito *bits*. Para fins de referência, rotulamos os registradores com os valores de 0 a 15 e endereçamos as células com os valores de 0 a 255. Por conveniência, representamos esses rótulos e endereços com valores na base dois e comprimimos os padrões de *bits* usando a notação hexadecimal. Assim, os registradores são rotulados com 0 a F e as células de memória, endereçadas com 0 a FF.

A versão codificada de uma instrução de máquina normalmente é composta de duas partes: o **campo do código de operação** e o **campo do operando**. O padrão de *bits* que aparece no campo do código de operação indica qual das operações elementares, tais como STORE, SHIFT, XOR e JUMP, está sendo especificada pela instrução. Os padrões de *bits* encontrados no campo do operando complementam a informação sobre a operação indicada pelo código de operação. Por exemplo, no caso da operação STORE, a informação no campo de operando indica qual registrador contém os dados a serem armazenados e qual a posição da memória que deverá receber tais dados.

A linguagem de máquina completa do nosso computador (Apêndice C) consiste em apenas 12 instruções básicas. Cada uma é codificada usando um total de 16 *bits*, representados por quatro dígitos hexadecimais (Figura 2.5). O código de operação correspondente a cada instrução consiste nos quatro primeiros *bits* ou, de maneira equivalente, no primeiro dígito hexadecimal. Note (Apêndice C) que esses códigos de operação são representados pelos dígitos hexadecimais entre 1 e C, inclusive. Mais especificamente, a tabela no Apêndice C nos diz que uma instrução iniciada com o dígito hexadecimal 3 se refere a uma instrução STORE, e qualquer código de instrução que comece com o dígito hexadecimal A corresponde a uma instrução ROTATE.

Observando o campo do operando, percebe-se que ele consiste em três dígitos hexadecimais (12 *bits*) e, em cada caso (com exceção da instrução STOP, que não precisa de qualquer refinamento adicional), complementa a instrução geral especificada pelo código de operação. Por exemplo (Figura 2.6), se o primeiro dígito hexadecimal de uma instrução for 3 (código da operação STORE, para armazenar o conteúdo de um registrador), o próximo dígito hexadecimal da instrução indicará qual dos registradores será utilizado, e os dois últimos dígitos hexadecimais indicarão a posição de memória que receberá os dados. Assim, a instrução 35A7 (em hexade-

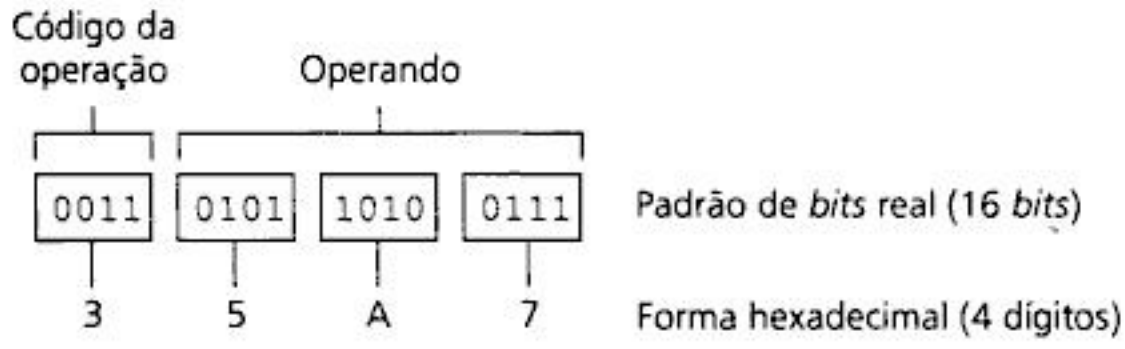



**Figura 2.5** A composição de uma instrução para a máquina do Apêndice C.

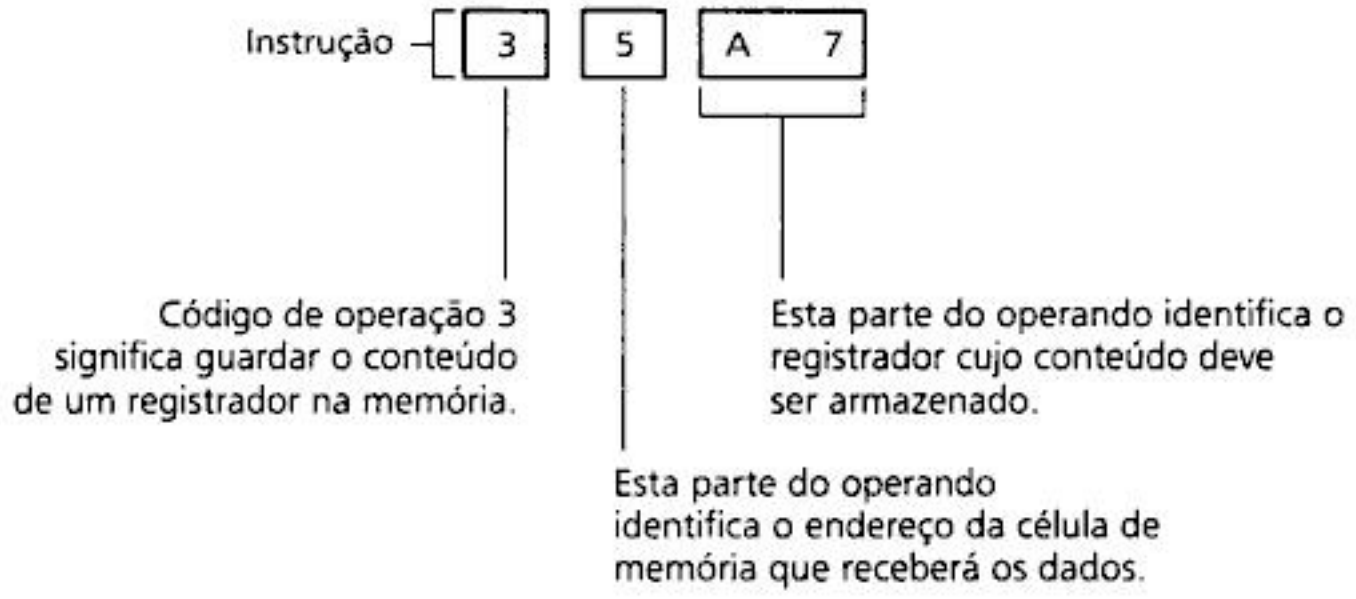



**Figura 2.6** Decodificação da instrução 35A7.

cimal) é traduzida como "Armazene o padrão de *bits* encontrado no registrador 3 na célula de memória cujo endereço é A7".

Outro exemplo, o código de operação 7 (em hexadecimal), especifica que seja aplicada a operação OR sobre o conteúdo de dois registradores. (Veremos mais tarde na Seção 2.4 o que isso significa, por ora nosso interesse é meramente a codificação das instruções.) Nesse caso, o dígito hexadecimal seguinte indica onde o resultado deverá ser colocado, enquanto os dois últimos dígitos hexadecimais do campo do operando são utilizados para indicar quais serão os dois registradores cujos conteúdos irão sofrer a ação da operação OR. Assim, a instrução 70C5 equivale ao comando "Efetue a operação lógica OR sobre os conteúdos dos registradores C e 5 e coloque o resultado no registrador 0".

Existe uma distinção sutil entre as duas instruções LOAD da nossa máquina. Aqui, o código de operação 1 (hexadecimal) se refere à instrução que carrega um registrador com o conteúdo de uma posição de memória, enquanto o código de operação 2 (hexadecimal) refere-se à instrução que carrega um registrador com um dado valor. A diferença é que o campo do operando de uma instrução do primeiro tipo contém um endereço, enquanto o do segundo tipo contém o próprio padrão de *bits* a ser carregado.

A máquina tem duas instruções de adição (ADD), uma para somar números representados em complemento de dois e outra para somar representações em vírgula flutuante. Esta distinção resulta do fato de que efetuar a adição de padrões de *bits* em notação binária exige da unidade aritmética/lógica ações diferentes de quando se efetuam adições em notação de vírgula flutuante.

Encerramos esta seção com a Figura 2.7, que contém uma versão codificada das instruções na Figura 2.2. Pressupomos que os valores a serem somados estão armazenados na notação de complemento de dois nos endereços de memória 6C e 6D, e que a soma deve ser posta na célula de memória no endereço 6E.

| Instrução codificada | Tradução |
|---|---|
| **156C** | Carregue o registrador 5 com o padrão de *bits* encontrado na célula de memória no endereço 6C. |
| **166D** | Carregue o registrador 6 com o padrão de *bits* encontrado na célula de memória no endereço 6D. |
| **5056** | Some o conteúdo dos registradores 5 e 6 considerando que representam números na notação de complemento de dois e deixe o resultado no registrador 0. |
| **306E** | Armazene o conteúdo do registrador 0 na célula de memória de endereço 6E. |
| **C000** | Pare. |

**Figura 2.7** Uma versão codificada das instruções da Figura 2.2.

### *QUESTÕES/EXERCÍCIOS*

1. Por que o termo mover é considerado incorreto para denominar a operação de transferência de dados de uma posição do computador para outra?
2. No texto, as instruções de desvio (JUMP) foram expressas identificando, de forma explícita, a instrução-alvo pelo nome (ou o número do passo) destinatário (por exemplo, "Desvie para o passo 6"). A desvantagem desta técnica é que, se futuramente o nome (ou número) da instrução for alterado, deveremos localizar todos os desvios para tal instrução e alterar o nome referenciado, para que aponte para o destino correto. Descreva outro modo de expressar uma instrução como essa, de forma que o nome da instrução-alvo não seja explicitamente declarado.
3. A instrução "Se 0 for igual a 0, então desvie para o passo 7" é um desvio condicional ou incondicional? Explique sua resposta.
4. Escreva, em padrões binários, o programa exemplo da Figura 2.7.
5. As seguintes instruções foram codificadas na linguagem de máquina descrita no Apêndice C. Reescreva-as em português.
   a. 368A b. BADE c. 803C d. 40F4
6. Qual a diferença entre as instruções 15AB e 25AB na linguagem de máquina do Apêndice C?
7. Eis algumas instruções escritas em português. Traduza-as para a linguagem de máquina do Apêndice C.
   a. Carregue o registrador número 3 com o número hexadecimal 56.
   b. Rode, três *bits* à direita, o registrador número 5.
   c. Execute uma operação AND entre o conteúdo do registrador A e o conteúdo do registrador 5 e coloque o resultado no registrador 0.

## 2.3 Execução de programas

Um computador efetua a execução de um programa armazenado em sua memória copiando, sempre que necessário, as instruções da memória para a unidade de controle. Uma vez na unidade de controle, cada instrução é decodificada e executada. A ordem em que as instruções são trazidas da memória corresponde à ordem na qual elas estão armazenadas na memória, exceto se algo em contrário for especificado por meio de uma instrução JUMP de desvio.

Para compreender o funcionamento de todo o processo de execução, é necessário observar mais detalhadamente a unidade de controle, no interior da UCP. Nela, estão dois registradores de propósito específico: o **contador de instruções*** e o **registrador de instruções** (Figura 2.4). O contador de instruções contém o endereço da próxima instrução a ser executada, servindo como instrumento para o computador manter-se informado sobre a posição do programa em que está ocorrendo a execução. O registrador de instruções é usado para manter a instrução que estiver sendo executada.

A unidade de controle executa seu trabalho repetindo, continuamente, um algoritmo, o **ciclo de máquina**, que consiste em três passos: busca, decodificação e execução (Figura 2.8). Durante o passo de busca, a unidade de controle solicita que a memória principal forneça a instrução armazenada no endereço indicado pelo contador de instruções. Uma vez que cada instrução da nossa máquina possui dois *bytes*, esse processo de busca inclui a transferência de duas células da memória principal. A unidade de controle guarda a instrução recebida da memória em seu registrador de instruções e então aumenta

*N. de T. Em inglês, *program counter*.


**Figura 2.8** O ciclo de máquina.

dois o contador de instruções de modo que este passasse a conter o endereço da próxima instrução armazenada na memória. Assim, o contador de instruções estará pronto para ser usado durante a próxima busca.

Agora, dispondo da instrução em seu registrador de instruções, a unidade de controle inicia a fase da decodificação, que envolve a separação dos componentes do campo de operando, de acordo com o código da operação.

A unidade de controle então executa a instrução, ativando os circuitos necessários para a realização da tarefa. Por exemplo, se a instrução solicitar que se carregue um registrador com um dado da memória, a unidade de controle ativará tal carregamento; se a instrução corresponder a uma operação aritmética, ela ativará o circuito apropriado da unidade aritmética/lógica, alimentando-o com os registradores indicados como os operandos da instrução.

Quando a instrução tiver sido totalmente executada, a unidade de controle recomeçará o ciclo, partindo do passo de busca. Observe-se que, como o contador de instruções, foi incrementado ao término do passo de busca anterior, ele agora indicará novamente para a unidade de controle o endereço correto da próxima instrução a ser executada.

Um caso muito especial refere-se à execução de uma instrução de desvio (JUMP). Por exemplo, considere a instrução B258 (Figura 2.9) que significa "Desvie para a instrução do endereço 58 se o conteúdo do registrador 2 for igual ao do registrador 0". Neste caso, o passo de execução do ciclo de máquina inicia com a comparação dos conteúdos dos registradores 2 e 0. Se forem diferentes, o passo de execução terminará, e iniciar-se-á a próxima busca. Contudo, se seus conteúdos forem iguais, a máquina colocará o valor 58 no contador de instruções durante o passo de execução. Nesse caso, então, o próximo passo de busca encontrará 58 no contador de instruções; assim, a instrução contida nesse endereço será a próxima a ser buscada e executada.


**Figura 2.9** Decodificação da instrução B258.

Note que se a instrução fosse B058, então a decisão de alterar o contador de instruções dependeria da igualdade de conteúdos entre o registrador 0 e o registrador 0. Todavia, esses são o mesmo registrador, portanto, devem ter o mesmo conteúdo. Logo, qualquer instrução na forma B0XY causará um desvio para a posição de memória com endereço XY.

## Um exemplo de execução de programa

Sigamos o ciclo de máquina percorrido ao executar o programa apresentado na Figura 2.7, que toma dois valores da memória, computa a sua soma e armazena o total em outra célula de memória. Primeiro, colocamos o programa em alguma área da memória. No nosso exemplo, ele é armazenado em endereços sucessivos a partir do endereço hexadecimal A0. Com o programa armazenado desta forma, colocamos o endereço da primeira instrução (A0) no contador de instruções e acionamos a máquina (Figura 2.10).

A unidade de controle começa o passo de busca extraindo a instrução (156C) da posição A0 e colocando-a em seu registrador de instruções (Figura 2.11a). Note-se que, no nosso computador, as instruções possuem 16 *bits* (dois *bytes*). Logo, a instrução que está sendo buscada ocupa duas posições de memória, nos endereços A0 e A1. A unidade de controle deve estar projetada para aceitar tal situação. Desta forma, ela lê o conteúdo de tais posições de memória e o deposita adequadamente no registrador de instruções, de 16 *bits*. A seguir, a unidade de controle soma 2 ao conteúdo do contador de instruções, de forma que este passe a conter o endereço da próxima instrução a ser executada (Figura 2.11b). Ao término do passo de busca do primeiro ciclo de máquina, o contador e o registrador de instruções apresentam os seguintes dados:

Contador de instruções: A2
Registrador de instruções: 156C

Em seguida, a unidade de controle analisa a instrução existente no seu registrador de instruções e conclui que deve carregar no registrador 5 o conteúdo da posição de memória de endereço 6C. Este carregamento é executado durante o passo de execução, e então a unidade de controle inicia um novo ciclo.

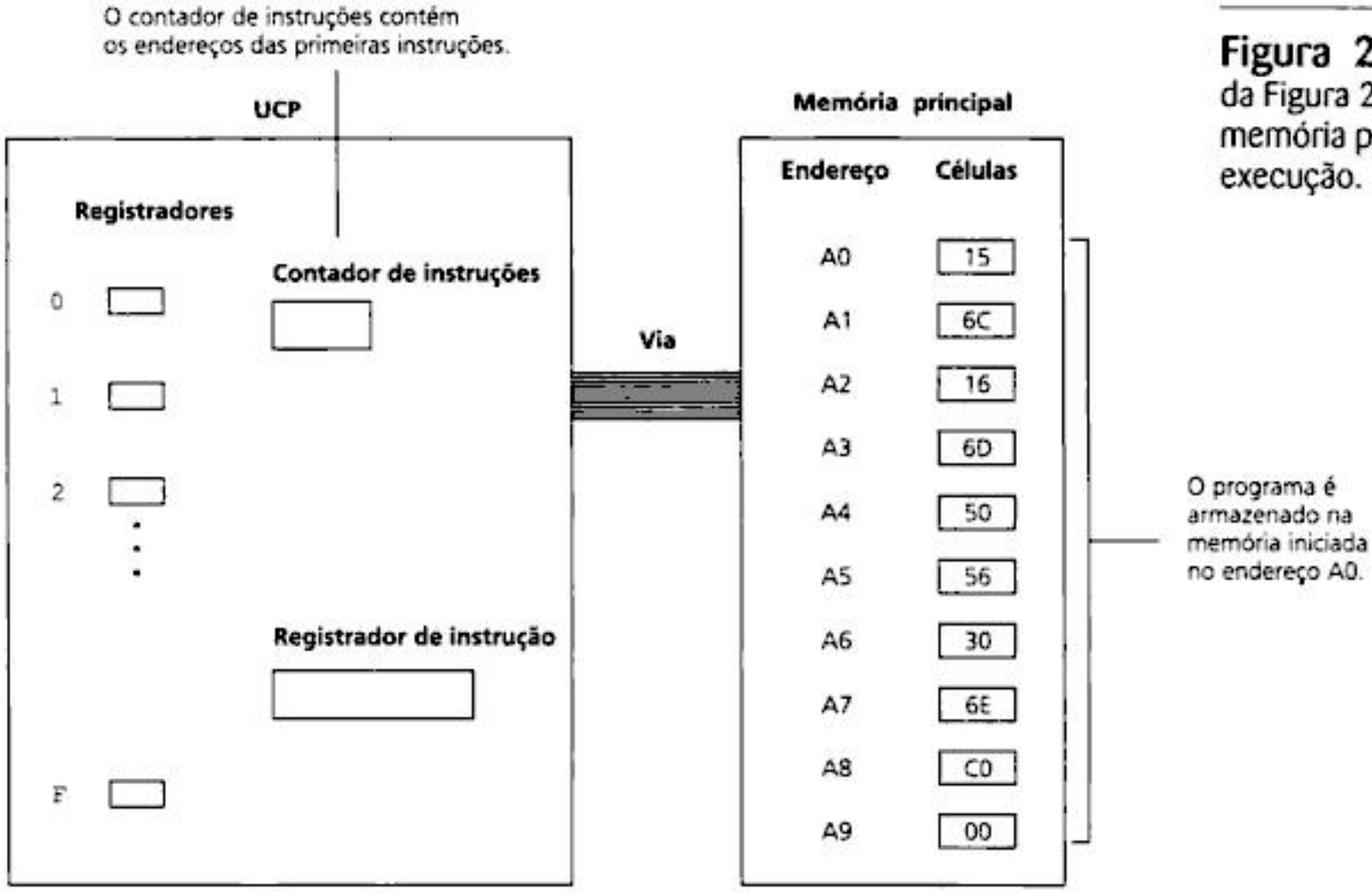
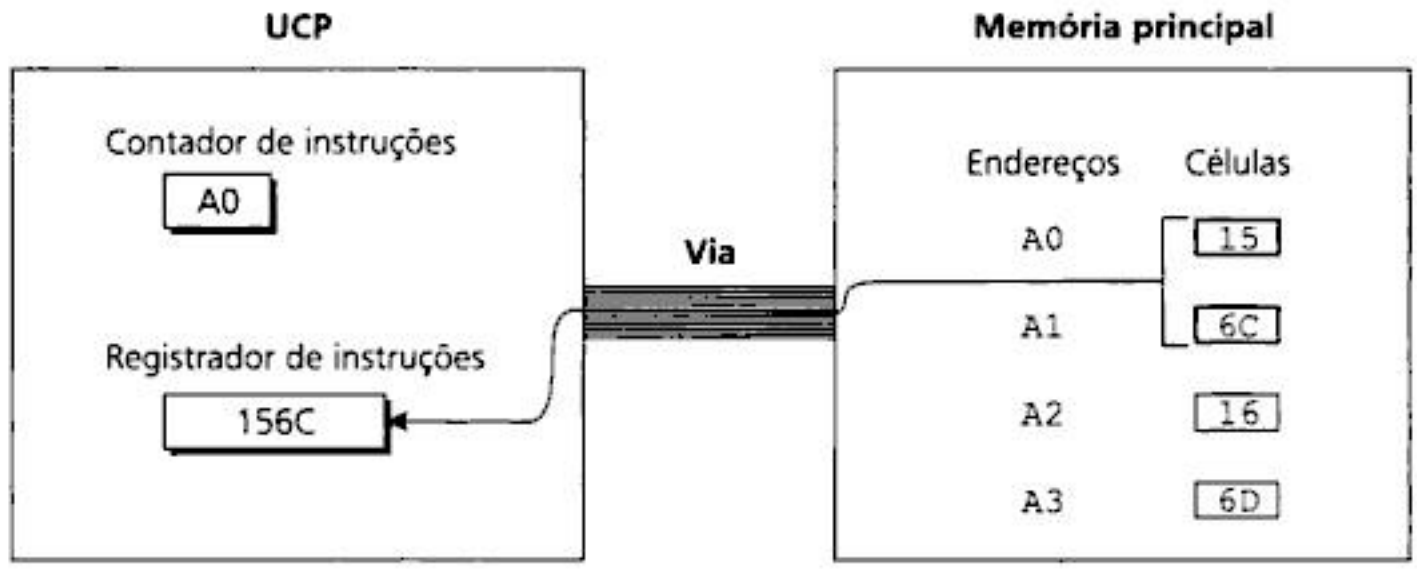
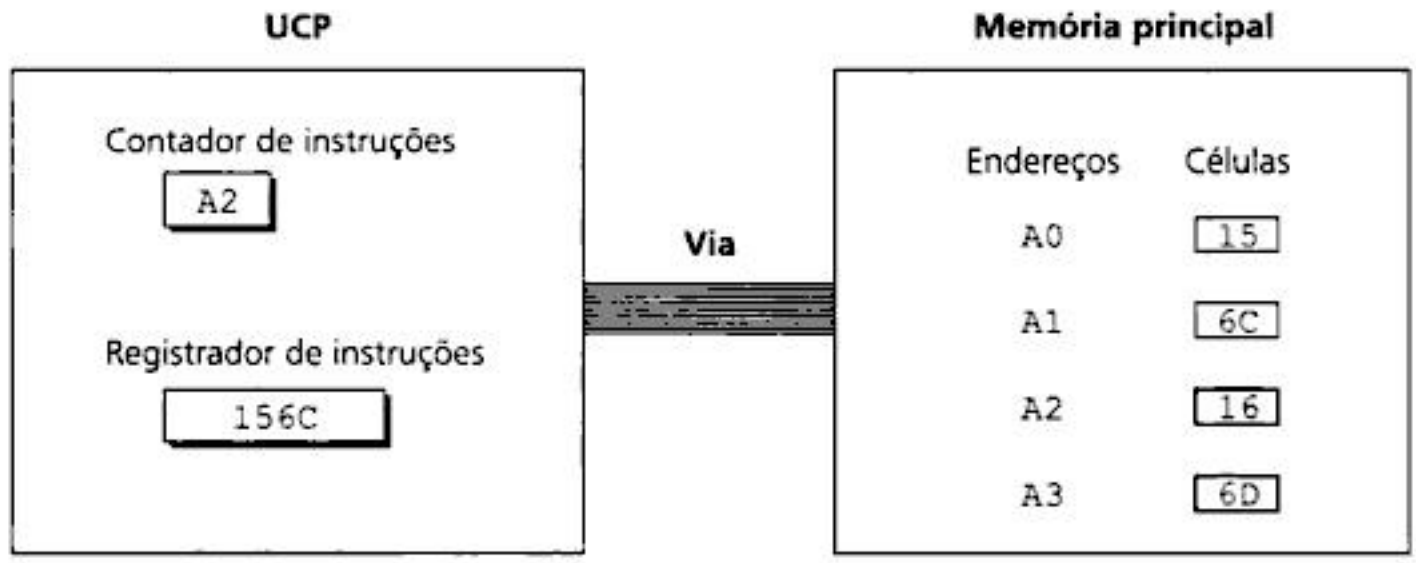



**Figura 2.10** Programa da Figura 2.7 armazenado na memória pronto para a execução.




**a.** No início do passo de busca, a instrução iniciada no endereço A0 é trazida da memória e colocada no registrador de instruções.




**b.** Então o contador de instruções é incrementado, de modo a apontar para a próxima instrução.

**Figura 2.11** Executando o passo de busca do ciclo de máquina.

Esse ciclo inicia buscando a instrução 166D nas posições de memória de endereços A2 e seguinte. A unidade de controle coloca esta instrução em seu registrador de instruções e incrementa o contador para A4. Os valores contidos no contador e no registrador de instruções serão:

Contador de instruções: A4
Registrador de instruções: 166D

Agora, a unidade de controle decodificará a instrução 166D e determinará o carregamento, no registrador 6, do conteúdo da memória de endereço 6D. A seguir, entrará no passo de execução, quando o registrador 6 será devidamente carregado.

Como o contador de instruções agora contém A4, a unidade de controle obterá a próxima instrução a partir de tal endereço. Como resultado, o número 5056 é depositado no registrador de instruções, sendo o contador de instruções incrementado para A6. Em seguida, a

## *Instruções de tamanho variável*

Para simplificar a explanação do texto, a linguagem de máquina descrita no Apêndice C usa tamanho fixo (dois *bytes*) para todas as instruções; a UCP sempre traz o conteúdo de duas células consecutivas da memória e aumenta o contador de instruções em dois. Na realidade, a maioria das linguagens de máquina usa instruções de tamanho diferentes. A série Pentium, por exemplo, possui instruções com tamanhos que variam de um único *byte* até múltiplos *bytes*, dependendo do uso exato da instrução. As UCPs com essas linguagens de máquina determinam o tamanho da instrução pelo seu código de operação, isto é, a UCP primeiramente busca o código da operação da instrução e então, baseada no padrão de *bits* recebido, sabe quantos *bytes* devem ser buscados da memória para obter o restante da instrução.

unidade de controle decodifica o conteúdo do seu registrador de instruções, entra no passo de execução e ativa o circuito de adição em complemento de dois, tendo como dados de entrada os registradores 5 e 6.

Durante este passo, a unidade aritmética/lógica executa a adição, deposita o resultado no registrador 0 (conforme solicitado pela unidade de controle) e informa à unidade de controle que a sua tarefa está terminada. A unidade de controle começa então outro ciclo de máquina. Mais uma vez, com a ajuda do contador de instruções, vai buscar a próxima instrução (306E), localizada nas posições de memória de endereços A6 e A7, e incrementa o contador de instruções para A8. A instrução é então decodificada e executada. Neste ponto, a soma obtida é depositada na posição de memória 6E.

A próxima instrução é obtida na posição de memória A8 e o contador de instruções é incrementado para AA. O conteúdo do registrador de instruções (C000) é decodificado como uma instrução de parada. Em conseqüência, o computador pára durante o próximo passo de execução do ciclo de máquina e a execução do programa se completa.

Em resumo, verificamos que a execução de um programa armazenado na memória envolve o mesmo processo que um de nós usaria para seguir uma lista detalhada de instruções. Da mesma forma que se necessita cuidar para não perder o ponto da lista enquanto se efetuam as operações indicadas, o computador faz uso do contador de instruções para esta finalidade. Determinada a próxima instrução a executar, ela é lida, e o seu significado, extraído. Finalmente, a tarefa solicitada é executada, e retorna-se à lista, em busca da próxima instrução, da mesma forma como o computador executa suas instruções durante o passo de execução, prosseguindo com a próxima busca.

## Programas *versus* dados

Muitos programas podem ser simultaneamente armazenados na memória principal de um computador, desde que ocupem posições diferentes. Assim, é fácil escolher o programa que deve ser executado ao ligar o computador, fixando-se adequadamente um valor para o contador de instruções.

Todavia, deve-se considerar que, como os dados também estão colocados na memória e codificados como cadeias de 0s e 1s, o computador não tem como distinguir os dados dos programas. Se for atribuído ao contador o endereço de algum dado em vez do endereço de uma instrução do programa e o computador não dispuser de outra informação, interpretará esses dados, escritos em padrões de *bits*, como instruções e os executará. O resultado dessa execução dependerá, portanto, dos dados envolvidos.

Não devemos concluir que o fato de programas e dados terem uma aparência comum na memória da máquina seja de todo ruim. Na verdade, ele tem demonstrado ser um atributo útil, pois permite ao programa manipular outros programas (ou mesmo a si próprio) como se fossem dados. Imagine, por exemplo, um programa que se automodifique em resposta a sua interação com o ambiente e assim apresente a capacidade de aprender, ou talvez um programa que escreva e execute outros programas com o propósito de resolver os problemas apresentados a ele.

## *QUESTÕES/EXERCÍCIOS*

**1.** Suponha que as células de memória nos endereços 00 a 05, do computador descrito no Apêndice C, contenham os padrões hexadecimais de *bits* da seguinte tabela:

| Endereço | Conteúdo |
|---|---|
| 00 | 14 |
| 01 | 02 |
| 02 | 34 |
| 03 | 17 |
| 04 | C0 |
| 05 | 00 |

Se ativarmos o computador e o seu contador de instruções contiver 00, qual será o padrão de *bits* na posição de memória de endereço hexadecimal 17 quando o computador parar?

**2.** Suponha que as células de memória nos endereços B0 a B8, do computador descrito no Apêndice C, contenham os padrões hexadecimais de *bits* da seguinte tabela:

| Endereço | Conteúdo |
|---|---|
| B0 | 13 |
| B1 | B8 |
| B2 | A3 |
| B3 | 02 |
| B4 | 33 |
| B5 | B8 |
| B6 | C0 |
| B7 | 00 |
| B8 | 0F |

a. Se o contador de instruções começar com B0, qual será o padrão de *bits* no registrador número 3 depois de executada a primeira instrução?
b. Qual será o padrão de *bits* na posição de memória B8 quando a instrução de parada for executada?

**3.** Suponha que as células de memória nos endereços de A4 a B1, do computador descrito no Apêndice C, contenham os padrões hexadecimais da seguinte tabela:

| Endereço | Conteúdo |
|---|---|
| A4 | 20 |
| A5 | 00 |
| A6 | 21 |
| A7 | 03 |
| A8 | 22 |
| A9 | 01 |
| AA | B1 |
| AB | B0 |
| AC | 50 |
| AD | 02 |
| AE | B0 |
| AF | AA |
| B0 | C0 |
| B1 | 00 |

Responda às seguintes questões, pressupondo que o computador inicie enquanto o seu contador de instruções contenha A4:

a. O que há no registrador 0 na primeira vez em que a instrução de endereço AA foi executada?
b. O que há no registrador 0 na segunda vez em que a instrução de endereço AA foi executada?
c. Quantas vezes a instrução do endereço AA é executada antes de o computador parar?

**4.** Suponha que as células de memória nos endereços de F0 a F9, do computador descrito no Apêndice C, contenham os padrões hexadecimais de *bits* listados na seguinte tabela:

| Endereço | Conteúdo |
|---|---|
| F0 | 20 |
| F1 | C0 |

| | |
|---|---|
| F2 | 30 |
| F3 | F8 |
| F4 | 20 |
| F5 | 00 |
| F6 | 30 |
| F7 | F9 |
| F8 | FF |
| F9 | FF |

Se iniciarmos o computador e seu contador de instruções contiver F0, o que ele executará ao alcançar a instrução do endereço F8?

## 2.4 Instruções aritméticas e lógicas

Conforme foi mencionado anteriormente, o grupo das instruções aritméticas e lógicas compõe-se de instruções que solicitam operações aritméticas, lógicas e de deslocamentos (*shifts*). Nesta seção, analisaremos com maior atenção tais operações.

### Operações lógicas

No Capítulo 1, foram apresentadas as operações lógicas AND (e), OR (ou) e XOR (ou exclusivo), como operações que combinam dois *bits* de entrada para produzir um único *bit* de saída. Essas operações podem ser estendidas de forma que combinem duas cadeias de *bits* para produzir uma única cadeia de saída, aplicando, para tanto, a operação básica para cada coluna individual. Por exemplo, o resultado da operação AND dos padrões 10011010 e 11001001 será:

```
    10011010
AND 11001001
    --------
    10001000
```

onde apenas escrevemos, na base de cada coluna de dois dígitos, o resultado da operação AND dos dois *bits* em cada coluna. Do mesmo modo, efetuando-se as operações OR e XOR nestes mesmos moldes, produzir-se-iam:

```
    10011010
OR  11001001
    --------
    11011011
```

```
    10011010
XOR 11001001
    --------
    01010011
```

Uma das principais aplicações do operador AND é para colocar 0s em uma parte de um padrão de *bits* sem alterar a outra parte. Por exemplo, suponha que seja o *byte* 00001111 o primeiro operando da operação AND. Mesmo desconhecendo o conteúdo do segundo operando, podemos concluir que os quatro *bits* mais significativos do resultado são 0s. Além disso, que os quatro *bits* menos significativos do resultado são uma cópia da parte correspondente do segundo operando, como demonstrado no seguinte exemplo:

```
    00001111
AND 10101010
    --------
    00001010
```

Esta utilização do operador AND exemplifica a técnica denominada mascaramento (*masking*), em que um operando, chamado **máscara**, determina a parte do outro operando que irá afetar o resultado. No caso da operação AND, a máscara produz uma saída que é uma réplica parcial de um dos operandos, cujas posições não-duplicadas são preenchidas com 0s.

Essa operação é útil na manipulação de **mapas de *bits***, que constituem cadeias de *bits* em que cada um representa a presença ou a ausência de um objeto. Já vimos os mapas de *bits* no contexto da representação de imagens, onde cada *bit* é associado a um *pixel*. Como outro exemplo, uma cadeia de 52 *bits*, em que cada um está associado a uma carta de um baralho, pode ser usada para representar a mão de um jogador de pôquer. Para tanto, basta associar 1s aos 5 *bits* que representam as cartas da mão e 0s a todos os demais. Do mesmo modo, um mapa de 52 *bits*, dos quais 13 são 1s, pode ser utilizado para representar uma mão em um jogo de *bridge* e um mapa de 32 *bits*, para representar os 32 tipos disponíveis de sabores de sorvete.

Suponhamos, então, que uma célula de memória de oito *bits* esteja sendo usada como mapa de *bits*, e que desejemos saber se o objeto associado ao terceiro *bit* mais significativo está presente. Precisamos apenas aplicar a operação AND entre o *byte* completo e a máscara 00100000, o que resultará em um *byte* só de 0s, se, e somente se, o terceiro *bit* mais significativo do mapa for 0. O programa pode, então, tomar a ação apropriada, se a operação AND for seguida de uma instrução condicional. Além disso, se o terceiro *bit* mais significativo for 1 e desejarmos alterá-lo para 0, mas sem modificar os demais *bits*, poderemos efetuar a operação AND entre o mapa de *bits* e a máscara 11011111 e então armazenar o resultado no mapa de *bits* original.

Enquanto o operador AND pode ser utilizado para duplicar uma parte de uma cadeia e ao mesmo tempo preencher com 0s a parte não-duplicada, o operador OR pode ser usado para duplicar uma parte de uma cadeia de *bits*, preenchendo com 1s a parte não-duplicada. Para isso, novamente usamos uma máscara, mas, desta vez, indicamos com 0s a posição dos *bits* a serem duplicados e com 1s as posições não-duplicadas. Por exemplo, aplicar o operador OR entre qualquer *byte* e a máscara 11110000 produz um resultado com 1s nos seus quatro *bits* mais significativos, enquanto os demais *bits* conterão uma cópia dos quatro *bits* menos significativos do outro operando, como demonstra o seguinte exemplo:

```
        11110000
OR      10101010
        --------
        11111010
```

Como conseqüência, enquanto a máscara 11011111 pode ser usada pelo operador AND para forçar a inserção de um 0 no terceiro *bit* mais significativo de um mapa de oito *bits*, a operação OR com a máscara 00100000 pode sê-lo para forçar a inserção de um 1 naquela posição.

A principal aplicação do operador XOR está em formar o complemento de uma cadeia de *bits*. Por exemplo, note a relação existente entre o segundo operando e o resultado do seguinte exemplo:

```
        11111111
XOR     10101010
        --------
        01010101
```

## *Comparação da potência de computadores*

Quando se compra um computador pessoal, a freqüência do relógio é comumente usada para comparar as máquinas. O **relógio** de um computador é um circuito oscilador que gera pulsos usados para coordenar as atividades da máquina – quanto mais rápido for esse circuito oscilador, mais rapidamente a máquina executa o seu ciclo. A velocidade do relógio é medida em hertz (abreviado com Hz) – um Hz é igual a um ciclo (ou pulso) por segundo. As velocidades normais nos computadores de mesa estão na faixa de algumas centenas de MHz (modelos antigos) a vários GHz. (MHz é abreviação de megahertz, que é um milhão de Hz. GHz é abreviação de gigahertz, que é 1000 MHz.)

Infelizmente, projetos diferentes de UCP podem realizar quantidades de trabalho diferentes em cada ciclo de máquina. Assim sendo, a velocidade do relógio sozinha deixa de ser relevante quando se compara máquinas com UCPs diferentes. Quando se deseja comparar uma máquina baseada no PowerPC com outra baseada no Pentium, é mais significativo comparar os desempenhos por meio de *benchmarking*, que é o processo de comparar o desempenho de máquinas diferentes executando o mesmo programa, conhecido como **amostra de teste** (*benchmark*). Selecionando-se as amostras de teste que representam diferentes tipos de aplicação, as comparações obtidas podem ser significativas para diferentes segmentos do mercado. A melhor máquina para uma aplicação pode não ser a melhor para outra.

Aplicar o operador XOR entre qualquer *byte* e uma máscara de 1s produz o complemento do primeiro *byte*.

Na linguagem de máquina descrita no Apêndice C, os códigos de operação 7, 8 e 9 são usados para as operações lógicas OR, AND e XOR, respectivamente. Cada um deles especifica que a operação correspondente deve ser efetuada com o conteúdo de dois registradores designados e o resultado, colocado em outro registrador. Por exemplo, a instrução 7ABC indica que o resultado da operação AND nos registradores B e C deve ser colocado no registrador A.

## Operações de rotação e deslocamento

As operações do tipo rotação e deslocamento permitem movimentar *bits* no interior de um registrador e freqüentemente são utilizadas para resolver problemas de alinhamento, como preparar um *byte* para usos futuros em operações que envolvam máscaras ou na manipulação da mantissa das representações em vírgula flutuante. Tais operações são classificadas de acordo com a direção de seu movimento (à direita ou à esquerda) e o fato de o deslocamento ser ou não circular. Neste esquema de classificação existem misturas de terminologia muito diversas. Analisemos rapidamente as idéias envolvidas.

Se iniciarmos com uma seqüência de *bits* com um *byte* de comprimento e aplicarmos a operação de deslocamento (*shift*) de um *bit* para a direita ou para a esquerda ao seu conteúdo, poderemos imaginar um *bit* saindo para fora do *byte* por uma das extremidades e um vazio aparecendo na extremidade oposta. O que é feito com este *bit* perdido e com o vazio decorrente é que distingue os vários tipos de operações de deslocamento. Uma das possibilidades consiste em depositar o *bit* extra na lacuna formada na outra extremidade. O resultado é um deslocamento circular, também conhecido como rotação. Assim, se executarmos oito vezes um deslocamento circular à direita em um dado *byte*, obteremos o mesmo padrão de *bits* com o qual iniciamos.

Outra possibilidade é descartar o *bit* perdido, preenchendo cada lacuna resultante com um 0. O termo **deslocamento lógico** é empregado para designar tais operações. Um deslocamento lógico à esquerda pode ser utilizado para multiplicar números em notação de complemento de dois por 2. Afinal de contas, nessa notação, deslocar dígitos binários à esquerda corresponde a multiplicar por 2, e um deslocamento similar com dígitos decimais corresponde a multiplicar por 10. Do mesmo modo, a divisão inteira por 2 pode ser realizada deslocando-se para a direita a cadeia binária. Nos dois tipos de deslocamento, é preciso tomar alguns cuidados para que o *bit* de sinal seja preservado quando se usam certos sistemas de notação. Por essa razão, freqüentemente encontramos deslocamentos à direita que sempre preenchem a lacuna (que fica na posição do *bit* de sinal) não com zeros, mas com seu valor original. Os deslocamentos que mantêm inalterado o *bit* de sinal às vezes são chamados de **deslocamentos aritméticos**.

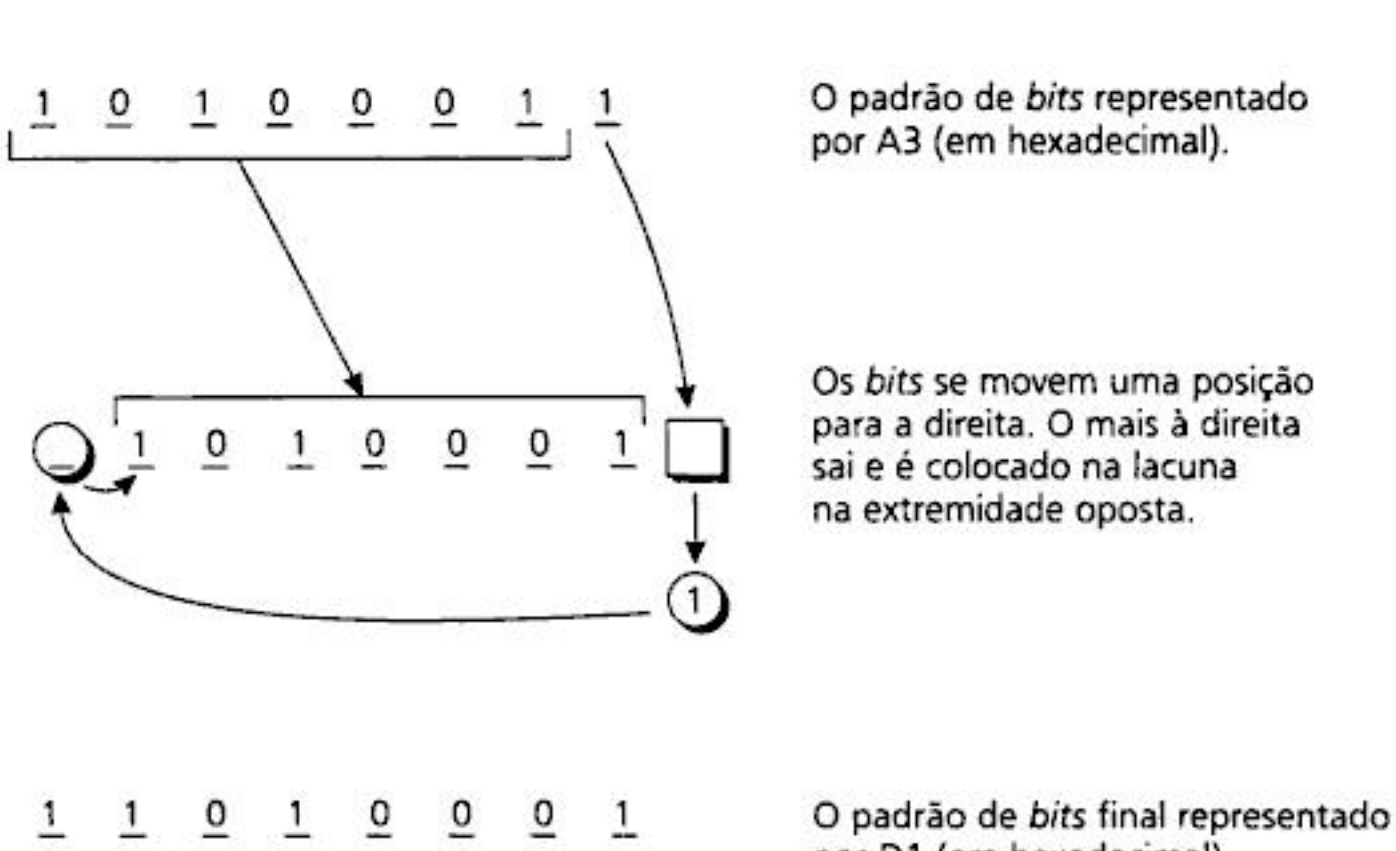



**Figura 2.12** Rotação à direita de um *bit* no padrão de *bits* A3.

Dentre as diversas possibilidades de instruções de deslocamento e rotação, a linguagem de máquina descrita no Apêndice C contém apenas uma instrução de deslocamento

circular à direita designada pelo código de operação A. Neste caso, o primeiro dígito hexadecimal no operando especifica o registrador a ser rotacionado e o restante do operando, o número de *bits* a rotacionar. Assim, a instrução A501 significa "Rotacionar o registrador 5 à direita em 1 *bit*". Se o registrador 5 originalmente continha o padrão de *bits* A3, passará a conter D1 após a execução da instrução (Figura 2.12). (Você pode constatar que outras instruções de deslocamento e rotação podem ser produzidas a partir de combinações das instruções contidas na linguagem de máquina do Apêndice C. Por exemplo, uma vez que um registrador tenha comprimento de oito *bits*, um deslocamento circular à direita de três *bits* produzirá o mesmo resultado que um deslocamento circular à esquerda de cinco *bits*.)

## Operações aritméticas

Embora já tenhamos mencionado as operações aritméticas de adição, subtração, multiplicação e divisão, alguns detalhes ainda precisam ser esclarecidos. Primeiramente, como já foi dito no Capítulo 1, este conjunto de operações geralmente pode ser realizado a partir de uma única operação de adição e de um processo de negação. Por esta razão, alguns computadores pequenos são projetados apenas com a instrução de adição.

Também devemos lembrar que existem diversas variantes para cada operação aritmética. Já comentamos isso em relação às operações de adição disponíveis em nosso computador do Apêndice C. No caso da adição, por exemplo, se os números a somar forem representados na notação de complemento de dois, o processo de adição deverá ser executado como uma adição binária. Contudo, se os operandos forem representados como números em vírgula flutuante, o processo de adição irá extrair a mantissa de cada número, deslocá-la para a direita ou esquerda de acordo com o conteúdo do campo de expoente, conferir os *bits* de sinal, executar a adição e converter o resultado para a notação de vírgula flutuante. Assim, embora ambas sejam denominadas operações de adição, o comportamento do computador é totalmente diferente em cada caso, pois, para a máquina, as duas operações não guardam qualquer relação entre si.

### *QUESTÕES/EXERCÍCIOS*

**1.** Execute as operações indicadas:

| | | | | | |
|---|---|---|---|---|---|
| a. | 01001011 | b. | 10000011 | c. | 11111111 |
| | AND 10101011 | | AND 11101100 | | AND 00101101 |
| d. | 01001011 | e. | 10000011 | f. | 11111111 |
| | OR 10101011 | | OR 11101100 | | OR 00101101 |
| g. | 01001011 | h. | 10000011 | i. | 11111111 |
| | XOR 10101011 | | XOR 11101100 | | XOR 00101101 |

**2.** Suponha que você deseje isolar os três *bits* do centro de uma cadeia de sete sem alterá-los e preencher os outros quatro *bits* com 0s. Qual máscara você deverá usar, em conjunto com qual operação?

**3.** Suponha que você deseje obter o complemento dos três *bits* centrais de uma cadeia de sete, mantendo inalterados os outros quatro. Qual máscara deverá ser usada, em conjunto com qual operação?

**4.** a. Suponha que o operador XOR seja inicialmente aplicado entre os dois primeiros *bits* de uma cadeia de *bits* e que o operador XOR seja aplicado sucessivamente para os demais *bits* entre cada resultado intermediário obtido e o próximo *bit* da cadeia. Qual a relação entre o resultado obtido e o número de 1s existente na cadeia?

b. Como este problema se relaciona com o da determinação do *bit* apropriado de paridade, nos casos de codificação de mensagens?

**5.** Em geral, é mais conveniente usar uma operação lógica do que uma operação aritmética equivalente. Por exemplo, a operação lógica AND combina dois *bits*, de forma muito similar à multiplicação aritmética. Qual a operação lógica cujo resultado se assemelha ao de somar dois *bits* e qual a discrepância que se observa neste caso?

**6.** Qual operação lógica, em conjunto com qual máscara, poderá ser usada para alterar códigos ASCII, convertendo letras minúsculas em maiúsculas? E para efetuar a operação inversa?

**7.** Qual o resultado obtido ao executar um deslocamento circular de três *bits* para a direita sobre as seguintes cadeias de *bits*?
a. `01101010` b. `00001111` c. `01111111`

**8.** Qual o resultado obtido ao executar um deslocamento circular de um *bit* para a esquerda sobre os seguintes *bytes*, representados em notação hexadecimal? Dê sua resposta em formato hexadecimal.
a. AB b. 5C c. B7 d. 35

**9.** Um deslocamento circular de três *bits* para a direita sobre uma cadeia de oito *bits* é equivalente a um deslocamento circular de quantos *bits* para a esquerda?

**10.** Qual o padrão de *bits* que representa a soma de 01101010 e 11001100 se os padrões representam valores denotados em complemento de dois? E se os padrões representarem valores na notação de vírgula flutuante estudada no Capítulo 1?

**11.** Utilizando a linguagem de máquina do Apêndice C, escreva um programa que coloque um 1 no *bit* mais significativo da posição de memória de endereço A7, sem alterar os demais *bits*.

**12.** Utilizando a linguagem de máquina do Apêndice C, escreva um programa que copie os quatro *bits* centrais da posição de memória ED nos quatro *bits* menos significativos da posição de memória E1, completando com 0s os quatro *bits* mais significativos desta mesma posição.

## 2.5 Comunicação com outros dispositivos

A memória principal e a UCP formam o núcleo de um computador. Nesta seção, investigaremos como esse núcleo, ao qual nos referiremos como computador, se comunica com os dispositivos periféricos, como os sistemas de armazenamento em disco, as impressoras, os teclados, ou *mouses*, os monitores e mesmo outros computadores.

### Comunicação via controladores

Em geral, a comunicação entre o computador e outros dispositivos é feita através de um dispositivo intermediário, conhecido como **controlador**. No caso de um computador pessoal, o controlador corresponde fisicamente a uma placa de circuitos, que é encaixada na placa onde está o circuito principal do computador (placa-mãe). A seguir, o controlador é conectado por cabos aos dispositivos dentro do computador ou talvez a um conector na parte posterior, onde os dispositivos externos são ligados. Esses controladores geralmente são eles próprios pequenos computadores, cada um com os seus circuitos de memória e processador, que executam um programa para dirigir as atividades do controlador.

O controlador converte as mensagens e os dados de um lado para outro, em formatos respectivamente compatíveis com as características internas do computador e do(s) dispositivo(s) periférico(s) ligado(s) ao controlador. Assim, cada controlador é projetado para um tipo específico de dispositivo.

Isso explica por que às vezes um novo controlador deve ser adquirido quando se compra um novo dispositivo periférico. Um controlador usado com um disco antigo pode não ser compatível com um novo.

Quando um controlador é afixado à placa-mãe do computador, ele fica eletronicamente ligado à mesma via que conecta o processador com a memória principal (Figura 2.13). Nesta posição, cada controlador se habilita a monitorar os sinais provenientes da UCP e da memória principal, bem como a injetar seus próprios sinais na via.

Mais especificamente, a UCP pode se comunicar com o controlador conectado à via da mesma maneira como ela se comunica com a memória principal. Para enviar um padrão de *bits* ao controlador, ele deve ser inicialmente construído e colocado em um registrador de propósito geral da UCP. Então, uma instrução similar a STORE é executada para "armazenar" o padrão de *bits* no controlador. Da mesma forma, para receber um padrão de *bits* armazenado no controlador, uma instrução similar a LOAD é usada. Em alguns projetos de computador, códigos adicionais de operação são incorporados à linguagem de máquina para estas operações. As instruções com esses códigos de operação são chamadas instruções de E/S. Elas identificam os diversos controladores por um sistema de endereçamento similar ao usado na memória principal. Cada controlador recebe um conjunto único de endereços (endereços de E/S) que é usado nas instruções de E/S para identificá-lo. O conjunto de endereços atribuído a um controlador é chamado **porta** porque representa um "local" por onde a informação entra e sai do computador. Uma vez que o aspecto desses endereços de E/S pode ser idêntico ao dos endereços da memória principal, as vias dos computadores desse tipo de projeto contêm um sinal que indica se a mensagem transmitida na via é para a memória principal ou para um controlador. Assim, em resposta a uma instrução de E/S para enviar o conteúdo de um registrador para um controlador específico, a UCP reage de uma maneira similar a uma instrução para enviar um padrão de *bits* para uma posição de memória, exceto em que ela indica na via que o padrão de *bits* vai para o controlador e não para a memória.

Uma alternativa para estender a linguagem de máquina incluindo códigos especiais de operação para as instruções de E/S é usar os mesmos códigos de operação LOAD e STORE já disponíveis para a comunicação com a memória principal. Nesse caso, cada controlador é projetado para responder a referências a um único conjunto de endereços (novamente chamado porta), enquanto a memória principal é projetada para ignorar as referências a essas localizações. Assim, quando a UCP envia uma mensagem na via para armazenar um padrão de *bits* em uma posição de memória atribuída a um controlador, ele é recebido pelo controlador, e não pela memória. De maneira semelhante, se a UCP tentar ler dados destas posições de memória com uma instrução de carga (LOAD), receberá um padrão de *bits* do controlador, e não da memória. Esse sistema de comunicação é cha-

## *Projeto de via*

O projeto das vias de um computador é uma matéria delicada. Por exemplo, os fios de uma via mal projetada podem funcionar como pequenas antenas – captando sinais transmitidos (de rádio, televisão, etc) e interrompendo a comunicação entre a UCP, a memória e dispositivos periféricos. Além disso, o comprimento da via (em torno de 15 cm) é significativamente maior do que os "fios" dentro do processador (medidos em microns). Assim, o tempo necessário para os sinais viajarem ao longo da via é muito maior do que o de transferência de sinais dentro da UCP. O resultado é que a tecnologia de vias está em constante competição para poder acompanhar a tecnologia das UCPs. Os computadores pessoais atuais refletem a variedade de projetos de vias, que diferem em características como a quantidade de dados transferidos simultaneamente, a taxa em que os sinais na via podem ser mudados e as propriedades físicas das conexões entre a via e a placa do controlador.

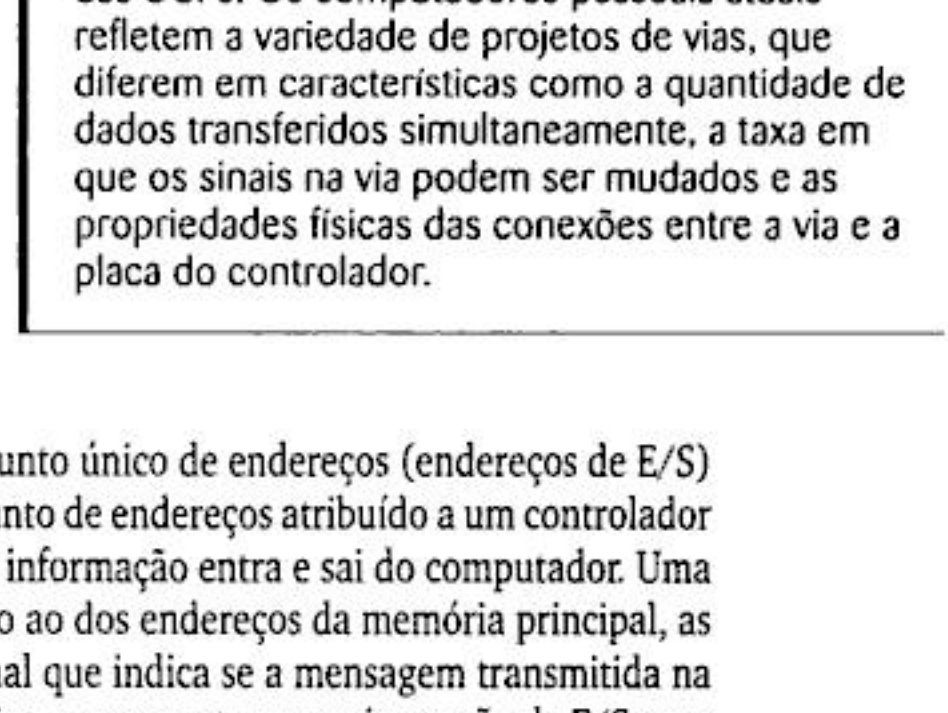



**Figura 2.13** Controladores conectados a uma via.

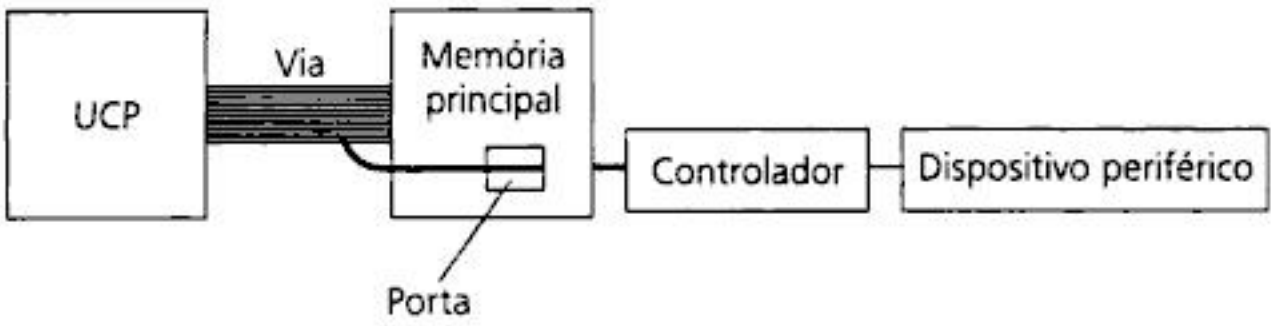



**Figura 2.14** Uma representação conceitual da entrada/saída mapeada na memória (*memory mapped I/O*).

mado **E/S mapeada na memória** (*memory mapped I/O*) porque os dispositivos de entrada/saída do computador aparentam estar em várias posições da memória (Figura 2.14).

Uma vez que o controlador esteja conectado à via do computador, ele poderá realizar as suas próprias comunicações com a memória principal durante os nanossegundos em que a UCP não estiver utilizando a via. Essa capacidade do controlador de ter acesso à memória principal é conhecida como **acesso direto à memória** (*direct memory access* — DMA), e é um recurso significativo para o desempenho do computador. Por exemplo, para recuperar dados do setor de um disco, a UCP pode enviar as requisições codificadas em padrões de *bits* ao controlador ligado ao disco, pedindo-lhe que leia o setor e coloque os dados em um bloco específico de células da memória. A UCP pode então continuar com outras tarefas enquanto o controlador realiza a operação de leitura e deposita os dados na memória via DMA. Assim, duas atividades estão em andamento ao mesmo tempo: a UCP está executando um programa enquanto o controlador supervisiona a transferência de dados entre o disco e a memória principal. Desta maneira, os recursos computacionais do processador não são desperdiçados durante a transferência de dados relativamente lenta.

O uso do DMA também tem um efeito prejudicial de aumentar a quantidade de comunicação que trafega pela via do computador. Os padrões de *bits* devem mover-se entre a UCP e a memória principal, entre a UCP e cada controlador e entre cada controlador e a memória principal. A coordenação de toda essa atividade na via é uma questão fundamental de projeto. Mesmo em um excelente projeto, a via central pode tornar-se um impedimento, uma vez que a UCP e os controladores competem entre si visando o acesso à via. Esse impedimento é conhecido como **gargalo de Von Neumann**, pois é uma conseqüência da **arquitetura de Von Neumann** subjacente, na qual a UCP busca as suas instruções através da via central.

Finalmente, devemos notar que a transferência de dados entre dois componentes de um computador raramente é feita em um só sentido. Ainda que possamos pensar na impressora como um dispositivo que apenas recebe dados, a verdade é que ela também envia dados de retorno ao computador. De fato, um computador produz e envia dados a uma impressora muito mais rapidamente do que a sua velocidade de impressão. Se um computador enviar os dados a uma impressora de maneira cega, ela poderá ficar saturada, e isso resultaria na perda de dados. Assim, um processo como o de impressão de um documento envolve um diálogo constante nos dois sentidos, no qual o computador e o dispositivo periférico trocam informações sobre o estado do dispositivo.

Esse diálogo freqüentemente envolve uma **palavra de estado** (*status word*), um padrão de *bits* gerado pelo dispositivo periférico e enviado ao controlador. Os *bits* na palavra de estado refletem as condições do dispositivo. Por exemplo, no caso de uma impressora, o valor do *bit* menos significativo da palavra de estado pode indicar que ela está sem papel, enquanto o próximo *bit*, se ela está pronta para receber mais dados. Dependendo do sistema, o controlador responde a uma informação de estado por si próprio, ou a disponibiliza para a UCP. Em cada caso, o programa do controlador ou o que está sendo executado pela UCP pode ser projetado para sustar o envio de dados para a impressora até que a informação de estado apropriada seja recebida.

## Taxas de comunicação de dados

A taxa em que os *bits* são transferidos de um componente a outro dentro de um computador é medida em ***bits* por segundo** (bps). Unidades comuns incluem kbps (kilo-bps, igual a 1.000 bps), Mbps (mega-bps, igual a 1 milhão de bps) e Gbps (giga-bps, igual a 1 bilhão de bps). A taxa de

transmissão disponível em um caso particular depende do tipo de caminho para a comunicação e da tecnologia usada em sua implementação. Essa taxa máxima freqüentemente é igual à **largura de banda** (*bandwidth*) do caminho de comunicação, isto é, dizer que um caminho de comunicação possui uma alta largura de banda significa que ele é capaz de transferir *bits* a uma taxa alta. Infelizmente, este é um uso impreciso do termo, uma vez que "taxas altas de transferência" podem significar taxas diferentes em diferentes situações.

Existem dois tipos básicos de caminhos de comunicação: paralelo e serial. Esses termos se referem à maneira como os padrões de *bits* são transferidos. No caso da **comunicação paralela**, vários *bits* são transferidos ao mesmo tempo, cada um em uma linha separada. Essa técnica permite a rápida transferência de dados, mas exige um caminho de comunicação relativamente complexo. Exemplos incluem a via interna do computador e a maioria das comunicações entre este e seus dispositivos periféricos, como os sistemas de armazenamento em massa e a as impressoras. Nesse caso, as taxas medidas em Mbps e unidades maiores são comuns.

A **comunicação serial**, por sua vez, se baseia na transferência de um único *bit* por vez. Esta técnica tende a ser mais lenta, porém requer um caminho de comunicação mais simples, porque todos os *bits* são transferidos em uma só linha, um após o outro. Geralmente é utilizada para a comunicação entre computadores, em que os caminhos de comunicação mais simples são mais econômicos.

Por exemplo, as linhas telefônicas atuais são inerentemente sistemas de comunicação serial, uma vez que transmitem tons um após o outro. A comunicação entre computadores nessas linhas é realizada primeiramente pela conversão dos padrões de *bits* em tons audíveis por meio de um **modem** (abreviatura de modulador / demodulador), transferência dos tons serialmente pelo sistema telefônico e finalmente a reconversão dos mesmos para *bits* por meio de outro modem instalado no destino.

Na realidade, a simples representação de padrões de *bits* por tons de diferentes freqüências (conhecida como chave por deslocamento de freqüência) é usada apenas nas comunicações em baixa velocidade, a não mais que 1200 bps. Para alcançar taxas de transferência de 2.400 bps, 9.600 bps e maiores, os modens combinam as mudanças de freqüência dos tons com as de amplitude (volume) e de fase (grau de atraso no qual a transmissão é realizada). Para alcançar taxas ainda maiores, as técnicas de compressão de dados freqüentemente são aplicadas, produzindo taxas aparentes de transferência de até 57,6 Kbps.

Essas taxas de transferência aparentam ser o limite que pode ser atingido na faixa de freqüência utilizada pelas linhas telefônicas tradicionais (de até 3 Hz), e são insuficientes para as necessidades atuais de comunicação. A transferência de gráficos a 57,6 Kbps pode se tornar exasperante, e o vídeo está à frente da razão. Assim, novas tecnologias vêm sendo desenvolvidas para proporcionar taxas de transferência maiores para os clientes que até então usavam as linhas telefônicas tradicionais. Uma dessas técnicas é conhecida como DSL (*Digital Subscriber Line*) que leva em conta o fato de que as linhas telefônicas atuais são capazes de trabalhar com freqüências maiores do que as utilizadas para a comunicação de voz. Esses sistemas normalmente obtêm taxas de transferência em torno de 1,5 Mbps, mas podem atingir até 6 Mbps em uma direção se as funcionalidades na outra direção forem restringidas. O desempenho exato depende da versão da DSL utilizada e do comprimento da linha até a central telefônica, que geralmente é limitado a não mais que 6 Km (3,5 milhas). Outras tecnologias que competem com a DSL incluem a tecnologia a cabo, como a utilizada nos sistemas de televisão a cabo, que pode atingir taxas na ordem de 40 Mbps, e as fibras óticas, cujas taxas se situam na faixa entre Mbps e Gbps.

## *QUESTÕES/EXERCÍCIOS*

1. Pressuponha que a máquina descrita no Apêndice C usa E/S mapeada na memória e que o endereço B5 é a posição da porta da impressora para a qual os dados a serem impressos devem ser enviados.
   a. Se o registrador 7 contém a código ASCII da letra A, que instrução em linguagem de máquina deve ser usada para efetivar a impressão dessa letra?

b. Se a máquina executa um milhão de instruções por segundo, quantas vezes esse caractere pode ser enviado à impressora em um segundo?
c. Se a impressora é capaz de imprimir cinco páginas convencionais de texto em um minuto, ela será capaz de acompanhar a velocidade em que os caracteres são enviados em (b)?

**2.** Suponha que o disco rígido de seu computador pessoal gire a uma velocidade de 3.000 rotações por minuto, cada trilha contenha 16 setores e cada setor, 1.024 *bytes*. Aproximadamente que taxa de transferência é necessária entre o acionador do disco e o controlador para que este último receba os *bits* do disco assim que forem lidos?

**3.** Quanto tempo leva para se transferir um romance de 300 páginas codificado em ASCII a uma taxa de transferência de 57.600 bps?

## 2.6 Outras arquiteturas

Para ampliar nossa perspectiva, consideremos algumas alternativas à arquitetura de máquina estudada nas seções precedentes.

### Canalização (*pipelining*)

Os sinais elétricos percorrem um fio condutor com uma velocidade não superior à da luz. Como a luz viaja a aproximadamente 1 pé por nanossegundo*, são necessários, no mínimo, 2 nanossegundos para a unidade de controle do processador buscar uma instrução de uma posição de memória que esteja a 1 pé de distância. (O pedido de leitura deve ser enviado para a memória, o que requer pelo menos 1 nanossegundo, e a instrução deve ser enviada de volta à unidade de controle, o que requer outro nanossegundo, no mínimo.) Como conseqüência, buscar e executar uma instrução nesta máquina exige vários nanossegundos, o que significa que aumentar a velocidade de execução de um computador corresponde, em última instância, a um problema de miniaturização. Embora muitos avanços fantásticos tenham sido realizados nesta área, isto continua a ser um fator limitante.

Em um esforço para resolver tal dilema, os engenheiros de computação se concentraram no conceito de vazão (*throughput*), em vez da mera velocidade de execução. A **vazão** se refere à quantidade total de trabalho realizado pelo computador em um dado período de tempo, e não ao tempo que ele leva para realizar uma dada tarefa.

Um exemplo de como a vazão de um computador pode ser aumentada sem exigir aumento na velocidade de execução envolve a **canalização** (*pipelining*)**, que é a técnica de permitir que os passos de um ciclo da máquina se sobreponham. Especificamente, enquanto uma instrução está sendo executada, a próxima instrução pode ser buscada, o que significa que mais de uma instrução pode estar no "cano" a qualquer momento, cada uma em um estágio diferente de seu processamento. Portanto, a vazão total da máquina é aumentada, muito embora o tempo necessário para buscar e executar cada instrução individual permaneça o mesmo. (Obviamente, quando uma instrução de desvio é encontrada, qualquer ganho que teria sido obtido com a busca prévia não se efetiva, pois as instruções do "cano" não são as desejadas.)

Os projetos das máquinas modernas empurram o conceito de canalização para além do nosso simples exemplo. Elas são capazes de buscar várias instruções ao mesmo tempo e de fato executar mais de uma instrução simultaneamente desde que elas não dependam uma da outra.

---

*N. de T. No sistema métrico, 1 pé = 0,3048m; 1 nanossegundo = $10^{-9}$ segundo = um bilionésimo de segundo. A velocidade da luz é cerca de $3{,}10^{8}$m/s.

**N. de T. *Pipeline* é um conceito similar ao de uma *linha de produção*, capaz de aumentar a vazão de uma fábrica sem alterar a velocidade do trabalho dos operários, por meio da especialização dos mesmos e do trabalho em equipe, explorando o paralelismo.

## Máquinas com multiprocessamento

A canalização pode ser vista como um primeiro passo para o **processamento paralelo**, que é a realização de várias atividades ao mesmo tempo. Contudo, o processamento verdadeiramente paralelo exige mais de uma unidade de processamento, o que resulta em computadores conhecidos como máquinas com multiprocessamento.

Atualmente, diversas máquinas têm sido projetadas dentro deste princípio. Uma forma de atingir tal meta consiste em conectar a uma mesma memória principal vários processadores, todos similares ao processador central existente em um computador convencional. Nesta configuração, os processadores podem funcionar de modo independente, embora coordenando seus esforços por meio de mensagens enviadas de uns para os outros através das suas áreas comuns de memória. Por exemplo, quando um processador recebe a incumbência de executar uma tarefa de grande porte, ele pode armazenar um programa que execute parte desta tarefa na área comum de memória para, em seguida, solicitar que algum outro processador o execute. O resultado é uma máquina em que diferentes seqüências de instruções são executadas com diferentes conjuntos de dados. Máquinas deste tipo são denominadas arquiteturas **MIMD** (*multiple-instruction stream, multiple-data stream* — múltiplos fluxos de instruções, múltiplos fluxos de dados), em vez da tradicional arquitetura **SISD** (*single-instruction stream, single-data stream* — único fluxo de instruções, único fluxo de dados).

Uma outra versão da arquitetura baseada em múltiplos processadores é obtida pela reunião dos processadores, de forma que executem a mesma seqüência de instruções em uníssono, cada qual sobre o seu próprio conjunto de dados. O resultado é um caso de arquitetura **SIMD** (*single-instruction stream, multiple-data stream* — único fluxo de instruções, múltiplos fluxos de dados). Tais máquinas são úteis em aplicações nas quais uma mesma tarefa deva ser aplicada a diversos conjuntos de itens semelhantes, dentro de um grande bloco de dados.

Outra abordagem ao processamento paralelo consiste em construir grandes computadores na forma de um conglomerado de computadores menores, cada qual com sua própria memória e UCP. Neste tipo de arquitetura, cada um dos computadores menores é ligado a seus vizinhos de forma que as tarefas solicitadas ao sistema como um todo possam ser distribuídas entre os diversos computadores individuais. Por conseguinte, se uma tarefa designada a um dos computadores internos puder ser dividida em subtarefas independentes, esta máquina poderá, por sua vez, solicitar que outras máquinas as executem ao mesmo tempo. Conseqüentemente, a tarefa original seria completada em um tempo muito menor do que se fosse executada por uma máquina convencional, com um único processador.

No Capítulo 10, estudaremos outra arquitetura de multiprocessamento, as redes neurais artificiais, cujo projeto se baseia nas teorias de funcionamento do cérebro humano. Tais máquinas consistem em muitos processadores elementares cujas saídas apenas representam reações à combinação de seus dados de entrada. Estes processadores simples são interligados, formando uma rede em que os dados de saída de alguns processadores são utilizados como dados de entrada para os demais. Esta máquina se programa ajustando-se o grau com que cada saída de um processador influi na reação dos outros processadores aos quais se conecta. Isto simula a teoria segundo a qual as redes neurais biológicas aprendem a produzir uma reação particular para um determinado estímulo, ajustando a composição química das conexões (sinapses) entre neurônios, as quais, em troca, ajustam a capacidade que o neurônio apresenta de influir no comportamento dos demais.

Os proponentes das redes neurais artificiais argumentam que embora a tecnologia esteja se habilitando a construir circuitos eletrônicos que contenham aproximadamente o mesmo número de unidades de chaveamento do que o de neurônios no cérebro humano (acredita-se que os neurônios sejam as unidades de chaveamento da natureza), a capacidade das máquinas atuais ainda está muito aquém da do cérebro humano. Isto, argumentam eles, é resultado do uso ineficiente que se faz dos componentes tradicionais dos computadores submetidos à arquitetura de Von Neumann. Afinal de contas, se uma máquina é construída com um grande circuito de memória que suporte alguns processadores, então a maioria de seus circuitos está fadada a ficar ociosa a maior parte do tempo. Em contraste, a maior parte da mente humana pode estar ativa a cada momento.

Assim, o projeto dos computadores atuais está expandindo o modelo básico UCP–memória principal e em alguns casos abandonando-o completamente, a fim de desenvolver máquinas mais úteis.

## *QUESTÕES/EXERCÍCIOS*

1. Com relação à questão 3 da Seção 2.3, se a máquina usar a técnica de canalização discutida no texto, qual será o conteúdo do "cano" quando a instrução de endereço AA for executada? Sob que condições a técnica de canalização não seria benéfica nesse ponto do programa?
2. Quais conflitos devem ser solucionados ao executar o programa da questão 4 da Seção 2.3 em uma máquina com canalização?
3. Suponha dois processadores "centrais", ligados à mesma memória e que executem programas diferentes. Suponha também que um destes processadores necessite somar uma unidade ao conteúdo de uma célula de memória, ao mesmo tempo em que um outro processador necessite subtrair uma unidade do conteúdo desta mesma célula (como resultado, essa célula de memória deveria permanecer intacta).
   a. Descreva uma seqüência de ações que produza como resultado, na posição de memória em questão, uma unidade a menos que o valor inicial.
   b. Descreva uma seqüência de ações cujo resultado, na posição de memória em questão, seja uma unidade a mais que o valor inicial.

# *Problemas de revisão de capítulo*

(Os problemas marcados com asteriscos se referem às seções opcionais.)

1. Dê uma breve definição de cada item abaixo:
   a. Registrador
   b. Memória *cache*
   c. Memória principal
   d. Armazenamento em massa
2. Seja um bloco de dados armazenado nas posições de memória do computador descrito no Apêndice C nos endereços de B9 a C1, inclusive. Quantas células de memória estão neste bloco? Liste os seus endereços.
3. Qual o valor do contador de instruções do computador descrito no Apêndice C logo após a execução da instrução B0BA?
4. Suponha que as células de memória dos endereços de 00 a 05 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 21 |
| 01 | 04 |
| 02 | 31 |
| 03 | 00 |
| 04 | C0 |
| 05 | 00 |

   Pressupondo que inicialmente o contador de instruções contenha 00, registre o conteúdo do contador, do registrador de instruções e da posição de memória de endereço 00 ao término de cada fase de busca do ciclo de máquina, até a sua parada.
5. Suponha que três valores ($x$, $y$ e $z$) estejam armazenados na memória de um computador. Descreva a seqüência de eventos (carregar registradores com dados da memória, armazenar valores na memória e assim por diante) para o cálculo das sentenças matemáticas $x + y + z$. Que tal $(2x) + y$?
6. Seguem abaixo algumas instruções, escritas na linguagem de máquina descrita no Apêndice C. Traduza-as para o português.
   a. 407E
   b. 9028
   c. A302
   d. B3AD
   e. 2835
7. Seja uma linguagem de máquina projetada com códigos de operação de quatro *bits*. Quantos tipos de instrução diferentes esta linguagem pode conter? O que aconteceria se o tamanho do código de operação fosse aumentado para oito *bits*?

**8.** Traduza as seguintes instruções, descritas em português, para a linguagem de máquina descrita no Apêndice C.
   a. Carregar o registrador 8 com o conteúdo da posição de memória 55.
   b. Carregar o registrador 8 com o valor hexadecimal 55.
   c. Deslocar ciclicamente três *bits* à direita o registrador 4.
   d. Aplicar o operador lógico AND sobre o conteúdo dos registradores F e 2 e guardar o resultado no registrador 0.
   e. Desviar para a instrução de memória de endereço 31 se o conteúdo do registrador 0 for igual ao do registrador B.

**9.** Classifique as instruções abaixo (na linguagem de máquina do Apêndice C) sob os seguintes aspectos: a execução da instrução modifica o conteúdo da posição de memória do endereço 3B, utiliza o conteúdo desta posição de memória, ou é independente deste conteúdo?

a. 153B b. 253B c. 353B
d. 3B3B e. 403B

**10.** Suponha que as células de memória dos endereços de 00 a 03 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 23 |
| 01 | 02 |
| 02 | C0 |
| 03 | 00 |

   a. Traduza a primeira instrução para o português.
   b. Se o computador iniciar suas atividades com o valor 00 no seu contador de instruções, qual será o padrão de *bits* existente no registrador 3 quando o computador parar?

**11.** Suponha que as células de memória dos endereços de 00 a 05 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 10 |
| 01 | 04 |
| 02 | 30 |
| 03 | 45 |
| 04 | C0 |
| 05 | 00 |

Responda às seguintes questões, pressupondo que 00 seja o conteúdo inicial do contador de instruções:
   a. Traduza para o português as instruções executadas.
   b. Qual será o padrão de *bits* existente na posição de memória do endereço 45 quando o computador parar?
   c. Qual será o padrão de *bits* existente no contador de instruções quando o computador parar?

**12.** Suponha que as células de memória dos endereços de 00 a 09 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 1A |
| 01 | 02 |
| 02 | 2B |
| 03 | 02 |
| 04 | 9C |
| 05 | AB |
| 06 | 3C |
| 07 | 00 |
| 08 | C0 |
| 09 | 00 |

Pressupondo que a máquina inicie com seu contador de instruções igual a 00:
   a. O que a célula de memória de endereço 00 conterá quando a máquina parar?
   b. Qual será o padrão de *bits* no contador de instruções quando a máquina parar?

**13.** Suponha que as células de memória dos endereços de 00 a 0D do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 20 |
| 01 | 03 |
| 02 | 21 |
| 03 | 01 |
| 04 | 40 |
| 05 | 12 |
| 06 | 51 |
| 07 | 12 |
| 08 | B1 |
| 09 | 0C |
| 0A | B0 |

| | |
|---|---|
| 0B | 06 |
| 0C | C0 |
| 0D | 00 |

Supondo que a máquina inicie com o seu contador de instruções igual a 00:

a. Qual será o padrão de *bits* existente no registrador 1 quando o computador parar?
b. Qual será o padrão de *bits* existente no registrador 0 quando o computador parar?
c. Qual será o padrão de *bits* existente no contador de instruções quando o computador parar?

**14.** Suponha que as células de memória dos endereços de F0 a FD do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| F0 | 20 |
| F1 | 00 |
| F2 | 21 |
| F3 | 01 |
| F4 | 23 |
| F5 | 05 |
| F6 | B3 |
| F7 | FC |
| F8 | 50 |
| F9 | 01 |
| FA | B0 |
| FB | F6 |
| FC | C0 |
| FD | 00 |

Se iniciarmos a máquina com o seu contador de instruções igual a F0, qual será o conteúdo do registrador 0 quando a máquina finalmente executar a instrução de parada na posição FC?

**15.** Se o computador do Apêndice C executar uma instrução a cada microssegundo, quanto tempo ele levará para completar o programa do Problema 14?

**16.** Suponha que as células de memória dos endereços de 20 a 28 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 20 | 12 |
| 21 | 20 |
| 22 | 32 |
| 23 | 30 |
| 24 | B0 |
| 25 | 21 |
| 26 | 20 |
| 27 | C0 |
| 28 | 00 |

Pressupondo que a máquina inicie com o seu contador de instruções igual a 20:

a. Quais serão os padrões de *bits* existentes nos registradores 0, 1 e 2 quando o computador parar?
b. Qual será o padrão de *bits* existente na célula de memória de endereço 30 quando o computador parar?
c. Qual será o padrão de *bits* existente na célula de memória de endereço B0 quando o computador parar?

**17.** Suponha que as células de memória dos endereços de AF a B1 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| AF | B0 |
| B0 | B0 |
| B1 | AF |

O que acontecerá se iniciarmos a máquina com o seu contador de instruções igual a AF?

**18.** Suponha que as posições de memória dos endereços de 00 a 05 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 25 |
| 01 | B0 |
| 02 | 35 |
| 03 | 04 |
| 04 | C0 |
| 05 | 00 |

Se iniciarmos o computador com seu contador de instruções igual a 00, quando o computador irá parar?

**19.** Para cada caso a seguir, escreva um pequeno programa na linguagem de máquina descrita no Apêndice C para executar as ações solicitadas. Trabalhe com a hipótese de que cada programa estará na memória a partir do endereço 00.

a. Mover o valor contido na posição de memória 8D para a posição de memória B3.
b. Trocar entre si os valores armazenados nas posições de memória 8D e B3.
c. Se o valor armazenado nas posições de memória 45 for 00, então colocar o valor CC na posição de memória 88; caso contrário, colocar o valor DD na posição de memória 88.

**20.** Um jogo popular entre os usuários de computador é o *core wars*, uma variação do jogo batalha naval. (O termo *core* tem sua origem nos primórdios da tecnologia de memórias, quando os 0s e os 1s eram representados por campos magnéticos em pequenos anéis feitos de material magnético.) O jogo é disputado entre dois programas adversários, cada qual armazenado em diferentes posições da mesma memória do computador. Admite-se que o computador alterne o processamento dos dois programas, executando ora uma instrução de um, ora do outro. O objetivo de cada programa é destruir o outro, gravando dados estranhos sobre o programa do adversário, sem que, entretanto, os programas conheçam a posição um do outro.

a. Escreva um programa na linguagem de máquina do Apêndice C, que aborde o jogo em uma maneira defensiva e que seja o menor possível.
b. Escreva um programa, na linguagem de máquina do Apêndice C, que tente evitar qualquer ataque do programa adversário movendo-se para outras posições diferentes. Mais precisamente, instrua o seu programa para começar na posição 00, fazer uma cópia de si mesmo na posição 70 e então desviar para esta nova cópia.
c. Estenda o programa do item (b) de modo que ele possa continuar se mudando para novas posições de memória. Em particular, faça o seu programa se mover para a posição 70, depois para E0 (=70 + 70) e, em seguida, para 60 (=70 + 70 + 70) etc.

**21.** Escreva um programa na linguagem de máquina do Apêndice C para computar a soma dos valores armazenados nas posições de memória A1, A2, A3 e A4 e codificados em complemento de dois. O programa deverá armazenar o total na posição de memória A5.

**22.** Suponha que as posições de memória dos endereços de 00 a 05 do computador descrito no Apêndice C contenham os seguintes padrões hexadecimais de *bits*:

| Endereço | Conteúdo |
|---|---|
| 00 | 20 |
| 01 | C0 |
| 02 | 30 |
| 03 | 04 |
| 04 | 00 |
| 05 | 00 |

O que acontecerá se iniciarmos o computador com o 00 no seu contador de instruções?

**23.** O que acontecerá se as posições de memória dos endereços 06 e 07 do computador descrito no Apêndice C contiverem os padrões de *bits* B0 e 06, respectivamente, e se o computador for iniciado com o valor 06 no seu contador de instruções?

**24.** Suponha que o seguinte programa, escrito na linguagem de máquina do Apêndice C, seja armazenado na memória principal a partir do endereço 30 (hexadecimal). Que tarefa o programa executará quando for processado?

```
2003
2101
2200
2310
1400
3410
5221
5331
3239
333B
B248
B038
C000
```

**25.** Resuma os passos envolvidos quando a máquina descrita no Apêndice C executa uma instrução com o código de operação B. Expresse a sua resposta com um conjunto de diretrizes, como se você estivesse dizendo à UCP o quê fazer.

***26.** Resuma os passos envolvidos quando a máquina descrita no Apêndice C executa uma instrução com o código de operação 5. Expresse a sua resposta com um conjunto de diretrizes, como se você estivesse dizendo à UCP o quê fazer.

***27.** Resuma os passos envolvidos quando a máquina descrita no Apêndice C executa uma instrução com o código de operação 6. Expresse a sua resposta com um conjunto de diretrizes, como se você estivesse dizendo à UCP o quê fazer.

***28.** Suponha que os registradores 4 e 5 do computador descrito no Apêndice C contenham os padrões hexadecimais de *bits* 3C e C8, respectivamente. Qual será o padrão de *bits* contido no registrador 0 após a execução das seguintes instruções?

a. 5045 b. 6045 c. 7045
d. 8045 e. 9045

***29.** Utilizando a linguagem de máquina descrita no Apêndice C, escreva programas que executem as seguintes tarefas:

a. Copiar para a posição de memória BB o padrão de *bits* armazenado na posição 66.
b. Alterar os quatro *bits* menos significativos da posição de memória de endereço 34 para 0s, mantendo inalterados os demais *bits*.
c. Copiar os quatro *bits* menos significativos da posição de memória de endereço A5 para os quatro *bits* menos significativos do endereço A6, mantendo inalterados os demais *bits* deste .
d. Copiar os quatro *bits* menos significativos da posição de memória de endereço A5 para os quatro *bits* mais significativos de A5. (Assim, os primeiros quatro *bits* em A5 ficarão iguais aos seus quatro últimos.)

***30.** Efetue as operações indicadas:

a. 111000 AND 101001
b. 000100 AND 101010
c. 000100 AND 010101
d. 111011 AND 110101
e. 111000 OR 101001
f. 000100 OR 101010
g. 000100 OR 010101
h. 111011 OR 110101
i. 111000 XOR 101001
j. 000100 XOR 101010
k. 000100 XOR 010101
l. 111011 XOR 110101

***31.** Identifique a máscara e a operação lógica necessárias à realização dos seguintes objetivos:

a. Colocar 0s nos quatro *bits* centrais de um padrão de oito, mantendo inalterados os demais *bits*.
b. Obter o complemento de um padrão de oito *bits*.
c. Obter o complemento do *bit* mais significativo de um padrão de oito *bits*, sem alterar os demais.
d. Colocar um 1 no *bit* mais significativo de um padrão de oito, sem alterar os demais.
e. Colocar 1s em todos os *bits* de um padrão de oito *bits*, exceto no seu *bit* mais significativo, o qual deverá permanecer inalterado.

***32.** Identifique uma operação lógica (bem como a máscara correspondente) que, quando executada sobre uma cadeia de entrada de oito *bits*, produzirá uma cadeia de saída formada apenas de 0s, se e somente se a cadeia de entrada for 10000001.

***33.** Descreva uma seqüência de operações lógicas (bem como as máscaras correspondentes) que, quando executadas sobre uma cadeia de entrada de oito *bits*, produzirá um *byte* de saída formado apenas de 0s somente se a cadeia de entrada for iniciada e terminada por 1s. Caso contrário, a saída deverá ter pelo menos um dos *bits* igual a 1.

***34.** Indique qual será o resultado obtido ao executar um deslocamento circular de quatro *bits* à esquerda sobre os seguintes padrões de *bits*:

a. 10101 b. 11110000 c. 001
d. 101000 e. 00001

***35.** Qual será o resultado obtido ao executar um deslocamento circular de um *bit* à direita sobre os seguintes *bytes*, representados em notação hexadecimal (forneça suas respostas na notação hexadecimal):

a. 3F b. 0D c. FF d. 77

***36.** Que instrução na linguagem de máquina descrita no Apêndice C poderia ser usada para fazer um deslocamento circular à direita de três *bits* no registrador B?

***37.** Escreva um programa, na linguagem de máquina do Apêndice C, que reverta o conteúdo da posição de memória de endereço 8C.

***38.** Uma impressora que imprime 40 caracteres em um segundo pode manter este ritmo de transferência para uma cadeia de caracteres ASCII (cada qual

com seu *bit* de paridade), chegando serialmente a uma velocidade de 300 bps? E se fosse a 1200 bps?

***39.** Suponha que uma pessoa digite 30 palavras em um minuto. (Considere uma palavra como cinco caracteres.) Se um computador executar uma instrução a cada microssegundo (milésimo de segundo), quantas instruções serão executadas durante o tempo decorrido entre a digitação de dois caracteres sucessivos?

***40.** Quantos *bits* por segundo um teclado deve transmitir para sustentar um ritmo de digitação de 30 palavras por minuto? (Suponha que cada caractere esteja codificado em ASCII, juntamente com o seu *bit* de paridade, e que cada palavra consista em cinco caracteres.)

***41.** Um sistema de comunicação é capaz de transmitir qualquer seqüência de oito estados diferentes a uma taxa de, no máximo, 300 estados por segundo. Com que velocidade, medida em *bits* por segundo, este sistema poderia ser usado para transferir informação?

***42.** Suponha que o computador descrito no Apêndice C se comunique com uma impressora utilizando a técnica de entrada/saída mapeada na memória. Suponha também que o endereço FF seja utilizado para enviar caracteres à impressora, e o endereço FE, para receber informações sobre o estado desta. Em particular, suponha que o *bit* menos significativo do endereço FE indique se a impressora está pronta ou não para receber mais um caractere (0 indica *não-pronto* e 1 indica *pronto*). Iniciando no endereço 00, escreva uma rotina, em linguagem de máquina, que aguarde até que a impressora esteja pronta para receber outro caractere, enviando então para a impressora o caractere representado pelo padrão de *bits* contido no registrador 5.

***43.** Escreva um programa, na linguagem de máquina descrita no Apêndice C, que preencha com 0s todas as posições de memória dos endereços de A0 a C0. Tal programa deverá ser suficientemente pequeno para caber nas posições de memória de endereços de 00 a 13 (hexadecimal).

***44.** Suponha que um computador com 20GB de espaço disponível para armazenamento em disco rígido receba dados por meio de uma conexão telefônica, a uma velocidade de transmissão de 14.400 bps. A esta velocidade, quanto tempo transcorrerá até que todo o espaço disponível seja preenchido?

***45.** Suponha que uma linha de comunicação esteja sendo utilizada para transmitir dados serialmente, a uma velocidade de 14.400 bps. Se houver uma interferência com 0,01 segundo de duração, quantos *bits* de dados serão afetados por ela?

***46.** Suponha que haja 32 processadores disponíveis, cada um capaz de calcular a soma de dois números, compostos de vários dígitos, em um milionésimo de segundo. Descreva como as técnicas de processamento simultâneo podem ser aplicadas para calcular a soma de 64 números em apenas seis milionésimos de segundo. Quanto tempo seria necessário para um único processador calcular esta mesma soma?

***47.** Faça um resumo das diferenças entre as arquiteturas CISC e RISC.

***48.** Identifique dois métodos para aumentar a vazão de processamento em um computador.

***49.** Descreva de que forma a média de um conjunto de números pode ser calculada mais rapidamente com uma máquina com multiprocessamento do que com outra com apenas um processador.

## *Questões sociais*

As seguintes questões procuram auxiliar o leitor no entendimento de alguns assuntos éticos, sociais e legais no campo da computação. O objetivo não é meramente o de fornecer respostas a tais questões. O leitor também deve justificá-las e verificar se as justificativas apresentadas preservam sua consistência de uma questão para outra.

**1.** Suponha que um fabricante de computadores desenvolva uma nova arquitetura de máquina. Até que ponto deve ser permitido a esta companhia manter a propriedade de tal arquitetura? Qual seria a melhor política a adotar, tendo em vista o bem da sociedade?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

PARTE 2

# *Software*

Na primeira parte, discutimos os principais componentes de um computador. Desses, os que são materiais denominam-se *hardware*. Entretanto, os programas que o *hardware* executa são imateriais e denominados *software*. Nesta Parte 2 do livro, dirigiremos a nossa atenção para tópicos relacionados a *software*, o que nos conduzirá ao âmago da Ciência da Computação, o estudo de algoritmos. Em particular, investigaremos a descoberta, a representação e a comunicação de algoritmos.

Começaremos discutindo os sistemas operacionais no Capítulo 3. Tais sistemas são pacotes de *software* grandes e complexos que controlam as atividades globais de um computador ou de um grupo de computadores conectados em rede. Assim, nosso estudo de sistemas operacionais abrange o tópico de redes de computadores em geral e a Internet em particular. No Capítulo 4, estudaremos os algoritmos, com ênfase no seu descobrimento e em suas representações. No Capítulo 5, analisaremos como os algoritmos são comunicados às máquinas por meio do processo de programação e investigaremos as características das linguagens de programação populares. Finalmente, no Capítulo 6, estudaremos todo o processo de desenvolvimento de programas, no contexto da Engenharia de *Software*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

são que propicia de haver mais de uma tarefa sendo executada ao mesmo tempo. Independentemente do número de usuários do ambiente, constatou-se que o conceito de tempo partilhado promovia o aumento da eficiência global de uma máquina. Esta constatação se mostrou particularmente surpreendente ao se levar em conta o considerável processamento adicional exigido para a implementação do revezamento que caracteriza a técnica de tempo partilhado. Entretanto, na sua ausência, um computador acaba gastando mais tempo enquanto espera que seus dispositivos periféricos completem suas tarefas, ou que um usuário faça sua próxima solicitação ao sistema. Em ambientes de tempo partilhado, este tempo pode ser cedido a alguma outra tarefa. Assim, enquanto uma tarefa espera pela sua vez de utilizar o processador, a outra pode prosseguir a sua execução. Como resultado, em um ambiente de tempo partilhado, um conjunto de tarefas pode ser concluído em tempo menor do que se fosse executado de modo seqüencial.

## Sistemas com multiprocessamento

Mais recentemente, a necessidade de compartilhar informações e recursos entre diferentes máquinas suscitou o desejo de unir as máquinas para o intercâmbio de informações. Para preencher essa necessidade, popularizaram-se os sistemas com computadores interconectados, conhecidos como **redes** de computadores. De fato, o conceito de uma máquina central grande, que servisse a muitos usuários, foi substituído pelo conceito de muitas máquinas pequenas, conectadas por uma rede na qual os usuários compartilham recursos espalhados pela rede — como serviços de impressão, pacotes de *software*, equipamentos de armazenamento de dados e de informação. O principal exemplo é a **Internet**, uma rede de redes que hoje une milhões de computadores do mundo todo. Estudaremos a Internet com mais pormenores nas Seções 3.5 e 3.6.

Muitos dos problemas de coordenação que ocorrem em projetos de redes são iguais ou semelhantes aos enfrentados pelos sistemas operacionais. De fato, o *software* de controle de rede pode ser visto como um sistema operacional projetado para grandes redes. Sob este ponto de vista, o desenvolvimento de *software* de redes é uma extensão natural do campo dos sistemas operacionais. Enquanto as redes mais antigas foram construídas como um conjunto de computadores individuais, levemente interligados, cada qual sob o controle do seu próprio sistema operacional, as pesquisas recentes na área de redes estão se concentrando nos sistemas estruturados como grandes redes, cujos recursos são igualmente compartilhados entre as tarefas atribuídas à rede. Essas tarefas são designadas para ser executadas nos processadores da rede, de acordo com a necessidade, sem levar em consideração a posição física real de tais processadores. Um exemplo é o sistema de servidor de nomes utilizado na Internet, que estudaremos na Seção 3.5. Tal sistema permite que uma ampla gama de máquinas, espalhadas pelo mundo, possa trabalhar em conjunto para traduzir a forma humana e mnemônica de endereços da Internet para sua forma numérica, compatível com a rede.

As redes representam apenas um exemplo dos projetos de multiprocessadores que estão inspirando o desenvolvimento dos sistemas operacionais modernos. Enquanto uma rede produz um sistema com multiprocessamento mediante a combinação de máquinas, em que cada uma contém apenas um único processador, outros sistemas com multiprocessamento são projetados como computadores únicos, porém com mais de um processador. Um sistema operacional para tais computadores não apenas coordenará a competição entre as várias tarefas que são de fato executadas simultaneamente, mas também controlará a alocação de tarefas aos diversos processadores. Este processo envolve problemas de **balanceamento de carga** (alocação dinâmica de tarefas a vários processos, de modo a garantir que os processadores sejam utilizados eficientemente), bem como de **escalação** (divisão das tarefas em várias subtarefas, cujo número seja compatível com o número de processadores disponíveis na máquina).

Vimos então que o desenvolvimento de sistemas com multiprocessamento criou dimensões adicionais no estudo de sistemas operacionais, uma área que deverá se manter em franca atividade nos anos vindouros.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

rárquica, possibilitando que cada diretório por sua vez possa conter subdiretórios. Por exemplo, um usuário pode criar um diretório chamado *Registros* que contenha subdiretórios chamados *RegistrosFinanceiros*, *RegistrosMédicos* e *RegistrosDomésticos*. Em cada um destes subdiretórios podem ser armazenados arquivos que pertençam à categoria correspondente. Uma seqüência de aninhamentos de níveis de diretórios é denominada **caminho** (*path*). Os caminhos normalmente são indicados listando os diretórios separados por barras. Por exemplo, *animais/pré-históricos/dinossauros* representa o caminho iniciado no diretório chamado animais, passando por seu subdiretório chamado *pré-históricos* e terminando no subdiretório *dinossauros*.

Qualquer acesso a arquivos, por parte de algum módulo de *software*, é efetuado através do gerente de arquivos. O procedimento inicia com a solicitação ao gerente para fazer acesso ao arquivo. Este procedimento é conhecido como "abrir o arquivo". Se o gerente de arquivos aceitar o pedido, ele fornecerá a informação necessária para encontrar e manipular o arquivo. Essa informação é mantida em uma área da memória principal denominada **descritor de arquivo**. É com base na informação contida nesse descritor de arquivo que operações elementares individuais são executadas sobre o arquivo.

Outro componente do núcleo consiste em uma coleção de **dirigentes de dispositivo** (*device drivers*), que são as unidades de *software* que se comunicam com os controladores (ou, às vezes, diretamente com os dispositivos periféricos) para efetuar as operações nos dispositivos ligados à máquina. Cada dirigente é projetado para um tipo particular de dispositivo (como impressora, acionador de disco, unidade de fita magnética ou monitor) e traduz as requisições genéricas em passos mais técnicos exigidos pela dispositivo atribuído àquele dirigente. Por exemplo, um dirigente de dispositivo para uma impressora contém o *software* de leitura e decodificação da palavra de estado da impressora, bem como todos os detalhes do processo de comunicação. Assim, os outros componentes de *software* não precisam lidar com as tecnicalidades quando desejam imprimir um arquivo. Em vez disso, eles podem meramente pedir ao dirigente de dispositivo para imprimir o arquivo e deixar que ele tome conta dos detalhes. Dessa maneira, o projeto das outras unidades de *software* pode ser independente das características únicas de um dispositivo em particular. O resultado é um sistema operacional genérico que pode ser personalizado para dispositivos particulares ao instalar os dirigentes apropriados.

Um outro componente do núcleo de um sistema operacional é o **gerente de memória**, que se encarrega de coordenar a utilização da memória principal da máquina. Essa função é mínima nos ambientes nos quais a máquina executa apenas uma tarefa a cada instante. Nesses casos, o programa que desempenha a tarefa é colocado na memória principal, executado e então substituído por outro, correspondente à próxima tarefa. Porém, em ambientes multiusuário ou multitarefa, nos quais a máquina se encarrega de várias atividades ao mesmo tempo, os deveres do gerente de memória são mais complexos. Nestes casos, muitos programas e blocos de dados devem coexistir na memória principal, cada um em uma área própria, determinada pelo gerente de memória. Conforme as necessidades das diferentes atividades, o gerente de memória vai providenciando as áreas necessárias e mantendo um mapa das regiões de memória que não estão mais ocupadas.

A tarefa do gerente de memória torna-se mais complexa quando a área total de memória principal solicitada excede o espaço realmente disponível na máquina. Neste caso, o gerente de memória pode criar a ilusão de espaço adicional alternando os programas e os dados entre a memória principal e o sistema de armazenamento em massa. Suponha, por exemplo, que seja solicitada uma memória de 256 MB, mas apenas 128 MB estejam de fato disponíveis. Para criar a ilusão de um espaço de memória maior, o gerente divide o espaço solicitado em unidades chamadas **páginas**, cujo conteúdo ele guarda no sistema de armazenamento em massa (uma página comum possui tamanho não superior a alguns *kilobytes*). Uma vez que diferentes páginas são necessárias na memória principal, o gerente deve trocá-las por outras que não estão mais em uso, e assim as outras unidades de *software* executam como se houvesse 256 MB de memória principal na máquina. Este espaço ilusório de memória é chamado **memória virtual**.

No núcleo de um sistema operacional também estão situados o **escalador** (*scheduler*) e o **despachante** (*dispatcher*), que estudaremos na próxima seção. Por ora, mencionamos apenas que, em um sistema de tempo partilhado, o escalador determina quais atividades serão executadas e o despachante controla a distribuição de fatias de tempo para tais atividades.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

chante então seleciona o processo de maior prioridade dentre os que se encontrarem prontos, reinicia a operação do temporizador e permite que o processo selecionado inicie a sua fatia de tempo.

Fundamental em um sistema de tempo partilhado é a capacidade de parar um processo para continuá-lo mais tarde. Caso ocorra uma interrupção durante a leitura de um livro, a capacidade do leitor de continuar a leitura mais tarde depende de sua habilidade de relembrar o ponto em que parou, bem como de reter a informação acumulada até tal ponto. Em suma, deve ser capaz de recriar o ambiente existente imediatamente antes da ocorrência da interrupção. Esse ambiente é denominado *estado* do processo. Lembre-se que este estado inclui o valor do contador de instruções, o conteúdo dos registradores e os das posições relevantes de memória. As máquinas projetadas para operar sistemas de tempo partilhado incluem recursos para guardar tal estado a cada ocorrência de interrupção. Também possuem instruções em linguagem de máquina para recarregar um estado anteriormente armazenado. Tais características simplificam a tarefa, de responsabilidade do despachante, de alternar os processos e ilustram até que ponto o projeto das máquinas modernas pode ser influenciado pelas necessidades dos sistemas operacionais.

Às vezes, a fatia de tempo de um processo termina antes do tempo determinado pelo temporizador. Por exemplo, se um processo executar uma operação de E/S, solicitando dados de um disco, sua fatia de tempo será truncada pelo sistema, uma vez que, de outra forma, tal processo desperdiçaria o tempo restante dessa fatia, aguardando que o controlador terminasse de executar a operação solicitada. Neste caso, o escalador atualizará a tabela de processos, marcando o processo corrente como em estado de espera, e o despachante fornecerá uma nova fatia a outro processo que já esteja pronto para ser executado. Depois (talvez várias centenas de milissegundos mais tarde), quando o controlador indicar que aquela operação de E/S foi completada, o escalador reclassificará o processo como pronto para a execução, habilitando-o assim a competir novamente por outra fatia de tempo.

## O modelo cliente-servidor

As diversas unidades internas de um sistema operacional funcionam como processos independentes que, em um sistema de tempo partilhado, competem por fatias de tempo, sob a supervisão do despachante. Tais processos se intercomunicam para coordenar suas atividades. Por exemplo, para escalar um novo processo, o escalador solicita espaço de memória ao gerente de memória. Para acessar um arquivo em disco, o processo deve primeiro obter informações do gerente de arquivos.

A troca de mensagens entre processos é chamada **comunicação entre processos** e representa uma área extensa de pesquisa. De fato, a comunicação entre processos pode ser feita de várias formas. Uma delas, chamada **modelo cliente-servidor** (Figura 3.7) tornou-se muito comum no campo das redes de computadores. O modelo define as funções básicas executadas pelos componentes como a de **cliente**, o qual envia suas solicitações para outras unidades, e a de **servidor**, que satisfaz as solicitações recebidas dos clientes. Por exemplo, o gerente de arquivos de um sistema operacional pode funcionar como um servidor, fornecendo acessos a arquivos conforme as solicitações dos seus clientes.

O uso do modelo cliente-servidor em projetos de *software* leva a uma padronização dos tipos de comunicação existentes no sistema. Um cliente apenas faz solicitações aos servidores e espera as suas respostas, enquanto um servidor apenas executa os serviços solicitados e envia as respostas correspondentes aos clientes. O papel de um servidor é o mesmo para clientes que residem na mesma máquina ou em máquinas distantes ligadas na rede. A distinção fica com o *software* que controla a comunicação — não nos clientes e servidores.


**Figura 3.7** O modelo cliente-servidor.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

mando compulsoriamente alguns dos recursos alocados. O nosso exemplo da tabela de processos completamente lotada se encaixa nesta situação. Um administrador de sistema normalmente dimensiona uma tabela de processos suficientemente grande para cada instalação. Entretanto, se os enlaces mortais ocorrerem por causa de alguma tabela lotada, o administrador precisa apenas lançar mão de seu privilégio de *superusuário* para remover (o termo técnico é *kill* — matar) alguns dos processos que lotam esta tabela, garantindo assim que os demais possam continuar suas tarefas.

As técnicas de tratamento das duas primeiras condições são conhecidas como esquemas para evitar os enlaces mortais. Por exemplo, elimina-se a segunda condição exigindo que cada processo solicite todos os recursos necessários de uma só vez. Outra técnica, talvez mais criativa, elimina a primeira condição, sem remover diretamente a competição, mas convertendo recursos não-compartilháveis em compartilháveis. Por exemplo, suponha que o recurso em questão seja uma impressora e vários processos solicitem o seu uso. Toda vez que um processo solicitar a impressora, o sistema operacional atenderá o pedido. Contudo, em lugar de conectar o processo ao dirigente da impressora, o sistema operacional o conecta a um dirigente de dispositivo que armazena a informação a ser impressa em um disco, em vez de enviá-la à impressora. Assim, sob esta configuração, tudo se passa como se cada processo tivesse acesso à impressora e, portanto, cada um continua normalmente a sua execução. Mais tarde, quando a impressora estiver disponível, o sistema operacional transferirá os dados do disco para ela. Desta maneira, o sistema operacional fez um recurso não-compartilhável comportar-se como se fosse compartilhável, criando a ilusão de haver mais de uma impressora. Esta técnica de armazenar dados para uma saída posterior em uma ocasião mais oportuna é chamada de ***spooling***, e é muito comum em sistemas de todos os tamanhos.

Naturalmente, outros problemas podem surgir quando existe competição dos processos pelos recursos de uma máquina. Por exemplo, um gerente de arquivos, em geral, concede a vários processos o acesso a um mesmo arquivo, desde que se trate de acessos apenas para a leitura do arquivo. Nesse caso, ocorrerão conflitos se mais de um processo tentar alterar um mesmo arquivo ao mesmo tempo. Assim, um gerente de arquivos permite acesso ao arquivo de acordo com as necessidades dos processos, permitindo que vários leiam os dados, mas que somente um grave informações em um determinado momento. Outros sistemas dividem o arquivo de forma que diferentes processos possam alterar diferentes partes do mesmo arquivo concomitantemente. Todavia, ainda há problemas que devem ser solucionados. Por exemplo, de que maneira esses processos, que possuem autorização para ler dados de um arquivo, devem ser notificados quando um outro processo estiver alterando o conteúdo do arquivo?

## QUESTÕES/EXERCÍCIOS

1. Suponha que os processos A e B compartilhem tempo na mesma máquina e que ambos necessitem de um mesmo recurso não-compartilhável por pequenos períodos (por exemplo, cada processo imprime uma série de pequenos relatórios independentes). Cada processo pode, repetidamente, acessar um recurso, liberá-lo e depois solicitá-lo novamente. Descubra uma situação adversa quando se resolve controlar os acessos aos recursos da seguinte maneira:

   > Comece atribuindo o valor 0 a um sinalizador. Se o processo A solicitar o recurso e o sinalizador contiver 0, atenda o pedido. Caso contrário, faça o processo A esperar. Se o processo B solicitar o recurso e o sinalizador for 1, atenda o pedido. Caso contrário, faça o processo B esperar. Todas as vezes que o processo A liberar o recurso, altere o sinalizador para 1. Todas as vezes que o processo B terminar de utilizar o recurso, altere o sinalizador para 0.

2. Suponha que uma estrada de dupla pista se reduza a uma pista única ao atravessar um túnel. Para coordenar o uso do túnel, foi instalado o seguinte sistema de sinalização:

   Ao entrar por qualquer extremidade do túnel, um carro faz com que seja ligado um sinal luminoso vermelho nas duas aberturas do túnel. No momento em que o carro sair, esse sinal

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Se alguém "ficar" no portão e "olhar" a nuvem, poderá identificar diversas estruturas. Sem dúvida, a Internet tem crescido de uma maneira imprevisível, uma vez que vários domínios encontram pontos nos quais se conectam à nuvem. Uma estrutura popular, contudo, reúne os portões de diversos domínios para formar uma rede regional de portões. Por exemplo, um grupo de universidades pode decidir juntar seus recursos para construir esse tipo de rede. Essa rede regional, por sua vez, pode ser conectada a uma rede mais global a que outras redes regionais se interliguem. Dessa maneira, a porção da nuvem assume uma estrutura hierárquica (Figura 3.12).

Indivíduos que desejam acessar a Internet podem registrar, implementar e manter os seus próprios domínios. Contudo, é mais comum para um indivíduo ter acesso à Internet por meio de um domínio estabelecido pela organização à qual ele pertence, ou contratando um provedor de serviços (Internet Services Provider — ISP) para se conectar ao domínio estabelecido pelo provedor. Na maioria dos casos, a sua conexão ao provedor é temporária, feita através de linha telefônica.

**Endereçamento na Internet** À cada máquina na Internet é atribuído um único endereço chamado **endereço IP**, usado para identificá-la nas comunicações. Cada endereço IP é um padrão de 32 *bits* que consiste em duas partes: a identificação do domínio ao qual a máquina pertence e uma identificação da máquina dentro desse domínio. A parte do endereço que identifica o domínio, o **identificador da rede**, é atribuído pelo ICANN quando o domínio é estabelecido e registrado. Assim, é por meio desse processo de registro que se garante que cada domínio na Internet possui um único identificador na rede. A parte do endereço que identifica uma máquina particular dentro do domínio é chamada **endereço do hospedeiro** (o termo **hospedeiro** se refere a uma máquina da rede, em reconhecimento ao papel de hospedar as solicitações de outras máquinas). O endereço do hospedeiro é atribuído pela autoridade local do domínio — geralmente uma pessoa cuja função é identificada como administrador de rede ou administrador de sistema. Por exemplo, o identificador da rede da companhia de publicações Addison Wesley é 192.207.177 (os identificadores de redes tradicionalmente são escritos em notação decimal com pontos; veja o Exercício 8 no final da Seção 1.4). No entanto, uma máquina dentro deste domínio terá um endereço como 192.207.177. 133, cujo último *byte* é o endereço do hospedeiro.

Endereços no formato de padrões de *bits* são difíceis para os seres humanos memorizarem. Por isso, a cada domínio é atribuído um endereço mnemônico, conhecido como **nome do domínio**. Por exemplo, o nome do domínio de Addison Wesley é aw.com. Essa classificação é chamada **domínio de alto nível** (*top-level domain* — TLD). Existem muitos TLDs, incluindo `edu` para instituições de educação, `gov` para instituições governamentais, `org` para organizações sem fins lucrativos, `museum` para museus, `info` para uso irrestrito e `net`, que foi originalmente dedicado aos provedores, mas hoje é usado em larga escala. Além desses TLDs gerais, existem TLDs de duas letras para os países (chamados TLD de código do país) como `au` para Austrália e `ca` para Canadá.

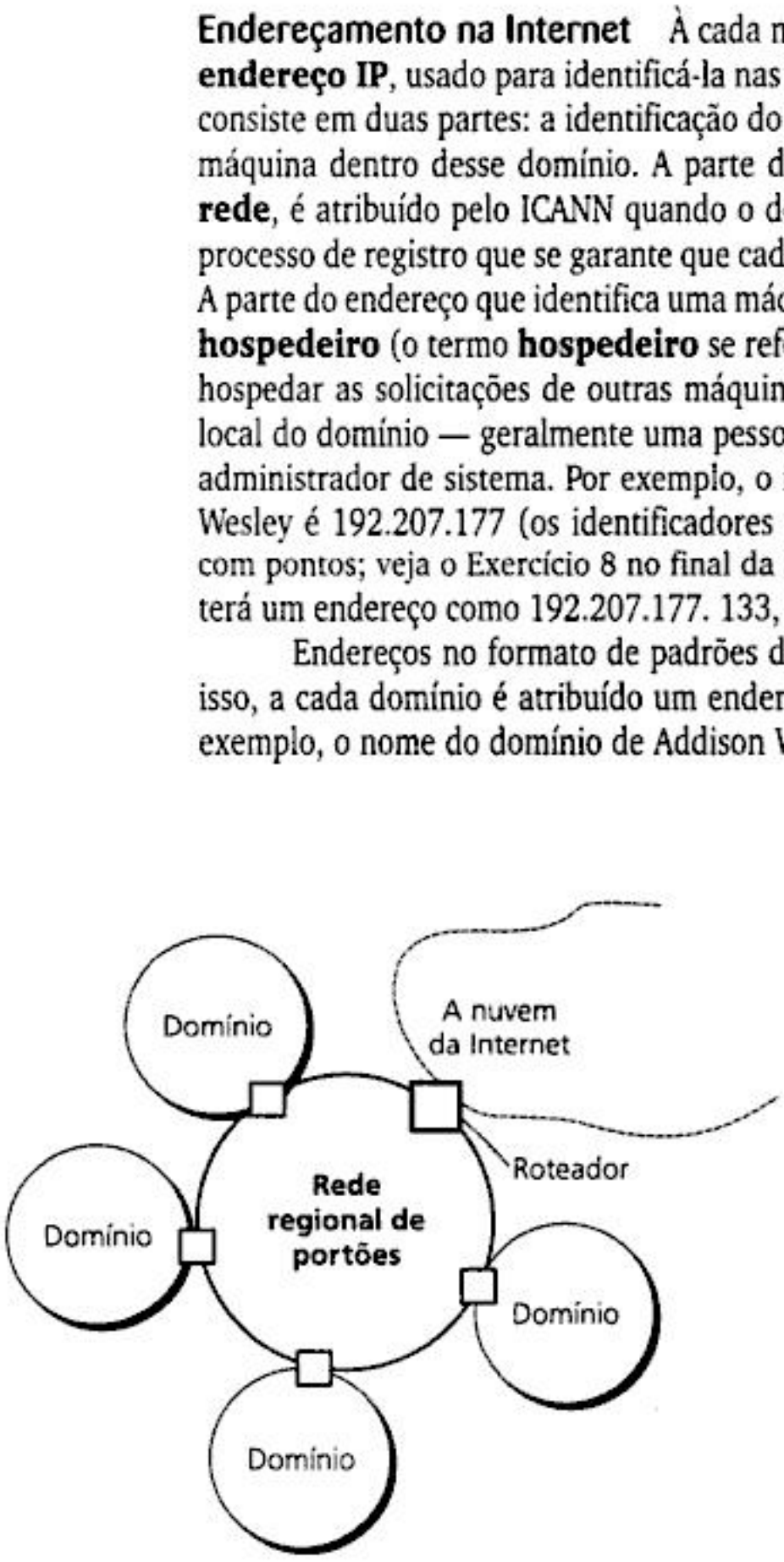



**Figura 3.12** Uma abordagem típica de conexão à Internet.

Uma vez que um domínio tenha um nome mnemônico, sua autoridade local poderá estendê-lo para obter nomes mnemônicos para as máquinas do domínio. Por exemplo, uma máquina individual dentro do domínio *aw.com* pode ser identificada como `ssenterprise.aw.com`.

Devemos enfatizar que a notação decimal com pontos, usada nos endereços mnemônicos, nada tem a ver com a usada para representar endereços no formato de padrões de *bits*. Em vez disso, as seções de um endereço mnemônico

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

cada atividade, incluindo o modo como são enviadas as mensagens, a maneira como a autorização para transmitir mensagens é delegada às máquinas e a forma como são manipuladas as tarefas de compactar e descompactar mensagens para a transmissão. Consideremos primeiramente os protocolos que controlam a autorização de uma máquina para transmitir suas próprias mensagens pela rede.

## Controle dos privilégios de transmissão

Um método para coordenar as autorizações de transmissão de mensagens é o **protocolo *token-ring*** (anel de símbolos) desenvolvido pela IBM em 1970 e que continua sendo um protocolo popular para redes com topologia em anel. Nele, cada máquina só transmite mensagens à sua vizinha à direita e só recebe mensagens da vizinha à esquerda, conforme ilustrado na Figura 3.15. A mensagem de uma máquina para outra deve ser enviada, dentro da rede, sempre no sentido horário, até que alcance o seu destino. Quando ela atinge o seu objetivo, a máquina receptora mantém uma cópia da mensagem e envia outra para percorrer o anel. Quando a cópia enviada atingir a máquina que a originou, esta interpretará que a mensagem alcançou o seu destino, devendo portanto ser retirada do anel. Naturalmente, este sistema depende da cooperação entre essas máquinas. Se cada uma insistir em transmitir constantemente apenas suas mensagens em vez de também enviar as das outras, nenhum trabalho útil será realizado.

Para resolver esse problema, se passa através do anel de comunicação um padrão de *bits*, chamado *símbolo*. As máquinas que apresentarem este mesmo símbolo recebem a autorização para transmitir suas próprias mensagens; sem ele, a máquina fica autorizada apenas a retransmitir as mensagens que recebe. Normalmente, cada máquina apenas passa o símbolo da esquerda para a direita, fazendo o mesmo com as mensagens. Entretanto, se ao receber o símbolo a máquina tiver suas próprias mensagens para ser enviadas à rede, ela as transmitirá, mas reterá o símbolo. Quando as mensagens tiverem completado o seu ciclo ao redor do anel, a máquina devolverá o símbolo para a próxima do anel. Do mesmo modo, quando a próxima máquina receber o símbolo, poderá devolvê-lo imediatamente ou transmitir primeiro a sua própria mensagem, antes de devolver o símbolo para a máquina seguinte. Desta maneira, cada máquina da rede tem igual oportunidade de transmitir suas próprias mensagens enquanto o símbolo permanecer circulando no anel.

Outro protocolo para coordenar a autorização de transmissão foi criado pela Ethernet, que é uma versão popular de uma rede com topologia de via. Na Ethernet, a autorização para transmitir mensagens é controlada por um protocolo conhecido como **CSMA/CD** (*Carrier Sense, Multiple Access with Collision Detection*), o qual estabelece que toda mensagem transmitida por uma máquina deve ser retransmitida para todas as máquinas da via (Figura 3.16). Cada uma pode monitorar todas as mensagens, mas retém apenas aquelas que lhe estiverem endereçadas. Para transmitir uma mensagem, a máquina espera até detectar silêncio na via, quando começará a transmitir, enquanto continua monitorando a via. Se outra máquina também começar a transmitir, ambas detectarão o sinal e aguardarão durante um breve período aleatório, antes de tentar novamente a transmissão. O resultado é um sistema semelhante ao usado por um pequeno grupo de pessoas em uma conversação. Se duas pessoas começam a falar ao mesmo tempo, ambas param. A diferença é que elas podem pedir desculpas, como: "Desculpe, o que você ia dizer?", ou "Nada, nada. Por favor, fale primeiro!", enquanto com o protocolo CSMA/CD cada máquina simplesmente tenta de novo.

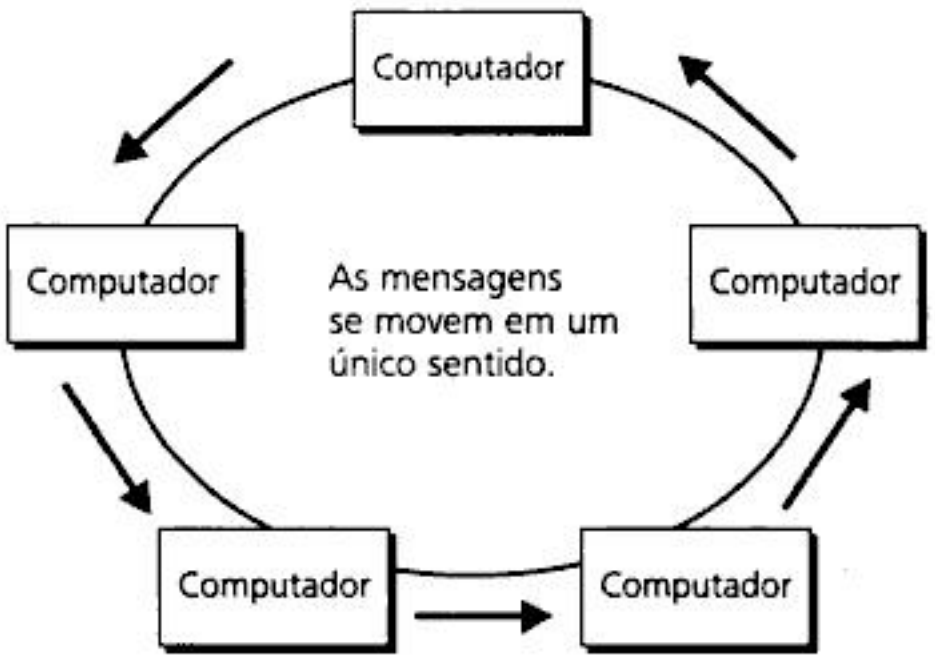



**Figura 3.15** Comunicação com topologia em anel.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

quenos segmentos de mensagem podem se entrelaçar nesses pontos, enquanto uma mensagem longa força os outros a esperar até que ela passe (similar aos carros que esperam um longo trem passar pelo cruzamento).

A camada de transporte adiciona números de seqüência a esses pequenos segmentos que ela produz, de modo que eles podem ser reagrupados no destino da mensagem. Então, anexa o endereço de destino a cada segmento e entrega esses segmentos endereçados, conhecidos como **pacotes** (*packets*) à camada de rede. A partir desse ponto, os pacotes são tratados individualmente, sem relação com as mensagens, até que atinjam a camada de transporte do destino. É possível que os pacotes de uma mensagem sigam caminhos diferentes ao longo da Internet.

A camada de rede é responsável por verificar se os pacotes recebidos são repassados de uma rede da Internet para outra até que atinjam seu destino. Assim, é a camada de rede que deve lidar com a topologia da Internet. Se o caminho de um pacote específico através da Internet deve passar por muitas redes individuais, é a camada de rede em cada parada intermediária que determina a direção na qual o pacote dever ser enviado. Isso é feito acrescentando-se um endereço destinatário intermediário a cada pacote, de acordo com as seguintes normas: "Se o destino final do pacote estiver dentro da rede corrente, o endereço anexado será uma duplicata do endereço final do destinatário; caso contrário, o endereço anexado será o do roteador da rede corrente, por onde o pacote deve passar para ser transferido para uma rede adjacente." Deste modo, o pacote destinado a uma máquina dentro da rede corrente será enviado a ela, e o pacote destinado a uma máquina externa a ela continuará a sua jornada rede a rede.

Uma vez determinado o destino intermediário do pacote, a camada de rede anexa esse endereço ao pacote e o entrega à camada de ligação.

A responsabilidade da camada de ligação é a de lidar com os detalhes da comunicação com a rede específica em que a máquina se encontra. Se esta rede for do tipo anel de símbolos, a camada de ligação terá de aguardar o recebimento do símbolo antes de realizar sua transmissão. Se a rede usar um protocolo CSMA/CD, a camada de ligação terá de esperar por um silêncio na via antes de começar a transmitir. Além disso, cada rede individual dentro da Internet possui o seu próprio sistema de endereços, que é independente do sistema usado pela Internet. Afinal de contas, muitas dessas redes já funcionavam por si próprias bem antes de se associarem à Internet. Assim, a camada de ligação terá de traduzir os endereços da Internet anexados aos pacotes para um sistema local apropriado de endereços.

Cada vez que um pacote é transmitido, ele é recebido pela camada de ligação da máquina receptora. É então enviado à camada de rede, onde o destinatário final é comparado com a localização corrente. Se não coincidirem, a camada de rede anexará um novo endereço intermediário ao pacote e o devolverá à camada de ligação para retransmissão. Dessa maneira, cada pacote salta de uma máquina para outra a seu modo, até o seu destino final. Note que apenas as camadas de ligação e de rede estão envolvidas nas paradas intermediárias (veja novamente a Figura 3.19).

Se a camada de rede constatar que um pacote recebido alcançou seu destino final, ela o enviará à camada de transporte. Ao receber os pacotes da camada de rede, a de transporte extrai os segmentos e reconstitui a mensagem original de acordo com os números de seqüência que foram fornecidos pela camada de transporte na máquina de origem. Uma vez que a mensagem esteja completa, será enviada à camada de aplicação — completando assim o processo de transmissão de mensagens.

A determinação de qual aplicação deve receber uma nova mensagem é uma tarefa importante da camada de transporte. Isto é feito atribuindo um número único de porta (nada a ver com as portas de E/S discutidas no Capítulo 2) às várias unidades de aplicação e exigindo que a aplicação que envia a mensagem coloque o número apropriado da porta no endereço da mensagem antes que esta inicie a sua jornada. Assim, quando a mensagem tiver sido recebida pela camada de transporte do destino, esta necessitará apenas enviar a mensagem ao *software* de aplicação instalado no número designado da porta.

Os usuários da Internet raramente precisam se preocupar com números de porta, uma vez que as aplicações comuns possuem números aceitos universalmente. Por exemplo, se um navegador Web for solicitado recuperar o documento cujo URL é `http://www.zoo.org/animals/frog.html` ele saberá que o contato com o servidor HTTP em `www.zoo.org` é feito pela porta de número 80. De modo

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

net tem acesso autorizado à informação que está sendo comunicada aos seus clientes? Até que ponto um provedor é responsável pelo conteúdo da comunicação entre seus clientes? Tais questões vêm desafiando a comunidade jurídica atual.

Nos Estados Unidos, muitas dessas questões são remetidas ao Electronic Communication Privacy Act (ECPA) de 1986, o qual se originou na legislação para controlar a escuta telefônica. Embora a lei seja extensa, seu escopo é capturado em alguns pequenos trechos. Em particular, ela declara que:

> Exceto nos casos especificamente mencionados neste capítulo, qualquer pessoa que intencionalmente interceptar, procurar interceptar, ou que mandar outra pessoa interceptar ou procurar interceptar qualquer comunicação oral, por fio ou eletrônica... deve ser punida conforme a subseção (4) ou sujeita à ação conforme a subseção (5)
>
> e
>
> ... qualquer pessoa ou entidade que forneça serviço de comunicação eletrônica ao público não deve intencionalmente divulgar o conteúdo de qualquer comunicação nesse serviço a qualquer pessoa ou entidade que não o endereçado ou receptor de tal comunicação ou o agente do endereçado ou do receptor.

Em resumo, o ECPA confirma o direito individual à comunicação privada — é ilegal a pessoa desautorizada se meter em comunicação alheia, e é ilegal um provedor da Internet liberar a informação que diga respeito à comunicação entre seus clientes. Contudo, a lei também declara o seguinte:

> É permitido a um funcionário, empregado ou agente da Comissão Federal de Comunicações em seu trabalho normal, na conclusão do monitoramento responsável exercido pela Comissão para a garantia do Capítulo 5 do título 47 da Constituição, interceptar uma comunicação oral, por fio ou eletrônica ou comunicação oral transmitida por rádio, ou abrir e usar a informação obtida.

Assim, o ECPA explicitamente dá à Comissão Federal de Comunicações o direito de monitorar a comunicação eletrônica sob certas circunstâncias. Isso leva a alguns tópicos complicados. Primeiro, para que a Comissão possa exercer o seu direito garantido pelo ECPA, os sistemas de comunicação devem ser construídos e programados de forma que a comunicação possa ser monitorada. O estabelecimento dessa capacidade é considerado no Communications Assistence for Law Enforcement Act — CALEA. Ele exige que as companhias de telecomunicações modifiquem os seus equipamentos para se ajustarem à lei. Contudo, a implementação desse ato mostrou-se complexa e cara, resultando em um adiamento do prazo para a sua efetivação.

Um tópico ainda mais controverso envolve o choque entre o direito da Comissão de monitorar as comunicações e o direito do público de usar criptografia. Afinal de contas, se as mensagens monitoradas são bem criptografadas, então a intromissão na comunicação pelas agências legais de nada adianta. Os governos nos Estados Unidos, Canadá e Europa estão considerando sistemas que exigiriam o registro de chaves de criptografia (ou talvez chaves para as chaves). Entretanto, vivemos em um mundo onde a espionagem feita por uma corporação se tornou tão significativa quanto a militar. Assim, é compreensível que a exigência de registrar chaves de criptografia causaria desconforto a corporações e cidadãos idôneos. Até que ponto o sistema de registro é seguro?

O problema do vandalismo é exemplificado pela ocorrência de danos causados por vírus de computador e vermes de rede. Em geral, o **vírus** é um trecho de programa que se anexa a outros programas do sistema. Por exemplo, ele pode se inserir como prefixo de um programa do sistema, e assim, toda vez que esse programa hospedeiro for ativado, o vírus será automaticamente executado antes. Este vírus, por sua vez, pode praticar atos maliciosos prontamente constatáveis, ou apenas procurar outros programas, para neles instalar outras réplicas de si mesmo. Se um programa infectado for transferido para um novo computador, por meio da rede ou de disquetes, o vírus nele contido começará a infectar os programas existentes neste outro computador, assim que o programa recém-transferido for ativado. Desta maneira, o vírus se transmite de um computador para outro. Em alguns casos, os vírus são planejados inicialmente

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

de nível 1, poderá requisitar todos os de nível 2, e assim por diante. Há possibilidade de ocorrer um enlace mortal neste sistema? Justifique sua resposta.

***39.** Cada braço de um robô está programado para retirar conjuntos montados sobre a esteira de uma linha de montagem, testá-los quanto às suas tolerâncias e depositá-los em uma de duas caixas, de acordo com os resultados desse teste. As peças chegam, uma de cada vez, guardando entre si um intervalo suficiente de tempo. Para impedir que os dois braços tentem agarrar uma mesma peça, os computadores que os controlam compartilham uma mesma posição de memória. Se um braço estiver disponível quando houver a aproximação de um conjunto, o computador de controle deste braço lerá o valor contido na posição de memória comum. Se este valor for diferente de zero, o braço deixará passar o conjunto. Caso contrário, o computador de controle irá colocar um valor diferente de zero nessa posição de memória, instruir o braço a apanhar o conjunto e recolocar o valor 0 nessa posição de memória após completar a ação. Que seqüência de eventos poderia conduzir a uma disputa entre os dois braços?

***40.** O seguinte problema "filósofos durante a janta", foi originalmente proposto por E. W. Dijkstra e atualmente faz parte do folclore da Ciência da Computação.

Cinco filósofos estão sentados ao redor de uma mesa circular. À frente de cada um, está um prato de macarrão. Existem cinco garfos sobre a mesa, cada um entre dois pratos. Cada filósofo deseja alternar entre pensar e comer. Para comer, o filósofo requer a posse dos dois garfos que estão adjacentes a seu prato. Identifique a possibilidade de enlace mortal e de inanição presentes no problema do jantar dos filósofos.

***41.** Suponha que cada computador em uma rede em anel seja programado para transmitir simultaneamente, nas duas direções, as mensagens originadas naquele nó que estejam endereçadas a todos os outros nós pertencentes à rede. Além disso, suponha que isto seja feito obtendo-se inicialmente acesso à comunicação com a máquina situada à esquerda do nó, bloqueando-se o uso desse caminho enquanto o acesso à trajetória de comunicação com a máquina à direita do nó não for obtido, e para só então transmitir a mensagem. Identifique o enlace mortal que ocorrerá se todas as máquinas da rede tentarem emitir tal mensagem simultaneamente.

***42.** Identifique o uso de uma fila no processo de *spooling* — envio de saídas — para uma impressora.

***43.** O cruzamento de duas ruas pode ser considerado como um recurso não-compartilhável, pelo qual competem os carros que dele se aproximam. Um semáforo é usado, em vez de um sistema operacional, para controlar a alocação desse recurso. Se o equipamento for capaz de quantificar o fluxo de tráfego que chega de cada direção e programado para sempre conceder sinal verde ao tráfego mais pesado, o tráfego mais leve poderia sofrer do que se chama inanição. Qual o significado desse termo? O que poderia acontecer em um sistema computacional multiusuário, no qual as rotinas são atendidas segundo a sua prioridade, se a competição pelos recursos fosse sempre resolvida estritamente com base na prioridade?

***44.** Quais problemas poderão ocorrer em um sistema de tempo partilhado se o despachante sempre designar intervalos de tempo de acordo com um sistema de prioridades, segundo o qual a prioridade de cada tarefa seja sempre a mesma? (Sugestão: de acordo com a relação entre a prioridade da rotina que acabou de utilizar o intervalo de tempo a que tinha direito, e a das rotinas que estão aguardando, qual deverá ser a rotina a receber o próximo intervalo de processamento?)

***45.** Qual a semelhança entre enlace mortal e inanição? (Consulte o Problema 44.) Qual a diferença entre esses dois conceitos?

***46.** Que problema surgirá se, em um sistema de tempo partilhado, a duração dos intervalos de tempo for sendo reduzida progressivamente? E se tais intervalos fossem aumentados gradativamente?

***47.** Com o desenvolvimento da Ciência da Computação, as linguagens de máquina têm sido estendidas para prover instruções especializadas. Três dessas instruções de máquina que foram introduzidas na Seção 3.4 são usadas extensivamente pelos sistemas operacionais. Quais são elas?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# 4.1 O conceito de algoritmo

Na introdução, definimos informalmente algoritmo como um conjunto de passos que define a maneira como uma tarefa deve ser executada. Nesta seção, analisaremos mais de perto este conceito básico.

## Uma revisão informal

Já encontramos muitos algoritmos em nosso estudo. Vimos algoritmos para converter representações de números de uma forma para outra, detectar e corrigir erros nos dados, compactar e descompactar arquivos de dados, controlar o compartilhamento de tempo em ambientes multitarefa e muitos mais. Além disso, vimos como eles podem ser expressos em uma linguagem de máquina e executados por uma máquina, cuja UCP desempenha as suas tarefas seguindo o algoritmo:

Enquanto a instrução de parada não for executada, continue a executar os seguintes passos:

a. Busque a próxima instrução.
b. Decodifique a instrução.
c. Execute a instrução.

Como demonstrado pelo algoritmo que descreve um truque de mágica na Figura 0.1, os algoritmos não se restringem às atividades técnicas. Na verdade, eles estão subjacentes mesmo em atividades rotineiras, como descascar ervilhas:

Obtenha uma cesta de ervilhas com casca e uma tigela vazia.
Enquanto existirem ervilhas na cesta, continue a executar os seguintes passos:

a. Pegue uma ervilha da cesta.
b. Abra a casca da ervilha.
c. Tire a ervilha da casca e coloque na tigela.
d. Jogue fora a casca.

De fato, muitos pesquisadores acreditam que qualquer atividade da mente humana, inclusive imaginação, criatividade e tomada de decisão, é resultado da execução de algoritmo — conjectura que abordaremos novamente em nosso estudo de inteligência artificial (Capítulo 10).

Entretanto, antes de prosseguir, consideremos a definição formal de algoritmo.

## A definição formal de algoritmo

Conceitos informais, fracamente definidos, são aceitáveis e comuns na vida diária, mas uma ciência deve se basear em terminologia bem-definida. Considere então a definição formal de algoritmo apresentada na Figura 4.1.

Note que a definição exige que o conjunto de passos em um algoritmo seja ordenado. Isso significa que os passos em um algoritmo devem possuir uma estrutura bem estabelecida em termos da ordem na qual são executados. Isso não implica que os passos sejam executados em seqüência consistam no primeiro passo, seguido do segundo e assim por diante. Alguns algoritmos, conhecidos como algoritmos paralelos, por exemplo, apresentam mais de uma sucessão de passos possível, cada qual projetada para ser executada por processadores diferentes em uma máquina com multiprocessamento. Em tais casos, o algoritmo como um todo não apresenta uma linha única de passos, em que possa ser identificada uma seqüência autêntica. Em vez disso, apresenta uma estrutura de várias ramificações e reconexões simultâneas de processamento, cada uma responsável pela execução de uma das diferentes partes do algoritmo. (Voltaremos a esse conceito na Seção 6 do Capítulo 5.) Outros exemplos incluem algoritmos executados por circuitos, como no caso do *flip-flop* da Seção 1.1, no qual cada porta lógica executa um único passo do algoritmo global. A ordem em que os passos são executados decorre de relações de causa e efeito, à medida que a saída de cada porta lógica se propaga pelo circuito.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

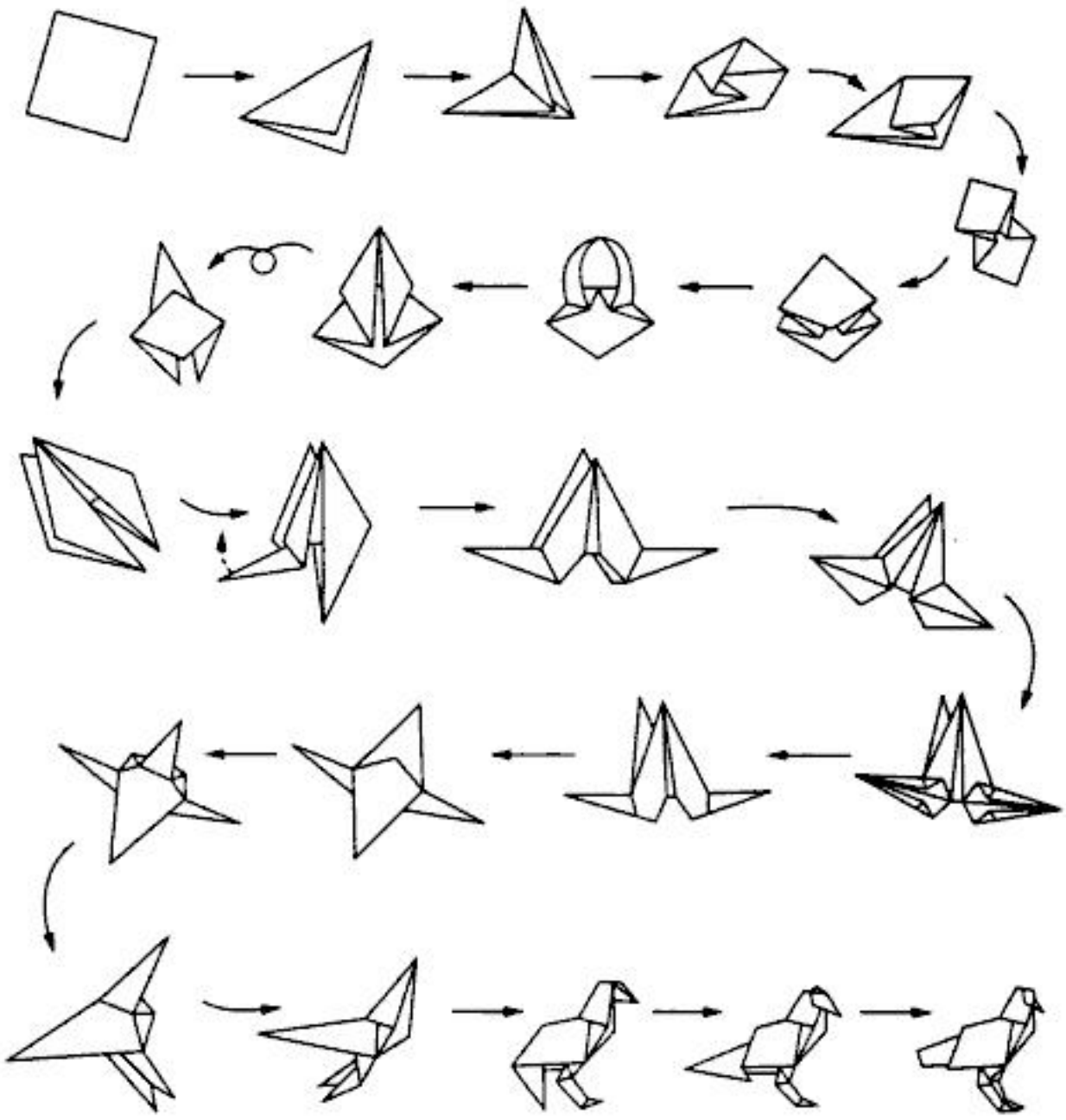


**Figura 4.2** Construção de um pássaro pela dobradura de um pedaço quadrado de papel.

## Pseudocódigo

Por ora, vamos adiar a apresentação de uma linguagem formal de programação, em favor de um sistema notacional menos formal e mais intuitivo, conhecido como pseudocódigo. Em geral, um **pseudocódigo** é um sistema notacional no qual as idéias podem ser informalmente expressas durante o processo de desenvolvimento do algoritmo.

Uma forma de obtenção de um pseudocódigo consiste em simplesmente relaxar as regras da linguagem formal na qual a versão final do algoritmo deverá ser expressa. Esta prática é adotada sempre que a linguagem de programação a ser usada para a codificação for conhecida de antemão. Neste caso, o pseudocódigo, utilizado durante os estágios iniciais do desenvolvimento do programa, compõe-se de estruturas sintático-semânticas semelhantes às empregadas pela linguagem de programação adotada, embora menos formais.

### *Nomenclatura de itens dos programas*

Em uma linguagem natural, os itens freqüentemente possuem nome com múltiplas palavras, tais como "custo de produção de um recurso" ou "tempo estimado de chegada". Contudo, quando expressamos algoritmos em um programa formal ou em uma versão do programa em pseudocódigo, os nomes com múltiplas palavras podem complicar a descrição de um algoritmo. A experiência tem mostrado que é melhor ter cada item identificado por um único bloco de texto. Ao longo dos anos, muitas técnicas foram usadas para comprimir múltiplas palavras em uma única unidade léxica, obtendo nomes descritivos para itens nos programas. Uma delas é usar o símbolo de sublinhado para conectar palavras, produzindo nomes como `tempo_estimado_de_chegada`. Outra é usar letras maiúsculas para auxiliar um leitor a compreender um nome composto de múltiplas palavras. Por exemplo, pode-se começar cada palavra com uma letra maiúscula, obtendo nomes como `TempoEstimadoChegada`. Essa técnica freqüentemente é chamada notação Pascal, porque foi popularizada pelos usuários da linguagem de programação Pascal. Uma variação da notação Pascal é a notação Camelo, idêntica à Pascal, exceto em que a primeira letra permanece minúscula, como em `tempoEstimadoChegada`. Este texto segue a notação Pascal, mas a escolha é questão de gosto pessoal.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Convém recordar que o objetivo do nosso pseudocódigo é o de fornecer meios para representar algoritmos de uma maneira legível e informal. Queremos um sistema de notação que nos ajude a expressar idéias — sem nos tornarmos escravos de regras formais e rigorosas. Assim, estaremos livres para expandir ou modificar nosso pseudocódigo quando necessário. Em particular, se as instruções dentro de um conjunto de parênteses envolverem outras instruções dentro de parênteses, poderá ficar difícil emparelhar visualmente parênteses que abrem com os que fecham. Nesses casos, muita gente acha útil complementar um parêntese que está se fechando com um comentário explicativo de que a instrução ou frase está sendo terminada. Especificamente, pode-se complementar o parêntese final de uma instrução *enquanto* com as palavras "fim do enquanto" produzindo uma instrução como

```
enquanto (...) faça
 (.
  .
  .
 ) fim do enquanto
```

ou talvez

```
enquanto (...) faça
 (se (...)
   então ( .
           .
           .
         ) fim do se
 ) fim do enquanto
```

O fato é que estamos tentando expressar um algoritmo em uma forma legível, e então podemos introduzir auxílios visuais (endentação, comentários etc.) para atingir esta meta. Além disso, se encontramos um tema recorrente que ainda não foi incorporado ao pseudocódigo, podemos decidir pela extensão do nosso pseudocódigo, adotando um sistema consistente para representar o novo conceito.

## *QUESTÕES/EXERCÍCIOS*

1. Uma primitiva em um dado contexto pode tornar-se, em outro contexto, uma combinação de primitivas. Por exemplo, a nossa instrução *Enquanto* é uma primitiva em nosso pseudocódigo, embora seja implementada como uma combinação de instruções de máquina. Dê dois exemplos deste fenômeno, em situações nas quais não sejam empregados computadores.

2. Em que sentido a construção de procedimentos corresponde à de primitivas?

3. O algoritmo de Euclides calcula o máximo divisor comum de dois inteiros positivos $X$ e $Y$ pelo seguinte processo:
   Enquanto nenhum dos valores de $X$ e $Y$ for zero, continuar dividindo o maior deles pelo menor, atribuindo para $X$ e $Y$ os valores do divisor e do resto, respectivamente. (O valor final de $X$ será o máximo divisor comum que se deseja calcular.)
   Expresse este algoritmo em nosso pseudocódigo.

4. Descreva um conjunto de primitivas que seja utilizado em uma área que não seja a programação de computadores.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

É claro que ser aconselhado a subir o primeiro degrau não é o mesmo que ser instruído acerca de como isso deve ser feito. Alcançar este ponto de apoio, bem como perceber como ampliar este ponto inicial de apoio até a obtenção de uma solução completa do problema, requer iniciativas criativas da parte do possível solucionador do problema. Há contudo diversos enfoques gerais, propostos por Polya e outros, sobre como encontrar tal ponto de partida. Deve-se tentar trabalhar o problema *de marcha a ré*. Por exemplo, se ele consiste em encontrar um modo de produzir uma certa saída a partir de uma dada entrada, pode-se partir desta saída e tentar percorrer o caminho de volta à entrada fornecida. Esta abordagem é tipicamente adotada por alguém que tente descobrir o algoritmo de construção do pássaro em dobradura de papel, mostrado na seção anterior. Ele tende a desfazer as dobras do pássaro acabado, em uma tentativa de observar de que maneira foi ele construído.

Outro enfoque para a resolução geral de problemas é procurar um problema relacionado com o que nos interessa no momento, que seja mais fácil de resolver ou já tenha sido resolvido antes, e então tentar solucionar o problema usando a mesma forma de resolução. Esta técnica é especialmente valiosa no contexto do desenvolvimento de programas. Freqüentemente, a maior dificuldade encontrada no desenvolvimento de programas não é resolver uma instância particular de um problema, mas encontrar um algoritmo geral, que possa ser empregado para resolver todas as instâncias desse problema. Mais precisamente, se estivermos diante da tarefa de desenvolver um programa encarregado de ordenar alfabeticamente listas de nomes, então a nossa tarefa não será a de ordenar uma determinada lista, mas a de encontrar um algoritmo geral capaz de ordenar qualquer lista de nomes. Assim, embora as instruções seguintes ordenem corretamente a lista David, Alice, Carol, Bob, elas não constituem o algoritmo de propósito geral que desejamos:

> Trocar de posição os nomes *David* e *Alice*.
> Mover o nome *Carol* para a posição entre os nomes *Alice* e *David*.
> Mover o nome *Bob* para a posição entre os nomes *Alice* e *Carol*.

O que precisamos é de um algoritmo que ordene esta lista, bem como quaisquer outras que encontremos. Isso não significa que a nossa solução para ordenar uma lista particular seja totalmente desprezível na nossa procura por um algoritmo de propósito geral. Por exemplo, estaríamos colocando nosso pé na porta ao considerar tais casos particulares, em uma tentativa de determinar princípios gerais que possam, por sua vez, ser usados para desenvolver o algoritmo de propósito geral desejado. Nesse caso, então, a nossa solução será obtida pela técnica da resolução de um conjunto de problemas relacionados.

Pode-se também aplicar a técnica do **refinamento sucessivo**, que consiste essencialmente em não tentar realizar imediatamente uma tarefa inteira, em todos os seus detalhes. A técnica propõe que primeiro se decomponha o problema em vários subproblemas. A idéia é que, dividindo o problema original pode-se visualizar a solução global desejada composta de uma série de passos. A técnica do refinamento sucessivo propõe que os passos assim construídos sejam, por sua vez, decompostos em passos menores, e estes, em outros ainda menores, e assim por diante, até que o problema inteiro seja reduzido a um conjunto de subproblemas cujas soluções possam ser facilmente obtidas.

Seguindo esta linha, o refinamento sucessivo constitui uma metodologia cima-baixo (*top-down*), na qual o processo de desenvolvimento leva do geral para o particular. A metodologia baixo-cima (*bottom-up*) leva do específico para o geral. Embora teoricamente contrastantes, na prática, os dois enfoques se complementam. Por exemplo, a decomposição de um problema, proposta pelo refinamento da metodologia cima-baixo, freqüentemente é guiada pela intuição do solucionador de problema, a qual opera de forma baixo-cima.

As soluções produzidas pela técnica do refinamento sucessivo apresentam uma estrutura modular natural, e é esta a razão principal para a sua popularidade em projetos de algoritmos. Se um algoritmo tem uma estrutura modular natural, pode ser facilmente adaptado a uma representação modular, o que conduz ao desenvolvimento de um programa gerenciável. Além disso, os módulos produzidos pelo refinamento sucessivo são compatíveis com o conceito de programação em equipe, no qual diversas pessoas

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
procedimento Busca (Lista,ValorDesejado);
se (Lista vazia)
    então
      (informar que a busca fracassou)
    senão
      (Selecionar como elemento em teste o primeiro elemento da lista;
       enquanto (ValorDesejado > elemento em teste e
                 ainda há elementos a considerar)
           faça (Selecionar como elemento em teste o próximo da lista.);
       se (ValorDesejado = elemento em teste)
           então (informar que a busca foi bem-sucedida.)
           senão (informar que a busca fracassou.)
       ) fim do se
```

**Figura 4.6** O algoritmo de busca seqüencial em pseudocódigo.

**enquanto** (*condição*) **faça** (*corpo*)

exemplifica o conceito de uma estrutura iterativa, cuja execução mostra um padrão cíclico

testar a *condição*
executar o *corpo*
testar a *condição*
executar o *corpo*
.
.
.
testar a *condição*

até que se torne *falso* o valor testado da *condição*.

Como regra geral, o uso de uma estrutura iterativa proporciona maior flexibilidade que simplesmente escrever várias vezes o corpo de tal estrutura. Por exemplo, embora a estrutura iterativa

Executar três vezes a instrução "Adicionar uma gota de ácido sulfúrico".

seja equivalente à seqüência:

Adicionar uma gota de ácido sulfúrico.
Adicionar uma gota de ácido sulfúrico.
Adicionar uma gota de ácido sulfúrico.

não podemos produzir uma seqüência semelhante que seja equivalente à instrução iterativa denotada por:

**enquanto** (o nível de pH for maior que 4) **faça**
(adicionar uma gota de ácido sulfúrico)

porque não sabemos com antecedência quantas gotas de ácido serão necessárias.

Analisemos, agora mais de perto, a composição do controle do laço. Pode-se ficar tentado a dar pouca importância a essa parte da estrutura iterativa, uma vez que é o corpo da iteração que de fato executa a tarefa desejada (por exemplo, adicionar gotas de ácido) — as atividades de controle da iteração aparecem como uma sobrecarga, simplesmente porque decidimos executar de modo iterativo o corpo da iteração. No entanto, a experiência mostra que o controle das iterações é a parte desta estrutura que se mostra mais sujeita a erros, merecendo, portanto, uma atenção especial.

O controle de um laço consiste em três atividades: iniciação, teste e modificação (Figura 4.7), sendo necessária a presença dos três componentes para que haja sucesso nesse controle. A atividade de teste deve propiciar o encerramento da iteração ao detectar uma condição que sinalize o término da operação. Esta condição é conhecida como condição terminal. É com este propósito que existe uma atividade de teste em cada instrução enquanto do nosso pseudocódigo. No caso da instrução enquanto, contudo, a condição declarada determina a execução do corpo do laço — a condição terminal é a negação da condição que aparece na estrutura enquanto. Assim, na instrução enquanto da Figura 4.6, a condição terminal é:

(ValorDesejado ≤ elemento em teste) ou (não existem mais elementos a considerar)

As outras duas atividades de controle da iteração asseguram que a condição terminal venha de fato a ocorrer. O passo de iniciação estabelece um ponto de partida para a iteração, e o passo de modificação altera esta condição até que seja atingido o seu término. Por exemplo, na Figura 4.6, a iniciação acontece na instrução que precede a instrução enquanto, onde se estabelece que o elemento em teste é o

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

contrário, poderemos ao menos restringir o processo de busca à primeira ou à última metade da lista, dependendo de o valor procurado ser menor ou maior que o elemento do meio. (Lembre-se de que a lista está ordenada.)

Para pesquisar o restante da lista, poderíamos aplicar a busca seqüencial, mas em vez disso, apliquemos à parte da lista o mesmo método que foi usado para a lista inteira, ou seja, selecionamos o elemento do meio na parte restante da lista como o próximo a ser considerado. Como antes, se ele for o procurado, terminamos a busca. Caso contrário, restringiremos nossa pesquisa a uma parte ainda menor da lista.

Essa abordagem ao processo de busca é ilustrada na Figura 4.12, na qual consideramos a tarefa de pesquisar a lista à esquerda da figura à procura do elemento John. Inicialmente, consideremos o elemento do meio, Harry. Uma vez que o elemento procurado é maior, a pesquisa continua considerando a metade inferior da lista original. O elemento do meio da lista é Larry. Como o procurado precede Larry, voltamos nossa atenção à primeira metade da sublista corrente. Quando interrogamos o elemento do meio dessa sublista secundária, encontramos o procurado, John, e declaramos que a pesquisa foi bem-sucedida. Em síntese, nossa estratégia é dividir a lista em questão sucessivamente em segmentos menores até que o elemento procurado seja encontrado ou que a pesquisa seja reduzida a um segmento vazio.

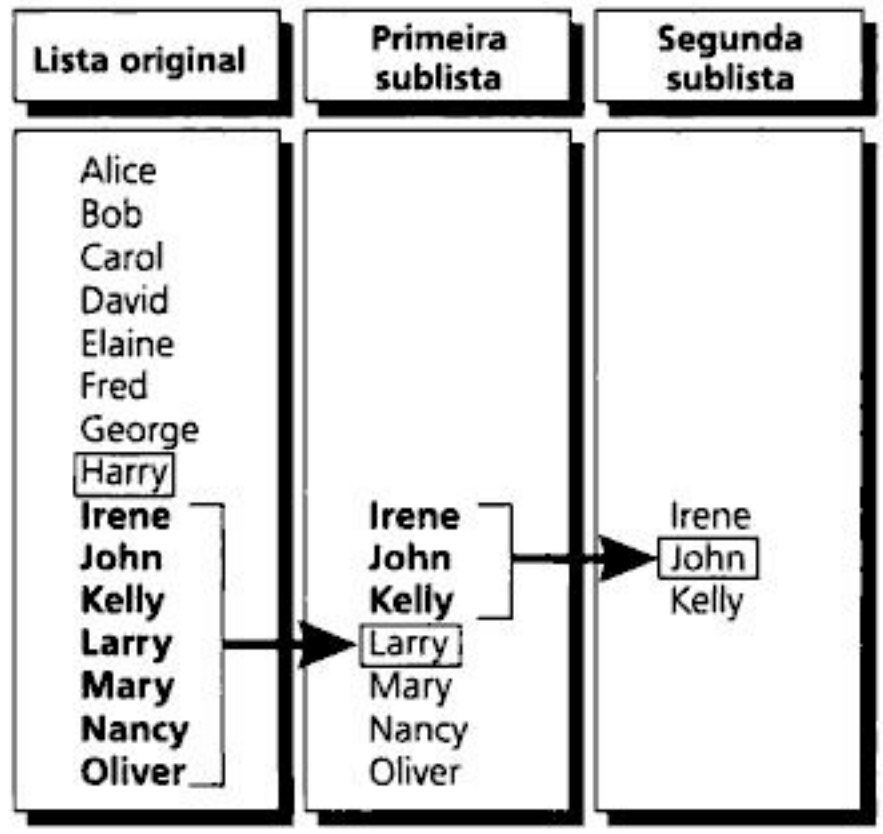


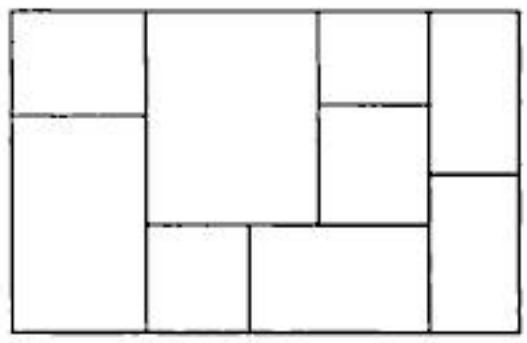

**Figura 4.12** Aplicação da nossa estratégia para pesquisar a lista à procura do elemento John.

É preciso enfatizar este último ponto. Se o elemento procurado não constar na lista original, nosso método de pesquisa irá prosseguir dividindo a lista em segmentos menores até que o segmento a ser considerado esteja vazio. Nesse ponto, o nosso algoritmo deverá reconhecer que a pesquisa fracassou.

A Figura 4.13 é um primeiro esboço de nossos pensamentos usando o nosso pseudocódigo. Ele indica que devemos começar uma pesquisa testando se a lista está vazia. Nesse caso, devemos declarar que a pesquisa fracassou. Caso contrário, consideraremos o elemento do meio da lista. Se ele não for o procurado, pesquisaremos na primeira ou na segunda metade da lista. As duas possibilidades requerem uma pesquisa

## *Estruturas recursivas na arte*

O seguinte procedimento recursivo pode ser aplicado a uma tela retangular para produzir desenhos no estilo do pintor holandês Piet Mondrian (1872-1944), que fazia quadros nos quais o painel retangular era dividido sucessivamente em retângulos menores. Tente seguir o procedimento para produzir desenhos similares ao mostrado à direita. Comece aplicando o procedimento a um retângulo que represente a tela na qual você está trabalhando.



```
procedimento Mondrian (Retângulo)
se (o tamanho do Retângulo for muito grande para o seu gosto artístico)
    então (dividir o Retângulo em dois retângulos menores;
        aplicar o procedimento Mondrian a um dos retângulos menores;
        aplicar o procedimento Mondrian ao outro retângulo menor)
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

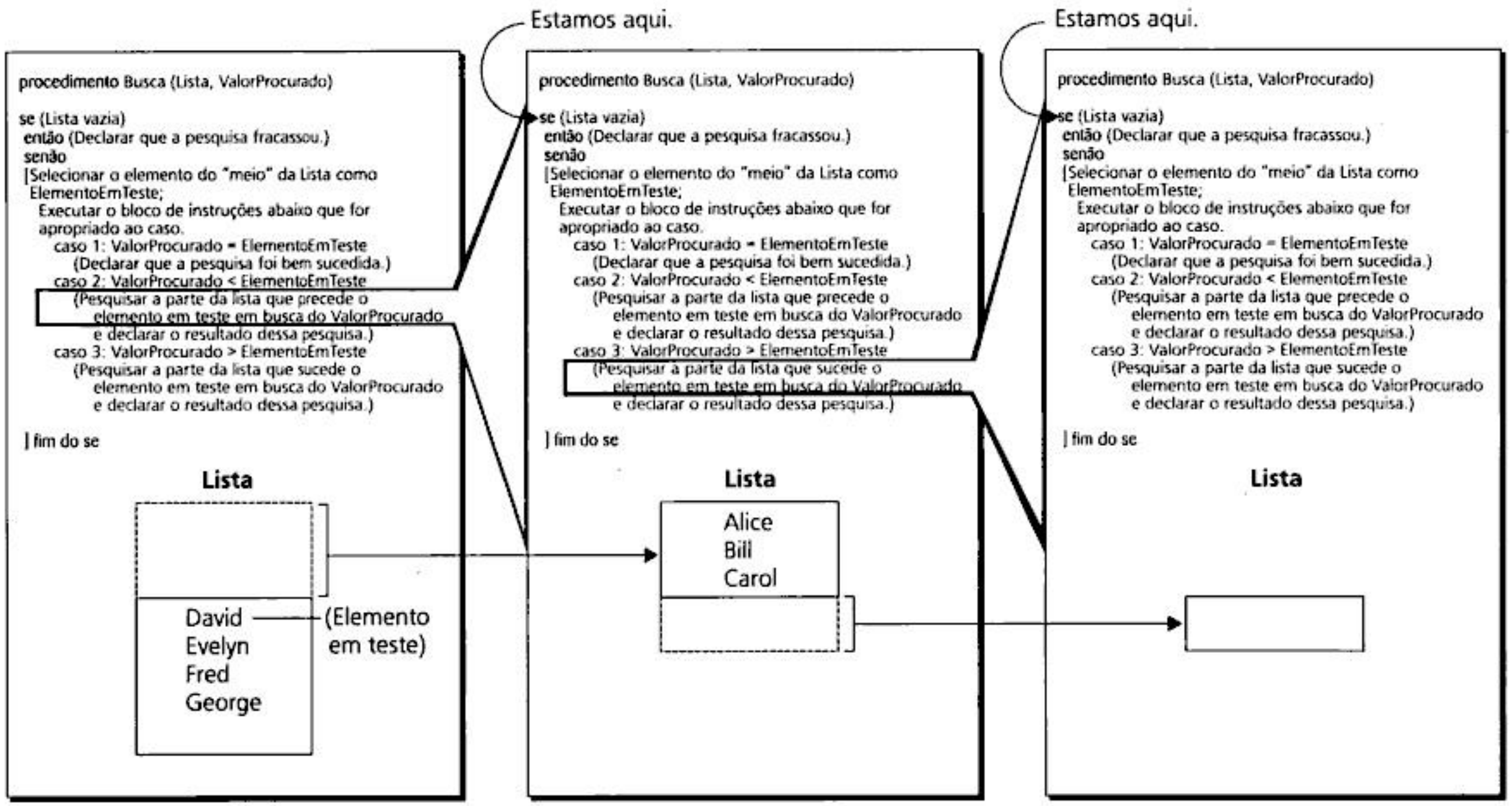



**Figura 4.17**

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

tos cada vez maiores. Assim, o algoritmo vai se tornando menos eficiente à medida que o tamanho da lista aumenta.

Vamos aplicar uma análise similar ao algoritmo de busca binária. Lembre-se de que concluímos que pesquisar uma lista com $n$ elementos usando esse algoritmo envolve a consulta de, no máximo, lg $n$ elementos, o que, mais uma vez, dá uma aproximação do tempo necessário para executar o algoritmo em listas de vários comprimentos. A Figura 4.20 mostra um gráfico baseado nessa análise, no qual novamente marcamos vários comprimentos de lista igualmente espaçados e identificamos o tempo gasto pelo algoritmo em cada caso. Note que o tempo aumenta com incrementos cada vez menores, isto é, o algoritmo de busca binária torna-se mais eficiente à medida que o tamanho da lista aumenta.

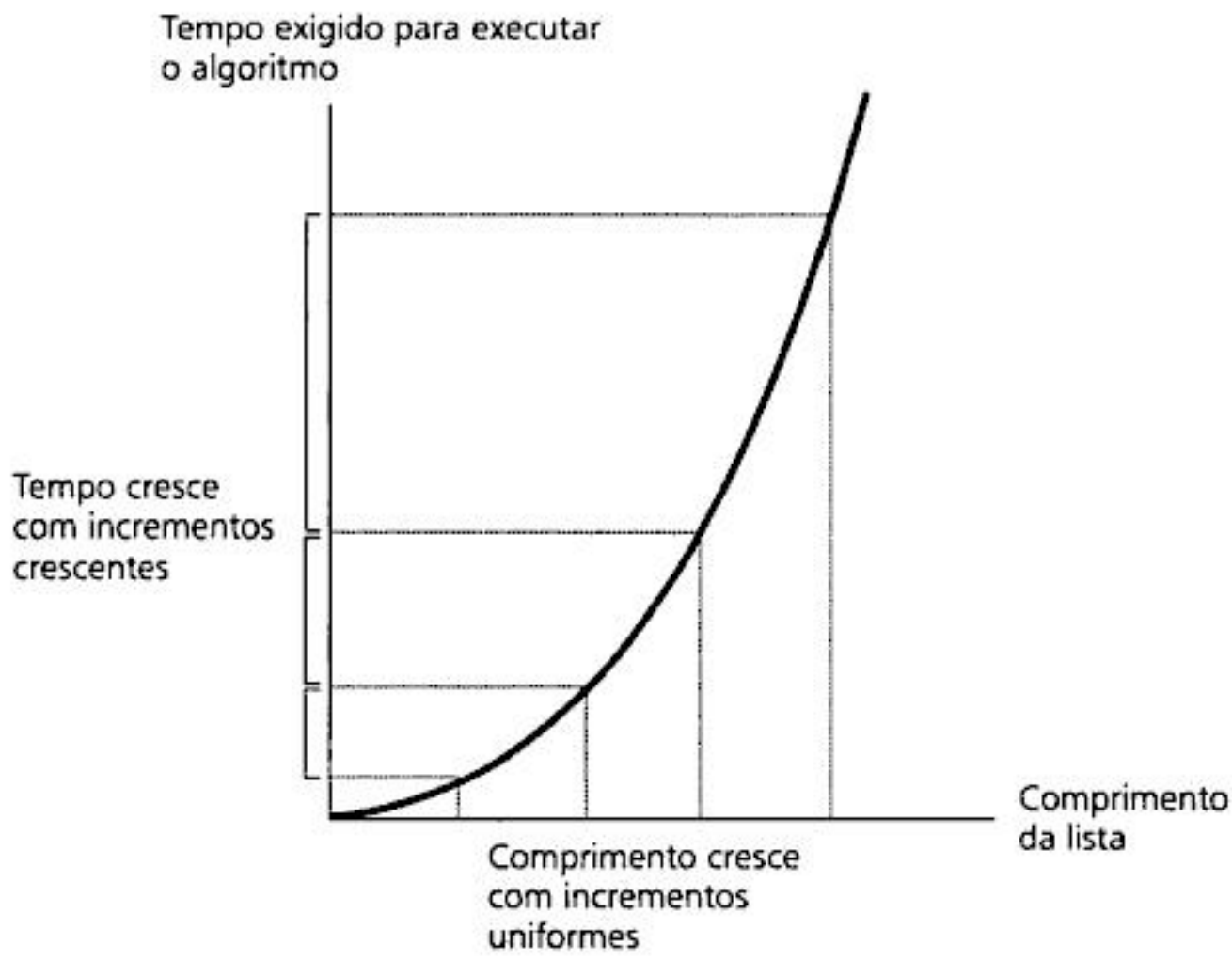



**Figura 4.19** Gráfico de análise do pior caso do algoritmo de ordenação por inserção.

A característica de distinção entre as Figuras 4.19 e 4.20 é a forma geral das curvas envolvidas. É a forma geral de um gráfico, em vez das específicas, que revela o desempenho de um algoritmo à medida que seus dados de entrada crescem. Observe que a forma geral de um gráfico é determinada pelo tipo de expressão que está sendo desenhada em vez da expressão específica — todas expressões lineares produzem uma linha reta; todas as quadráticas produzem uma parábola; todas as logarítmicas produzem a forma logarítmica mostrada na Figura 4.20. É costume identificar uma curva com a expressão mais simples que produz a sua forma. Em particular, identificamos a forma parabólica com a expressão $n^2$ e a logarítmica com a expressão lg $n$.

Vimos que a forma do gráfico obtido na comparação do tempo exigido por um algoritmo na realização de sua

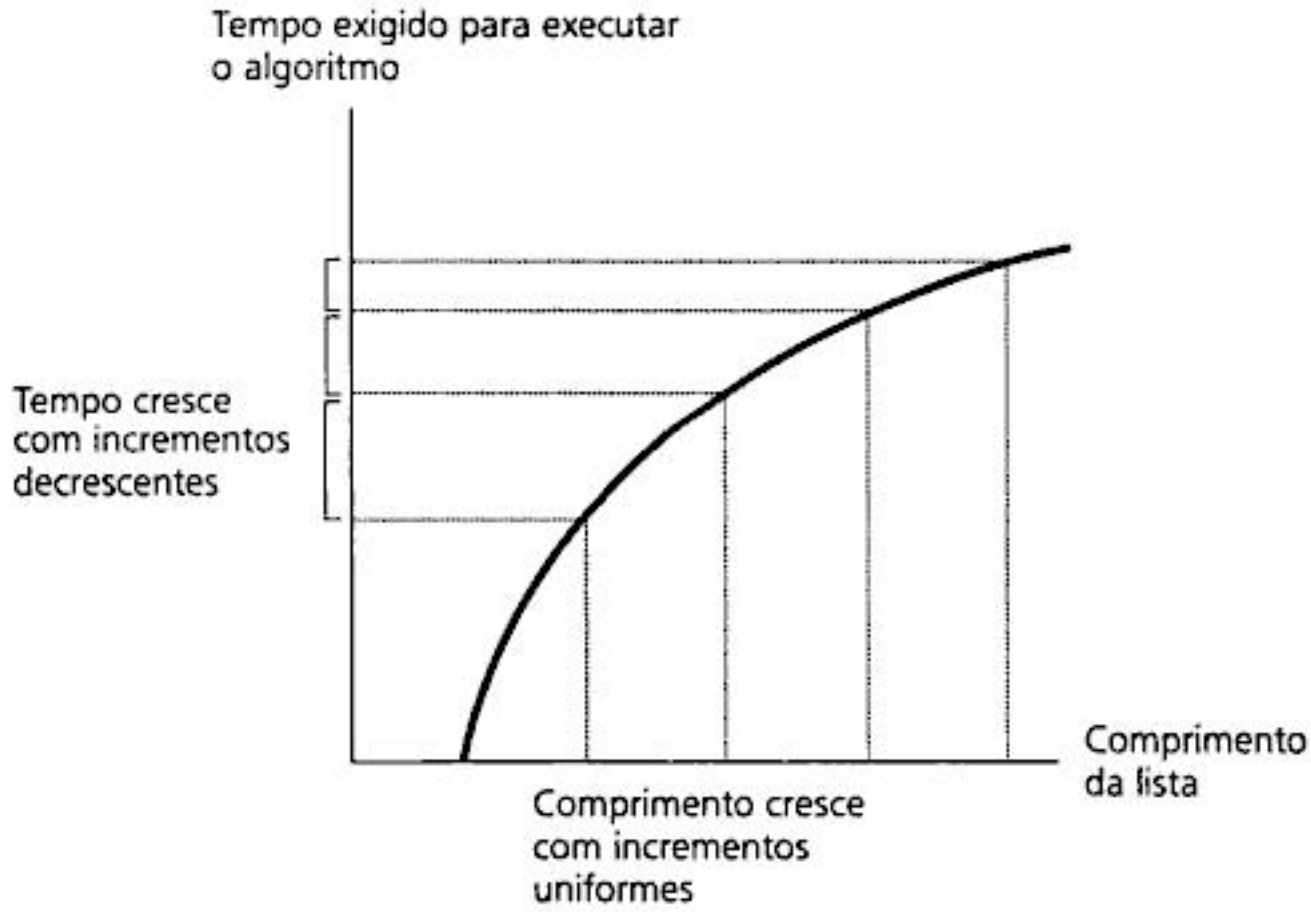



**Figura 4.20** Gráfico de análise do pior caso do algoritmo de busca binária.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

frágil. Afinal de contas, a verificação por testes nada prova, senão que o programa funciona corretamente para os dados de teste. Qualquer conclusão adicional é pura projeção. Os erros contidos em um programa muitas vezes são conseqüência de equívocos sutis facilmente escondidos também nos testes. Assim, erros em um programa, como o nosso no problema da corrente de ouro, podem ficar e freqüentemente ficam sem ser detectados, embora esforços significativos tenham sido feitos para evitá-los. Um exemplo drástico ocorreu na AT&T. Um erro no *software* que controlava 114 estações de chaveamento permaneceu sem ser detectado desde a instalação em dezembro de 1989 até 15 de janeiro de 1990, quando um conjunto único de circunstâncias causou o bloqueio desnecessário de aproximadamente cinco milhões de chamadas durante nove horas.

## QUESTÕES/EXERCÍCIOS

1. Suponha que uma máquina programada com o nosso algoritmo de ordenação por inserção necessite em média um segundo para ordenar uma lista de 100 nomes. Quanto tempo você estima que ela necessitará para ordenar uma lista de 1.000 nomes? E de 10.000?

2. Dê um exemplo de um algoritmo pertencente a cada uma das seguintes classes: $\Theta(\lg n)$, $\Theta(n)$ e $\Theta(n^2)$.

3. Liste as classes $\Theta(n^2)$, $\Theta(\lg n)$, $\Theta(n)$ e $\Theta(n^3)$ em ordem decrescente de eficiência.

4. Considere o seguinte problema e uma sugestão de resposta. A resposta sugerida está correta? Justifique.

   **Problema:** Suponha que uma caixa contenha três cartas. Uma delas é pintada de preto nas duas faces, outra de vermelho também nas duas faces, e a terceira carta é pintada de vermelho em uma das faces e de preto na outra. Uma delas é tirada da caixa e você é autorizado a ver uma de suas faces. Qual a probabilidade de a outra face da carta ser da mesma cor da face que você viu?

   **Resposta sugerida:** A probabilidade é 50%. Suponha que a face da carta que você viu seja vermelha. (Este argumento seria idêntico para a carta cuja face é preta.) Somente duas cartas entre as três possuem uma face vermelha. Assim, a carta que você viu deve ser uma delas. Uma dessas duas cartas é vermelha na outra face, enquanto a outra é preta. Assim, a carta que você viu tanto pode ser vermelha na outra face quanto preta.

5. O seguinte trecho de programa é uma tentativa de computar o quociente (despreze o resto da divisão) de dois inteiros positivos por meio da contagem do número de vezes que o divisor pode ser subtraído do dividendo antes que a diferença se torne menor que o divisor. Por exemplo, 7 dividido por 3 resulta em 2 porque 3 pode ser subtraído de 7 duas vezes. O programa abaixo está correto? Justifique sua resposta.

```
Contador ← 0;
Resto ← Dividendo:
repete (Resto ← Resto − Divisor;
        Contador ← Contador + 1)
até (Resto < Divisor)
Quociente ← Contador.
```

6. O seguinte trecho de programa foi projetado para calcular o produto de dois inteiros não-negativos X e Y, acumulando a soma de X cópias de Y — por exemplo, 3 multiplicado por 4 é calculado acumulando a soma de três quatros. O programa está correto? Justifique sua resposta.

```
Produto ← Y;
Contador ← 1;
enquanto (Contador < X) faça
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

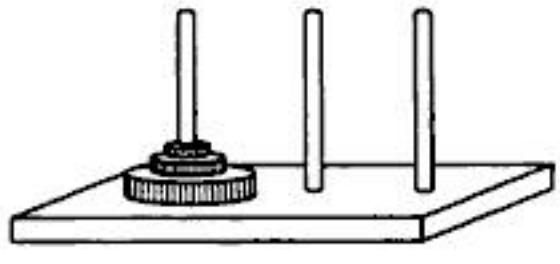
próximo inteiro menor que este e não-negativo. Usamos a notação $n!$ para expressar o fatorial do inteiro $n$. Assim, o fatorial de 3 (escrito 3!) pode ser calculado da seguinte maneira: $3 \times (2!) = 3 \times (2 \times (1!)) = 3 \times (2 \times (1 \times (0!))) = 3 \times (2 \times (1 \times (1))) = 6$. Escreva um algoritmo recursivo que calcule o fatorial de um valor fornecido.

**34.** a. Suponha que você deva ordenar uma lista com cinco nomes e que tenha escrito um algoritmo que ordena listas com quatro nomes. Escreva um algoritmo para ordenar a lista de cinco nomes e que utilize o algoritmo disponível.

b. Escreva um algoritmo recursivo para ordenar listas arbitrárias de nomes, baseado na técnica aplicada na resolução do item (a).

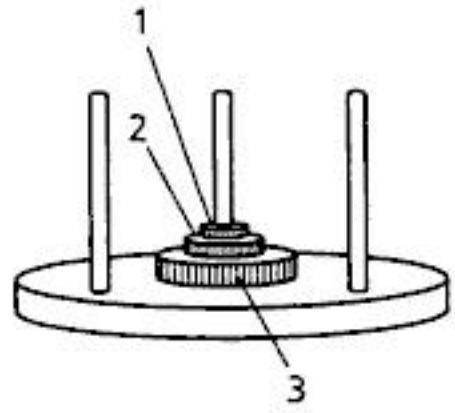
**35.** O quebra-cabeças chamado Torres de Hanoi consiste em três pinos, e cada um contém vários anéis, empilhados em ordem descendente de diâmetro, de baixo para cima. O problema consiste em mover a pilha de anéis para algum outro pino. É permitido mover somente um anel de cada vez, e em nenhum momento um anel pode ser colocado sobre um outro que lhe seja menor. Observe que, se o quebra-cabeças tivesse somente um anel, a solução seria trivial. Quando se trata de mover vários anéis, se pudermos mover todos, exceto o maior, para algum outro pino, o maior poderá ser colocado no terceiro pino e os restantes ser posteriormente transferidos para cima deste. Usando esta observação, desenvolva um algoritmo recursivo que resolva o quebra-cabeças das Torres de Hanoi para um número arbitrário de anéis.

**36.** Uma alternativa de solução para o quebra-cabeças das Torres de Hanoi (Problema 35) consiste em imaginar os pinos dispostos circularmente e localizados conforme as posições 4h, 8h e 12 horas de um relógio. Os anéis, que inicialmente estão todos em um dos pinos, são numerados como 1, 2, 3 e assim por diante,



atribuindo-se o valor 1 ao menor anel. Os anéis ímpares, quando estão no topo de uma pilha, podem mover-se para o próximo pino em sentido horário; de modo similar, os anéis pares podem mover-se em sentido anti-horário (contanto que estes movimentos não coloquem um anel sobre um outro menor). Sob estas condições, mover sempre o anel de maior valor e que puder ser deslocado. Com base nesta observação, desenvolva um algoritmo não-recursivo para resolver o problema das Torres de Hanoi.

**37.** Desenvolva dois algoritmos, um baseado em estruturas iterativas e o outro, em estruturas recursivas, para imprimir o salário diário de um trabalhador que, a cada dia, recebe o dobro do salário do dia anterior (iniciando com um centavo no primeiro dia de trabalho), durante 30 dias. Quais problemas, relativos ao armazenamento de dados,




você provavelmente encontrará se suas soluções forem implementadas em um computador real?

**38.** Escreva um algoritmo para calcular a raiz quadrada de um número positivo. Inicie com o próprio número como primeira tentativa. Produza repetidamente uma nova tentativa a partir da primeira, calculando a média entre a tentativa anterior e o resultado da divisão do número original por este valor. Analise o controle deste processo iterativo. Qual condição específica deve ser usada para terminar a iteração?

**39.** Desenvolva um algoritmo que liste todos os possíveis arranjos dos símbolos em uma cadeia de cinco caracteres distintos.

**40.** Desenvolva um algoritmo que, dada uma lista de nomes, encontre o nome mais longo da lista. Determine a ação que seu algoritmo executará se houver vários nomes com este mesmo comprimento. O que seu algoritmo fará se todos

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

CAPÍTULO 5

# *Linguagens de programação*

Seria quase impossível o desenvolvimento de sistemas complexos de *software*, como os sistemas operacionais, os *softwares* de redes e a grande quantidade de aplicações disponíveis atualmente, se os seres humanos fossem obrigados a expressar os seus algoritmos diretamente em linguagem de máquina. Trabalhar com um volume significativo de detalhes da linguagem de máquina e, ao mesmo tempo, tentar organizar sistemas complexos é, no mínimo, uma experiência desgastante. Como conseqüência, linguagens de programação semelhantes ao nosso pseudocódigo foram desenvolvidas, permitindo expressar algoritmos em um formato que fosse tanto agradável como fácil de traduzir para instruções em linguagem de máquina. Tais linguagens evitam a complexidade dos registradores, endereços de memória e ciclos de máquina durante o processo de desenvolvimento de programas, concentrando-se, em lugar disso, nas características do problema a ser solucionado. Neste capítulo, estudaremos o tópico de linguagens de programação.



**Os asteriscos indicam sugestões de seções consideradas opcionais.*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

rações de E/S têm causado o aparecimento na "mesma" linguagem de diferentes características, ou dialetos. Conseqüentemente, em geral é necessário fazer ao menos pequenas modificações no programa antes de movê-lo de uma máquina para outra.

A causa deste problema de portabilidade é, em alguns casos, a falta de concordância em relação à composição correta da definição de uma linguagem em particular. Para auxiliar nesta questão, o American National Standards Institute (ANSI) e a International Organization for Standardization (ISO) adotaram e publicaram padrões para muitas das linguagens mais populares. Em outros casos, surgiram padrões informais, devido à popularidade de um dado dialeto de uma linguagem e ao desejo, por parte de alguns autores de compiladores, de oferecerem produtos compatíveis. Contudo, mesmo no caso de linguagens padronizadas, os projetistas de compiladores geralmente oferecem características, às vezes chamadas *extensões da linguagem*, que não são parte da linguagem padronizada. Se um programador tirar vantagem dessas características, o programa produzido não será compatível com ambientes que usam compiladores de outros vendedores.

Na história global das linguagens de programação, é realmente pouco significativo o fato de as linguagens de terceira geração não terem conseguido atingir uma verdadeira independência de máquina. Primeiramente, elas se mostraram tão próximas da independência de máquina que os programas nelas desenvolvidos podiam ser transportados de uma máquina para outra com relativa facilidade. Em segundo lugar, o objetivo de independência de máquina acabou convertendo-se apenas em um embrião para metas muito mais exigentes. Na época em que a independência de máquina finalmente se tornou acessível, a sua importância se diluiu diante das exigências da época, então muito mais fortes. De fato, a constatação de que máquinas podiam executar instruções de alto nível como

```
atribua a Total o valor (Preço + Imposto)
```

conduziu os pesquisadores a sonhar com ambientes de programação que permitissem aos seres humanos comunicar-se com as máquinas usando diretamente os conceitos abstratos com os quais costumam analisar os problemas, evitando assim a tradução dos conceitos para um formato compatível com a máquina. Além disso, os pesquisadores buscam máquinas voltadas mais ao desenvolvimento de algoritmos do que à sua execução. O resultado tem sido uma evolução constante nas linguagens de programação que impede uma classificação clara em termos de gerações.

## Paradigmas de programação

A abordagem de gerações para classificar as linguagens de programação é baseada em uma escalar linear (Figura 5.1), de acordo com o grau em que o usuário é libertado do mundo da ininteligibilidade computacional e autorizado a analisar elementos associados ao seu próprio problema. Na realidade, o desenvolvimento das linguagens de programação não evoluiu exatamente desta forma, mas se ramificou conforme foram aparecendo diferentes formas de processos de programação (diferentes paradigmas). Por conseguinte, o desenvolvimento histórico das linguagens de programação fica mais bem representado por um diagrama com múltiplas trajetórias, conforme mostra a Figura 5.2, na qual estão apresentados os diversos caminhos, resultantes de diferentes paradigmas, que surgiram e progrediram de forma independente. A Figura 5.2 apresenta quatro caminhos que representam os paradigmas funcional, orientado a objeto, imperativo e

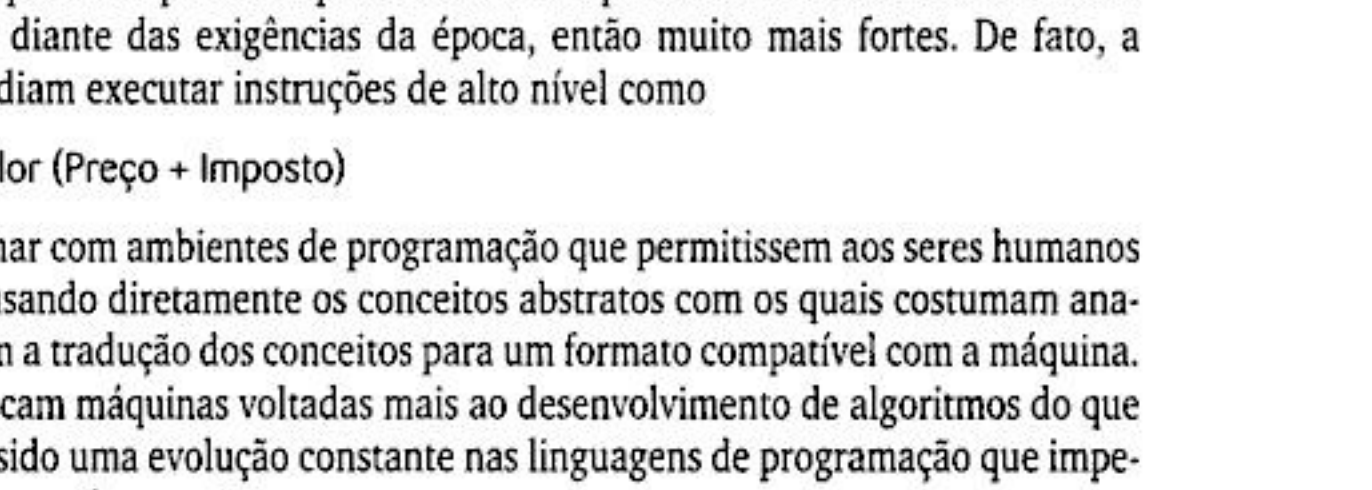



**Figura 5.1** Gerações de linguagens de programação.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

responder à ocorrência de vários eventos, tais como a de ser selecionado por meio de um aperto do botão do *mouse* ou a de ser deslocado pela tela afora, em resposta aos movimentos do *mouse*. Assim, o sistema inteiro tem a forma de uma coleção de objetos, e cada um sabe como responder a certos eventos.

Muitas das vantagens de um projeto orientado a objeto são conseqüências da estrutura modular decorrente da filosofia de orientação a objeto. Cada objeto é implementado como uma unidade independente e bem definida. Uma vez determinadas dessa forma as propriedades de uma entidade, a definição assim obtida pode ser reutilizada todas as vezes que esta entidade se fizer necessária. Assim, os adeptos da programação orientada a objeto argumentam que tal paradigma fornece um ambiente natural para o uso de "blocos construtivos" no desenvolvimento de *software*. Eles preconizam bibliotecas de *software* baseado em objetos, com o qual novos sistemas de *software* podem ser elaborados, de forma similar ao que ocorre com muitos produtos tradicionais, construídos a partir de componentes pré-fabricados.

O paradigma da orientação a objeto está se tornando muito popular, e muitos acreditam que exercerá uma influência dominante no futuro. De fato, nos capítulos seguintes, mencionaremos várias vezes o papel desempenhado pelos métodos orientados a objeto em toda a Ciência da Computação. Outra vantagem da estrutura modular obtida a partir do paradigma orientado a objeto é que toda a comunicação entre os módulos é feita de uma maneira uniforme e bem definida (trocando mensagens entre os objetos) — uma técnica que é prontamente adaptável à comunicação entre computadores ligados em rede. Sem dúvida, o conceito de objetos que trocam mensagens entre si é uma maneira natural de abordar a tarefa de desenvolvimento de sistemas de *software* que serão distribuídos em uma rede. Assim, você não deve se surpreender ao constatar que o *software* baseado no modelo cliente-servidor (Seção 3.3) freqüentemente é desenvolvido no contexto orientado a objeto — o servidor é um objeto que responde às mensagens de outros objetos, que fazem o papel de clientes.

O paradigma orientado a objeto está tendo forte influência no campo da computação, portanto, será discutido em maior detalhe na Seção 5.5. Além disso, veremos repetidamente as implicações desse paradigma nos capítulos futuros. Em particular, testemunharemos a influência do paradigma orientado a objeto na área de Engenharia de *Software* (Capítulo 6), no projeto de bancos de dados (Capítulo 9) e, no Capítulo 7, veremos como a abordagem orientada a objeto para a construção de *software* evoluiu como extensão natural do estudo das estruturas de dados.

Finalmente, devemos notar que as rotinas internas aos objetos que descrevem como eles devem responder às várias mensagens são essencialmente pequenas unidades de programas imperativos. Assim, a maioria das linguagens orientadas a objeto contém muitas das características encontradas nas linguagens imperativas. De fato, a popular linguagem orientada a objeto C++ foi desenvolvida com a adição de recursos orientados a objeto à linguagem imperativa conhecida como C. Nas seções 5.2 e 5.3, exploraremos características comuns tanto às linguagens imperativas quanto às orientadas a objeto. Assim fazendo, estaremos discutindo conceitos que permeiam a maioria dos *softwares* atuais.

## *QUESTÕES/EXERCÍCIOS*

1. Em que sentido um programa em linguagem de terceira geração é independente de máquina? Em que sentido ele ainda depende da máquina?
2. Qual a diferença entre um montador e um compilador?
3. Podemos resumir o paradigma de programação imperativo dizendo que ele coloca ênfase em descrever um processo que conduz à solução de um dado problema. Resuma, de forma semelhante, os paradigmas declarativo, funcional e orientado a objeto.
4. Em que sentido as linguagens de programação de terceira geração são de nível mais alto que as de gerações anteriores?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

to da segunda linha e quarta coluna seria referenciado por `Escores (2, 4)`. (Veja novamente a Figura 5.6).

Algumas linguagens fornecem ao programador uma flexibilidade significativa no estabelecimento das faixas de índices usadas para referenciar matrizes. Por exemplo, a instrução

```
Escores: array [3..4,
12..20] of integer;
```

em Pascal declara a variável `Escores` como uma matriz bidimensional de inteiros como antes, exceto em que as linhas são identificadas pelos valores 3 e 4 e as colunas, numeradas de 12 a 20. Assim, o elemento da segunda linha e quarta coluna seria referenciado por `Escores [4, 15]`.

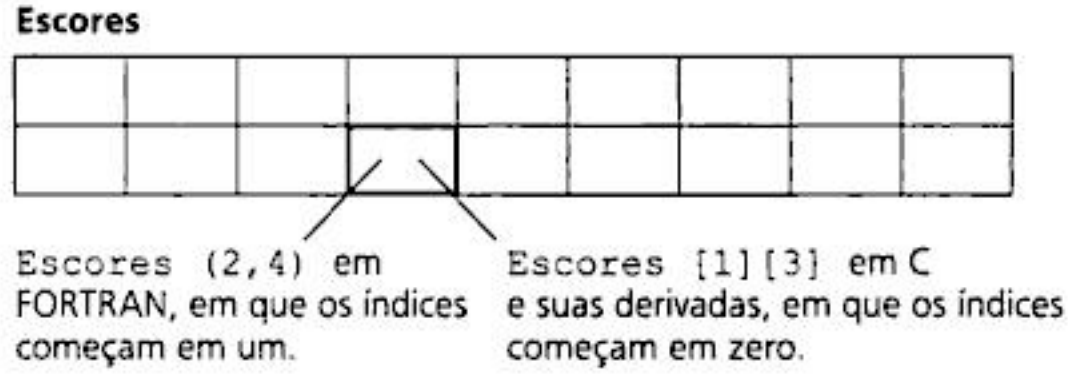



**Figura 5.6** Uma matriz bidimensional com duas linhas e nove colunas.

Em contraste com o arranjo homogêneo, no qual todos os itens de dados são do mesmo tipo, um **arranjo heterogêneo** é um bloco de dados no qual diferentes elementos podem ter diferentes tipos. Por exemplo, um bloco de dados que se refere a um funcionário pode consistir em um elemento chamado `Nome`, do tipo caractere, um elemento chamado `Idade` do tipo inteiro, e um elemento chamado `NiveldeQualificacao` do tipo real. A Figura 5.7 mostra como tal arranjo pode ser declarado em C e em Pascal.

Um componente de um arranjo heterogêneo normalmente é referenciado pelo nome do arranjo seguido por um ponto e o nome do componente. Por exemplo, o componente `Idade` no arranjo `Funcionario` da Figura 5.7 seria referenciado por `Funcionario.Idade`.

No Capítulo 7, veremos como as estruturas conceituais como os arranjos são de fato implementadas em um computador. Mais especificamente, aprenderemos que os dados contidos em um arranjo podem estar espalhados em uma larga área de memória principal ou do armazenamento em massa. Isso explica por que nos referimos às estruturas de dados como formas *conceituais* de dados. De fato, a disposição real dentro do sistema de armazenamento da máquina pode ser bem diferente da disposição conceitual.

## Constantes e literais

Em alguns casos, um valor fixo, predeterminado, é usado em um programa. Por exemplo, um programa para o controle do tráfego aéreo nas proximidades de um aeroporto pode conter numerosas referências à sua altitude acima do nível do mar. Ao escrever tal programa, pode-se incluir explicitamente este valor, digamos 645 pés

**a.** Declarações de arranjos em linguagem Pascal

```
var
  Funcionario: record;
                 Nome: packed array[1. .8] of char;
                 Idade: integer;
                 NiveldeQualificacao: real
               end
```

**b.** Declaração de arranjos em linguagem C

```
struct
{char Nome [8];
  int Idade;
  float NiveldeQualificacao;
} Funcionarios;
```

**c.** Organização conceitual do arranjo

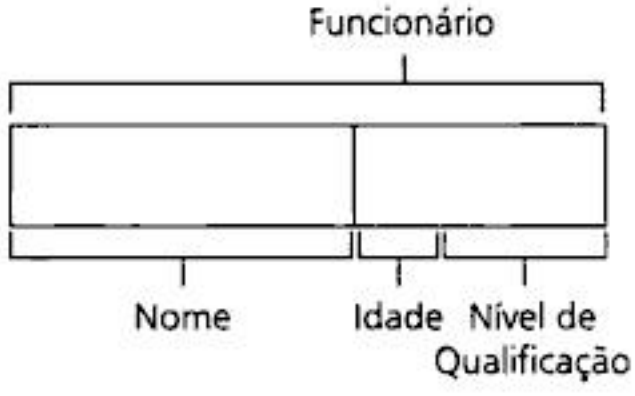
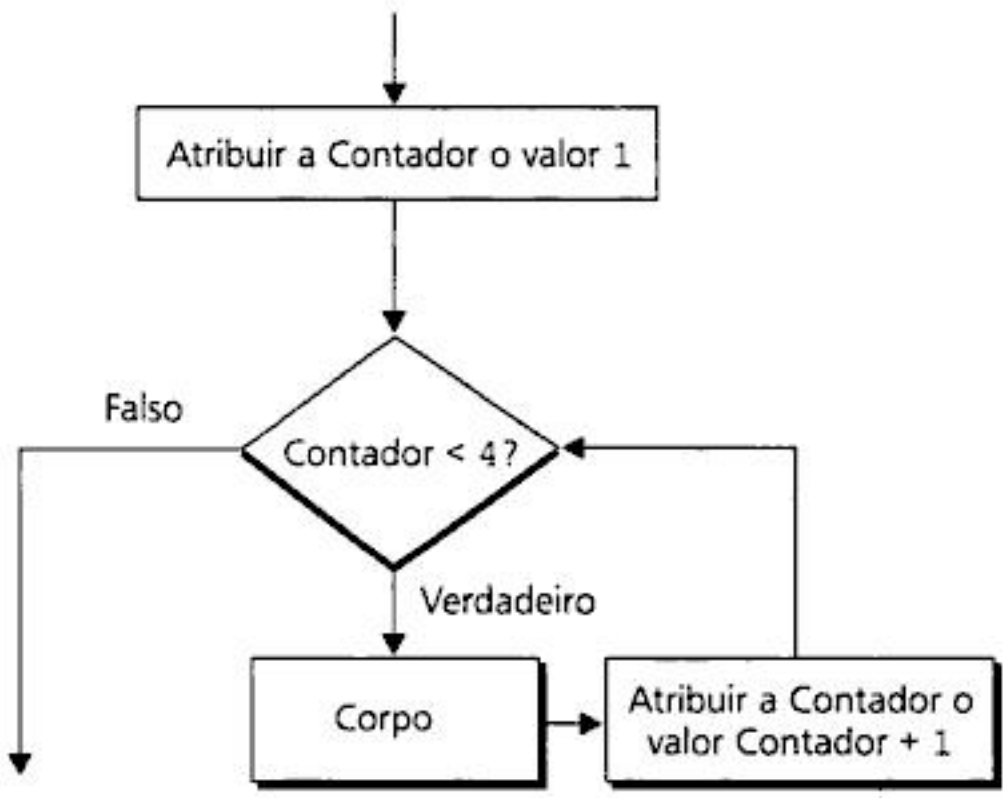



**Figura 5.7** Declaração de arranjos heterogêneos, em Pascal e C.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.




**Pascal**

```
for Contador := 1 to 3 do
begin
     corpo;
end;
```

**C++, C# e Java**

```
for (int Contador = 1; Contador<4; Contador++)
   corpo;
```

**Figura 5.9** A estrutura para e sua representação nas linguagens Pascal, C++, C# e Java.

corpo da iteração seja executado de modo que a variável `Contador` inicialmente tenha o valor 1, em seguida o valor 2, e finalmente, o valor 3.

O objetivo desses exemplos é demonstrar que estruturas ramificadas genéricas aparecem, com pequenas variações, nas linguagens de programação imperativa e orientada a objeto. Um resultado teórico surpreendente da Ciência da Computação é que a capacidade de expressar somente algumas destas estruturas assegura que uma linguagem de programação tenha meios para expressar uma solução para qualquer problema que tenha solução algorítmica. Analisamos este fato no Capítulo 11. Por ora, mencionamos apenas que aprender uma linguagem de programação não é uma tarefa interminável de aprendizado de diferentes instruções de controle. Na verdade, as principais instruções de controle disponíveis nas linguagens de programação atuais são essencialmente variações das instruções aqui identificadas.

A escolha das estruturas de controle a serem incorporadas a uma linguagem é uma importante decisão de projeto. O objetivo é criar uma linguagem que expresse algoritmos de forma legível e que auxilie o programador nesta tarefa. Isto é possível restringindo o uso desses recursos — os quais, historicamente falando, têm conduzido a programações de baixa qualidade — e encorajando o uso de recursos mais bem projetados. O resultado é a **programação estruturada**, uma prática pouco compreendida que abrange uma metodologia organizada de projeto, combinada com o uso disciplinado das instruções de controle da linguagem. A idéia é produzir um programa que possa ser prontamente compreendido e demonstrar que ele atende às suas especificações.

## Comentários

A experiência mostra que, a despeito do quanto uma linguagem de programação seja bem projetada, ou do quanto os seus recursos sejam bem utilizados, informações adicionais são necessárias, ou no mínimo úteis, para a compreensão humana de qualquer programa de porte razoável. Por isso, as linguagens de

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

casamento, precisamos meramente seguir as diretrizes do procedimento, assumindo que o termo genérico `Lista` se refere à lista de convidados. Se, porém, desejarmos ordenar uma lista de membros de uma corporação, deveremos interpretar o termo genérico Lista como uma referência à lista de membros.

Esses termos genéricos nos procedimentos são chamados **parâmetros**. Mais precisamente, os termos usados ao escrever o procedimento são chamados **parâmetros formais**, cujo significado preciso quando o procedimento é aplicado são chamados **parâmetros reais** ou **argumentos**. Em um certo sentido, os parâmetros formais representam encaixes do procedimento, nos quais os parâmetros reais são inseridos quando o procedimento é chamado. Na realidade, os parâmetros formais são variáveis do procedimento às quais são atribuídos valores (os parâmetros reais) quando o procedimento é executado.

Em geral, as linguagens de programação seguem o formato de nosso pseudocódigo para identificar os parâmetros formais em um procedimento, isto é, a maioria das linguagens de programação exige que, quando se define um procedimento, os parâmetros formais sejam listados entre parênteses no cabeçalho deste. Como exemplo, a Figura 5.11 apresenta a definição de um parâmetro chamado `Populacao-Projetada` como seria escrito na linguagem de programação C. O procedimento espera receber uma taxa de crescimento anual específica, quando é chamado. Baseado nessa taxa, o procedimento calcula a população projetada de uma espécie, pressupondo uma população inicial de 100, para os próximos 10 anos, e guarda esses valores em um vetor (matriz unidimensional) chamado `Populacao`.

A maioria das linguagens de programação também usa a notação entre parênteses para identificar os parâmetros reais quando um procedimento é chamado, isto é, a instrução que solicita a execução de um procedimento consiste no nome do procedimento seguido pela lista de parâmetros reais entre parênteses. Assim, uma instrução como

```
PopulacaoProjetada (0.03);
```

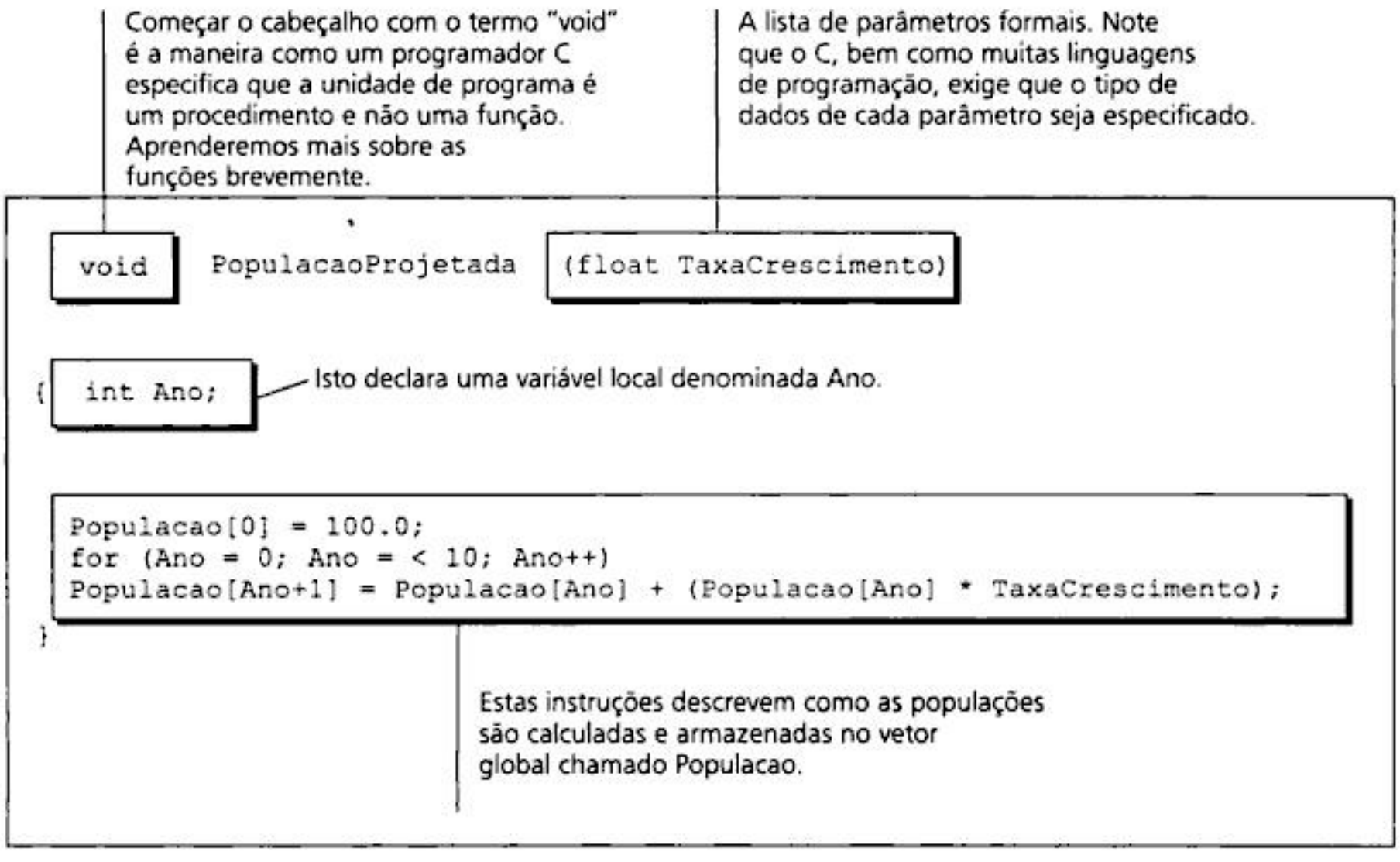



**Figura 5.11** O procedimento PopulacaoProjetada escrito na linguagem de programação C.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

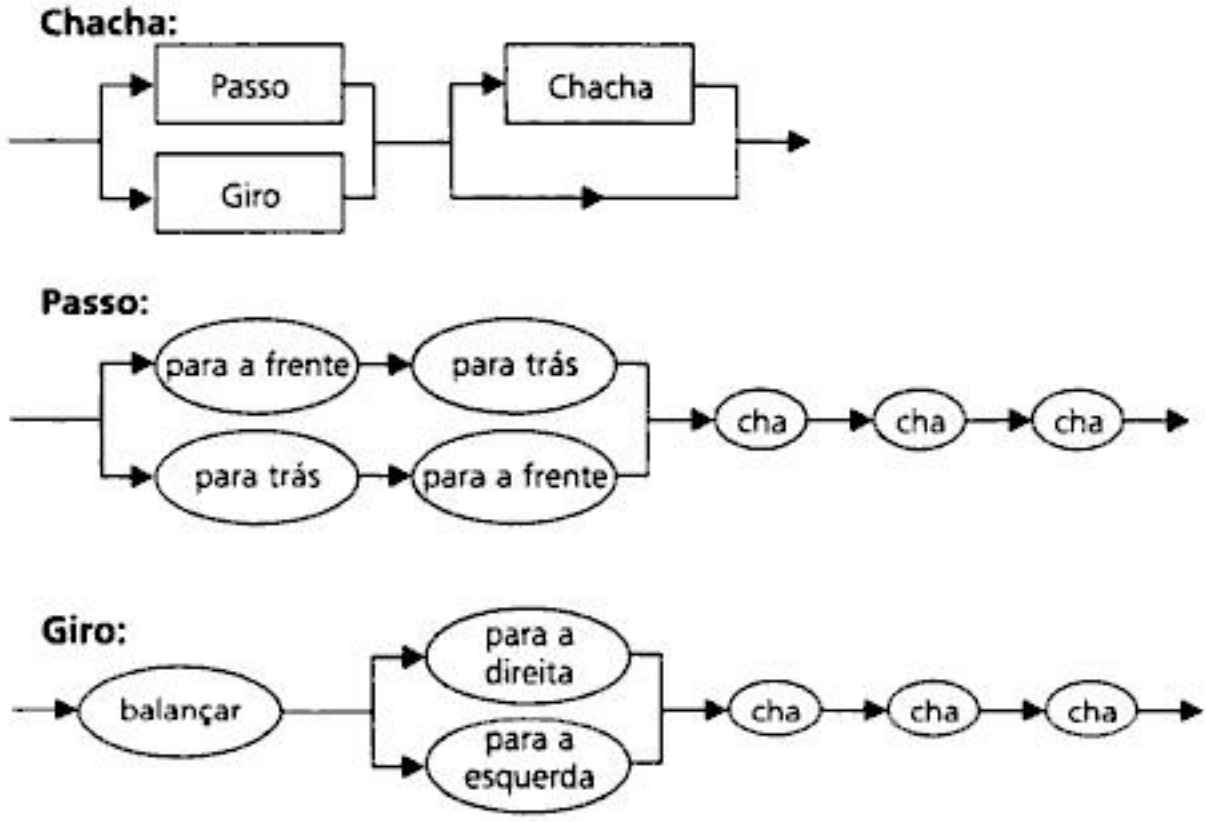
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

## QUESTÕES/EXERCÍCIOS

1. Descreva os três principais passos do processo de tradução.
2. O que é uma tabela de símbolos?
3. Construa uma árvore de sintaxe para a expressão

   ```
   x * y + x + z
   ```

   com base nos diagramas de sintaxe da Figura 5.18.
4. Descreva as cadeias que obedecem à estrutura Chacha, definida de acordo com os seguintes diagramas de sintaxe.




# 5.5 Programação orientada a objeto

Vimos na Seção 5.1 que o paradigma da orientação a objeto leva ao desenvolvimento de unidades de programa ativas, chamadas objetos, em que cada uma contém procedimentos que descrevem como esses objetos devem responder a vários estímulos. A abordagem orientada a objeto a um problema é identificar os objetos envolvidos e descrevê-los como unidades autônomas. Por sua vez, as linguagens de programação orientadas a objeto fornecem instruções para expressar essas idéias. Nesta seção, introduziremos algumas dessas instruções como aparecem nas linguagens C++, Java e C#, que atualmente são as três linguagens orientadas a objeto mais proeminentes.

## Classes e objetos

Considere a tarefa de desenvolver um jogo de computador no qual o jogador deve proteger a Terra de meteoros, atirando com raios *laser* de alta potência. Cada *laser* contém uma fonte de energia interna finita que é parcialmente consumida a cada disparo. Uma vez gasto, o *laser* se torna inútil. Cada *laser* deve responder aos comandos de apontar para a direita, apontar para a esquerda e de disparar o facho.

No paradigma orientado a objeto, cada *laser* no jogo de computador seria implementado como um objeto que contém um registro de sua energia disponível, bem como procedimentos para modificar a sua pontaria e disparar o facho de *laser*. Uma vez que todos os objetos *laser* possuem as mesmas propriedades, eles podem ser construídos a partir do mesmo modelo. No paradigma orientado a objeto, esse modelo é chamado **classe**. Nas linguagens C++, Java e C#, uma classe é descrita por uma declaração da forma

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

## *C# e .NET* framework

Para simplificar o processo de desenvolvimento de *software* que atenda às necessidades atuais (como o desenvolvimento de sítios Web que se comunicam entre si para servir a seus clientes), a Microsoft desenvolveu um ambiente de *software* chamado .NET *framework*. Dentre outras coisas, esse *framework* contém uma coleção (chamada Biblioteca de Classes) de componentes que podem ser usados como ferramentas abstratas quando se desenvolve *software* de aplicação. Embora grande parte dessa biblioteca possa ser acessada por sistemas de desenvolvimento de *software* já fornecidos pela Microsoft (como o Visual Basic), a Microsoft projetou uma nova linguagem de programação, chamada C#, que foi produzida visando .NET *framework*. Enquanto linguagem, o C# é um primo próximo do C++. A novidade do C# é que ele promete ser a principal linguagem para o desenvolvimento de *software* que utilize a .NET *framework*. Embora o C# e a .NET *framework* sejam propostos para ser usados apenas nas máquinas com sistemas operacionais da Microsoft, a popularidade desses sistemas implica que o C# será uma linguagem de programação popular em um futuro próximo.

gráfico orientado a objeto consista em diversos objetos, e cada um represente uma forma (circunferência, retângulo, triângulo etc.) Uma específica imagem pode consistir em uma coleção desses objetos. Cada objeto sabe o seu tamanho, localização e cor, bem como as respostas à mensagens que solicitam, por exemplo, que ele se mova para algum lugar, ou que seja exibido na tela do monitor. Para desenhar uma imagem, simplesmente enviamos uma mensagem "autodesenhe" a cada objeto da mesma. Contudo, a rotina usada para desenhar um objeto varia de acordo com a sua forma — desenhar um quadrado não é o mesmo processo que desenhar uma circunferência. Essa interpretação personalizada de uma mensagem é conhecida como **polimorfismo**; diz-se que a mensagem é polimórfica.

Outra característica associada com a programação orientada a objeto é o **encapsulamento**, que se refere ao acesso restrito às propriedades internas de um objeto. Dizer que certos recursos de um objeto são *encapsulados* significa que apenas o objeto pode ter acesso a eles. Diz-se que os recursos encapsulados são privados e os acessíveis de fora do objeto, públicos.

Como exemplo, retornemos à nossa `ClaseLaser` originalmente esboçada na Figura 5.23. Lembre-se de que ela descreve uma variável de instância `EnergiaRestante` e três métodos: `virarDireita`, `virarEsquerda` e `disparar`. Esses métodos devem ser chamados por

Os componentes da classe são designados públicos ou privados, conforme forem acessíveis a outras unidades de programa ou não.

```
classe ClasseLaser
{private int EnergiaRestante;

 public ClasseLaser (EnergiaInicial)
 {EnergiaRestante = EnergiaInicial;
 }

 public void virarDireita ( )
 { ... }

 public void virarEsquerda ( )
 { ... }

 public void disparar ( )
 { ... }
}
```

**Figura 5.25** Nossa definição de ClasseLaser utiliza o encapsulamento como se fosse aparecer em um programa Java ou C#.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Neste caso, as duas proposições originais formam uma terceira, que chamamos de **resolvente**. É importante observar que o resolvente é uma conseqüência lógica das afirmações originais. Isto é, se as proposições originais forem verdadeiras, o resolvente também deverá ser verdadeiro. (Se *Q* for verdadeiro, então *R* deverá ser, mas se *Q* for falso, então *P* deverá ser verdadeiro. Assim, independentemente da veracidade de *Q*, ou *P* ou *R* deverá ser verdadeiro.)

Representamos graficamente a resolução de duas proposições na Figura 5.26, na qual descrevemos as proposições originais a partir das quais são traçadas, para baixo, linhas em direção a seus resolventes. Note-se que as resoluções só podem ser aplicadas a pares de proposições que aparecem na **forma de cláusulas** — isto é, proposições cujos elementos básicos são conectados pela operação booleana OR. Assim,

$$P \text{ OR } Q$$

está em forma de cláusula, mas

$$P \rightarrow Q$$

não. O fato de este problema potencial não causar qualquer preocupação séria é uma conseqüência do teorema da lógica matemática, resultante do fato de que qualquer proposição, expressa em lógica de predicados de primeira ordem (um sistema extensivamente expressivo, usado para a representação de proposições), pode ser reescrita na forma de cláusulas. Não analisaremos este teorema importante, mas para futuras referências observemos que a proposição

$$P \rightarrow Q$$

é equivalente à cláusula

$$Q \text{ OR } \neg P$$

Um conjunto de proposições será considerado **inconsistente** se for impossível que todas sejam verdadeiras ao mesmo tempo. Em outras palavras, um conjunto inconsistente de proposições é um conjunto de proposições contraditórias. Um exemplo simples seria uma proposição da forma *P* combinada com outra, da forma $\neg P$. Os lógicos demonstraram que uma resolução repetida constitui um método sistemático para confirmar a inconsistência de um conjunto de cláusulas contraditórias. A regra é que se a aplicação de resolução repetida produz a cláusula vazia (resultado obtido ao solucionar uma cláusula da forma *P* com uma da forma $\neg P$), então o conjunto original de proposições deve ser inconsistente. Como ilustração, a Figura 5.27 mostra que o conjunto de proposições

$P \text{ OR } Q$
$R \text{ OR } \neg Q$
$\neg R$
$\neg P$

é inconsistente.

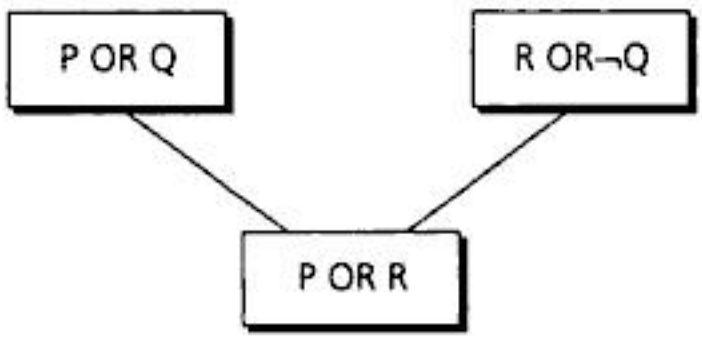



**Figura 5.26** Resolução das proposições (P OR Q) e (R OR ¬ Q) que resulta em (P OR R).

Suponhamos agora que queiramos confirmar que um conjunto de proposições implica uma outra proposição, a qual denotaremos por *P*. Implicar a proposição *P* é o mesmo que contradizer a proposição $\neg P$. Assim, para demonstrar que o conjunto original de proposições implica *P*, tudo o que precisamos fazer é expressar as proposições originais e a proposição $\neg P$ no formato de cláusulas, e em seguida aplicar a resolução até o aparecimento de uma cláusula vazia. Isto feito, poderemos concluir que a proposição $\neg P$ é inconsistente com as proposições originais, e assim estas devem implicar *P*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
femea(sue).
macho(bill).
macho(john).
genitor(john, carol).
genitor(sue, carol).
mae(X,Y):-
pai(X,Y):-
```

## *Problemas de revisão de capítulo*

(Os problemas marcados com asterisco se referem às seções opcionais.)

**1.** O que significa dizer que uma linguagem de programação é independente de máquina?

**2.** Traduza o seguinte programa em pseudocódigo para a linguagem de máquina descrita no Apêndice C.

```
x ← 0;
enquanto (x < 3) faça
(x ← x + 1)
```

**3.** Traduza a instrução

Meiocaminho ← Comprimento + Largura

para a linguagem de máquina do Apêndice C, pressupondo que `Comprimento`, `Largura` e `Meiocaminho` sejam representados em notação de vírgula flutuante.

**4.** Traduza a instrução de alto nível

```
if (X = 0)
  then Z ← Y + W
  else Z ← Y + X
```

para a linguagem de máquina do Apêndice C, pressupondo que `W`, `X`, `Y` e `Z` são representados por números em notação de complemento de dois que ocupa um *byte* de memória.

**5.** Por que era necessário identificar o tipo de dados associado às variáveis do Problema 4 para traduzir as instruções? Por que muitas linguagens de alto nível exigem que o programador identifique o tipo de dado de cada variável no início de um programa?

**6.** Cite e descreva quatro paradigmas diferentes de programação.

**7.** Suponha que a função f receba dois valores numéricos como parâmetros, determine o menor deles e retorne o resultado como valor de saída. Se $w$, $x$, $y$ e $z$ representam valores numéricos, qual será o resultado obtido ao calcular f(f($w$,$x$), f($y$,$z$))?

**8.** Seja f uma função que forneça o resultado da reversão de uma cadeia de símbolos e g uma função que retorne a concatenação de duas cadeias dadas como entrada. Se x for a cadeia *abcd*, qual será o resultado de g(f(x),x)?

**9.** Suponha que sua conta-corrente seja representada como um objeto, em um programa orientado a objeto projetado para manter o seu histórico financeiro. Que dados são armazenados neste objeto? Quais mensagens este objeto pode receber? Como são as respostas do objeto a tais mensagens? Que outros objetos poderiam ser usados no programa?

**10.** Resuma a diferença entre linguagem de máquina e linguagem de montagem.

**11.** Projete uma linguagem de montagem para a máquina descrita no Apêndice C.

**12.** O programador John argumenta que o recurso de declarar constantes dentro de um programa não é necessário, pois poderiam ser utilizadas variáveis em seu lugar. Por exemplo, o problema da altitude do aeroporto `AltitudedoAeroporto` da Seção 5.2 pode ser reescrito declarando `AltitudedoAeroporto` como variável e então atribuindo a ela o valor adequado no início do programa. Por que este método não é tão bom quanto o uso de uma constante?

**13.** Resuma a diferença entre as instruções declarativas e as imperativas.

**14.** Explique as diferenças entre um literal, uma constante e uma variável.

**15.** O que é precedência de operador?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Uma vez identificadas as necessidades do usuário em potencial, elas são reunidas para formar um conjunto de **requisitos** a que o novo sistema deve satisfazer. Esses requisitos são expressos na linguagem do usuário, e não na terminologia técnica usada pela comunidade de processamento de dados. Um dos requisitos é que o acesso aos dados fique restrito apenas ao pessoal autorizado. Outro, que os dados representem o estado corrente do cadastro, como, por exemplo, no final do último dia de negócios, ou que a apresentação dos dados na tela do computador seja compatível com o formato do formulário em uso.

Depois de identificados, os requisitos do sistema são traduzidos para a forma de **especificações** mais técnicas. Por exemplo, o requisito de os dados serem restritos apenas ao pessoal autorizado passa a ser uma especificação em que o sistema não responderá se não for digitada, no terminal, uma senha autorizada de oito dígitos; caso contrário, tais dados serão apresentados em forma codificada, até que sejam pré-processados por uma rotina, conhecida apenas por pessoal autorizado.

**Projeto** Enquanto a análise se concentra no *quê* o sistema proposto deve fazer, o projeto se concentra em *como* o sistema atingirá as metas. Aqui a estrutura do sistema de *software* é estabelecida.

É consenso que a melhor estrutura para um grande sistema de *software* é a modular. De fato, é por meio dessa decomposição modular que se torna possível a implementação de sistemas de grande porte. Sem ela, os detalhes técnicos necessários à sua implementação excederiam a capacidade da compreensão humana. Com um projeto modular, porém, somente os detalhes pertinentes ao módulo em questão são considerados de cada vez. Esta modularização também se aplica em futuras manutenções, pois permite fazer alterações com base nos módulos. (Se for feita uma alteração no método de cálculo dos benefícios do plano de saúde de cada funcionário, então apenas os módulos relacionados com este assunto serão considerados.)

Contudo, existem distinções em relação ao conceito de módulo. Se abordarmos a tarefa de projeto em termos do paradigma imperativo tradicional, os módulos irão consistir em procedimentos e o desenvolvimento de um projeto modular passará a ser a identificação das várias tarefas que o sistema proposto deve realizar. Em contraste, se abordamos a tarefa de projeto a partir da perspectiva orientada a objeto, os módulos

## *A engenharia de* software *no mundo real*

O seguinte cenário é comum quanto aos problemas encontrados pelos engenheiros de *software* do mundo real. A Companhia XYZ contrata uma firma de engenharia de *software* para desenvolver e instalar um sistema de *software* integrado para suprir as suas necessidades de processamento de dados. Como parte do sistema produzido pela Companhia XYZ, uma rede de PCs é usada para que os funcionários tenham acesso ao sistema geral da companhia. Assim, cada funcionário tem um PC na sua mesa. Pouco tempo depois, esses PCs são usados não apenas para acessar o novo sistema de gerência de dados, mas também como ferramentas personalizadas, com as quais cada funcionário procura aumentar a sua produtividade. Por exemplo, um funcionário desenvolve um programa de planilha que acelera a execução de suas tarefas. Infelizmente, essas aplicações personalizadas podem não ser bem projetadas, muitas vezes não são completamente testadas e podem incluir recursos que não sejam bem entendidos pelo funcionário. Com o passar dos anos, o uso dessas ferramentas precárias se integra aos procedimentos internos da empresa. (O funcionário que desenvolveu o programa de planilha pode sair da companhia, deixando os outros com um programa que eles não conhecem bem.) O resultado é que o sistema inicial, bem projetado e coerente, pode tornar-se dependente de aplicações sujeitas a erros, mal documentadas e mal projetadas.

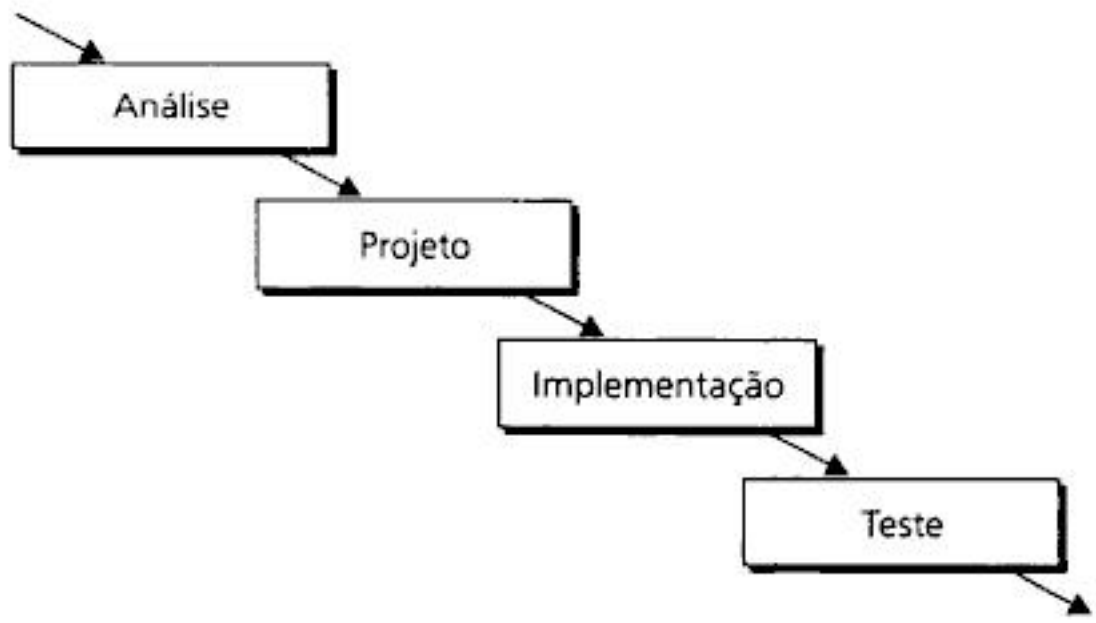



**Figura 6.2** O ciclo de vida do *software*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

rio de pedidos e vice-versa. Essa relação um para um é diferente da relação muitos para um `Pesquisando`, na qual a notação indica que vários clientes podem pesquisar o catálogo simultaneamente. (Retornaremos a essas idéias quando discutirmos os diagramas de entidades e relacionamentos na Seção 6.5.)

Nos últimos anos, um progresso significativo ocorreu na padronização de sistemas de notação para representar os projetos orientados a objeto — o mais proeminente exemplo é a **UML (Unified Modeling Language)**, um sistema para representar diversos conceitos orientados a objeto. A notação usada na Figura 6.4 é baseada na convenção UML.

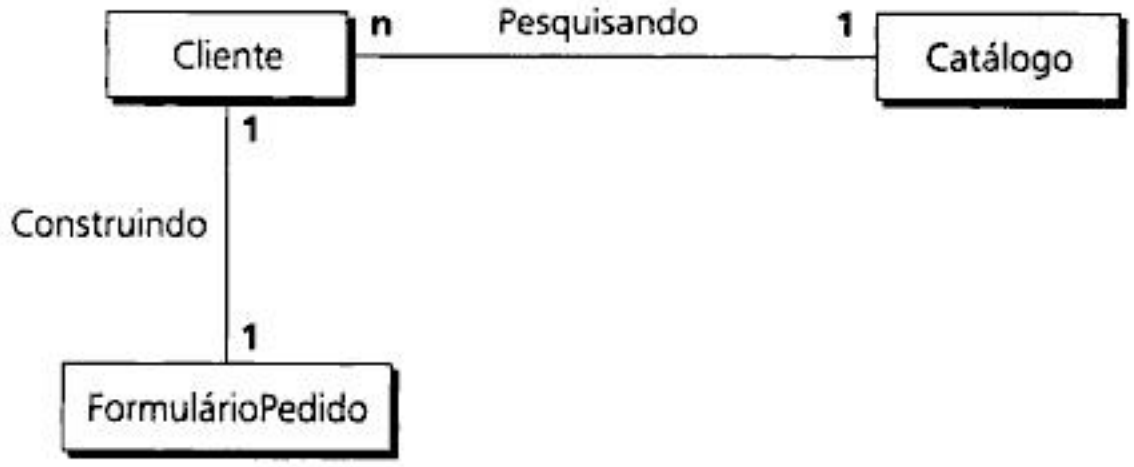



**Figura 6.4** Um diagrama de classes para uma loja virtual na Internet.

## Acoplamento

Introduzimos o conceito de modularidade como forma de obtenção de um *software* administrável. A idéia baseia-se na alta probabilidade de somente alguns dos módulos do *software* terem necessidade de modificação posterior, de forma que podemos restringir as atenções apenas a essa parte do sistema durante o processo de alteração. Isto obviamente depende da validade da hipótese de que as alterações em um dado módulo não irão afetar inadvertidamente outros módulos do sistema. Em conseqüência, o objetivo principal do projeto de um sistema modular deveria ser a maximização da independência intermódulos. Um obstáculo a esta meta reside no fato de serem necessárias conexões entre os módulos para que seja criado um sistema coerente. Tal conexão é denominada **acoplamento**. Maximizar a independência entre os módulos corresponde, portanto, a minimizar seu acoplamento.

O acoplamento entre módulos ocorre, na verdade, de várias maneiras. Uma delas é o **acoplamento de controle**, que ocorre quando um módulo passa o controle a outro, como na relação de transferência e retorno entre um programa e sub-rotinas ou funções. Outro caso é o do **acoplamento de dados**, referente ao compartilhamento de dados entre módulos.

O diagrama estrutural da Figura 6.3 já continha uma representação do acoplamento de controle existente na abordagem de procedimento do nosso exemplo de loja virtual. O acoplamento de dados é tradicionalmente representado em um diagrama estrutural por meio de setas adicionais, conforme ilustra a Figura 6.5. Este quadro indica os elementos de dados que são passados para um módulo quando seus serviços são solicitados pela primeira vez, assim como os elementos de dados retornados ao módulo original quando a tarefa requisitada for completada. O diagrama indica que o módulo `VisãoGeralSítio` receberá informação sobre cada cliente a partir do procedimento `ProcessaNovoCliente` e passará essa informação ao procedimento `VisãoGeralCatálogo`. Além disso,

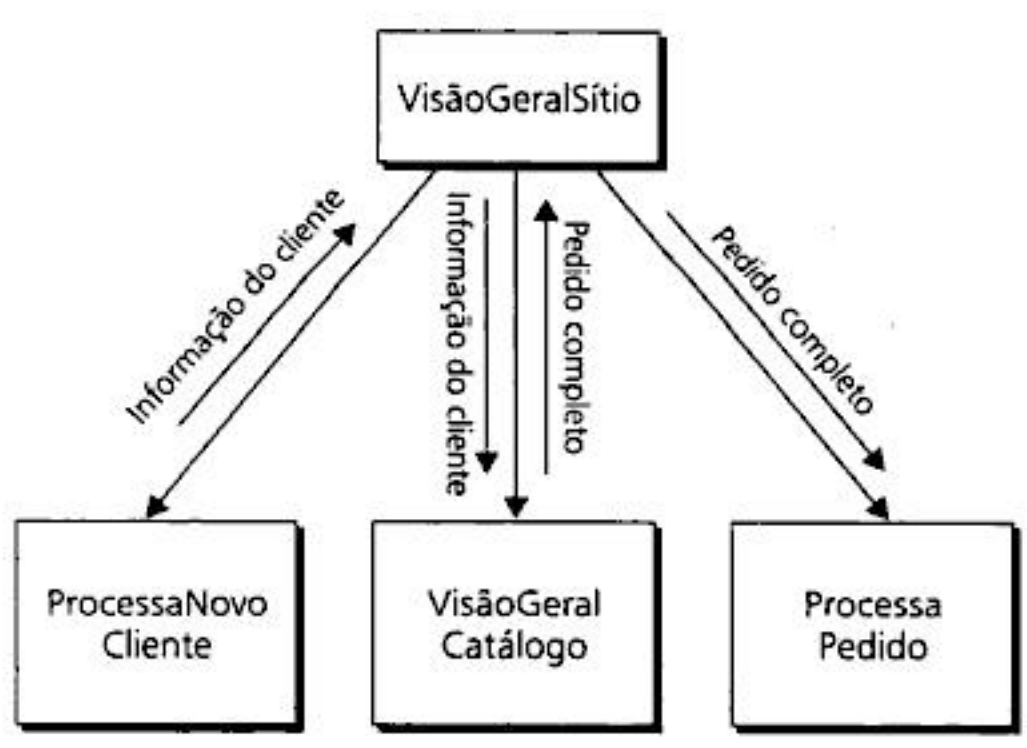



**Figura 6.5** Um diagrama estrutural, que mostra o acoplamento de dados.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

cações. Na prática, é possível que um pacote de *software* se desenvolva por meio de várias extensões em uma única semana.

Como os participantes do desenvolvimento de código aberto não estão organizados e não recebem remuneração, pode-se pensar que não se dedicam o bastante para produzir sistemas grandes e com alto nível de qualidade. Contudo, esta conjectura tem demonstrado ser falsa. O sistema operacional Linux, reconhecido como um dos sistemas operacionais mais confiáveis disponíveis atualmente, constitui um importante exemplo. De fato, Linus Torvald, o criador original do Linux, pode ser considerado o pai do desenvolvimento de código aberto, pois foi com essa metodologia que ele dirigiu o desenvolvimento inicial do Linux.

Um problema que impede o desenvolvimento de código aberto de se tornar uma metodologia mais popular de desenvolvimento de *software* é a dificuldade de uma única entidade manter a propriedade do produto final. De fato, exercer o desenvolvimento de código aberto é liberar o *software* para o domínio público.

Outro problema é a dificuldade de os gerentes treinados na comunidade de negócios aceitarem o estilo "livre" que é aparentemente exigido para que haja sucesso no desenvolvimento do código aberto. Tal desenvolvimento parece apoiar-se em um ambiente no qual o entusiasmo dos indivíduos é fomentado pela expressão de suas habilidades criativas e pelo sentimento de realização e orgulho. Os gerentes, porém, têm mostrado uma tendência para exercer um nível de controle que atenua esse entusiasmo. Por exemplo, um gerente pode querer reter parte do sistema de *software* para garantir a propriedade, mas o desenvolvimento de código aberto se apóia justamente no fornecimento do sistema completo para um indivíduo baixar, usar e eventualmente modificar de acordo como as suas necessidades. Em outros casos, um gerente pode atrasar a liberação de novas versões do *software* para evitar erros embaraçosos, quando na realidade a descoberta desses erros é que faz aumentar o apetite de muitos contribuidores ao desenvolvimento de código aberto.

Independentemente de o desenvolvimento de código aberto tornar-se um modelo popular para o desenvolvimento de *software* comercial, ele deve ser reconhecido como uma metodologia existente e viável que merece a atenção dos pesquisadores desse campo da engenharia de *software*. Sem dúvida, o aprendizado sobre o que faz o desenvolvimento de código aberto funcionar em alguns situações, ainda que falhe em outras, deve levar às investigações que contribuem para o campo como um todo.

### *QUESTÕES/EXERCÍCIOS*

1. Identifique algumas estruturas de *software* discutidas nos capítulos anteriores que poderiam ser consideradas padrões de projeto.
2. Que papel no processo da engenharia de *software* os pesquisadores esperam que as armações desempenhem?
3. Qual é a distinção entre a prototipação evolucionária tradicional e o desenvolvimento de código aberto?

## 6.5 Ferramentas do ofício

A Engenharia de *Software* produziu diversos sistemas notacionais para auxiliar na análise e projeto de sistemas. Desses, já vimos os diagramas estruturais, de classes e de colaboração. Um outro é o **diagrama de fluxo de dados**, representação gráfica das trajetórias de dados em um sistema, isto é, ele identifica a origem, o destino e os pontos de processamento dos dados em um sistema. Os vários símbolos existentes em um diagrama possuem significados específicos: as setas representam as trajetórias dos dados; os retângulos, as fontes e os sorvedouros de dados; os círculos (bolhas), os pontos de manipulação de dados; e as linhas retas grossas, o armazenamento de dados. Em cada caso, o símbolo é

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

a projetar pacotes compatíveis. Por exemplo, no caso de um novo sistema operacional, a distribuição de uma versão beta estimula o desenvolvimento de programas utilitários compatíveis, de forma que o sistema operacional final aparecerá nas prateleiras das lojas cercado de produtos relacionados. Finalmente, a existência do *software* beta pode gerar um sentimento de antecipação no mercado — uma atmosfera que aumenta a publicidade e as vendas.

### *QUESTÕES/EXERCÍCIOS*

1. Quando se testa um *software*, o teste é bem-sucedido quando se encontram erros ou quando não se encontram?
2. Que técnicas você propõe usar para a identificação dos módulos em um sistema que devem ser testados mais exaustivamente?
3. Qual seria um bom teste a ser realizado em um pacote de *software* projetado para ordenar uma lista com 100 elementos no máximo?

## 6.7 Documentação

Um sistema de *software* tem pouca utilidade, a menos que as pessoas possam entendê-lo e mantê-lo. Conseqüentemente, a documentação se mostra uma parte importante de um pacote de *software*, logo o seu desenvolvimento é um tópico não menos importante da engenharia de *software*.

Em geral, a documentação de um pacote de *software* é elaborada com dupla finalidade. A primeira destina-se a explicar as características do *software* e a descrever como usá-lo. Esta é conhecida como a **documentação de usuário**, pois é projetada para ser lida pelo usuário do *software*. Conseqüentemente, a documentação de usuário tende a ser não-técnica.

Atualmente, a documentação de usuário é reconhecida como importante ferramenta de *marketing*. Uma boa documentação de usuário (combinada com uma interface de usuário bem projetada) torna acessível o pacote de *software* e gera, portanto, um aumento em sua vendagem. Reconhecido este ponto, muitos fornecedores de *software* contratam escritores técnicos para trabalhar este lado dos seus produtos, ou fornecem versões preliminares dos mesmos para escritores independentes de manuais, de forma que manuais didáticos, que ensinam o funcionamento do *software*, estejam disponíveis nas livrarias na época do seu lançamento no mercado.

A documentação de usuário é tradicionalmente elaborada na forma de um livro ou manual, mas em muitos casos, a mesma informação é incluída no próprio *software*. Isto permite ao usuário consultar partes do manual através do monitor, enquanto continua a processar o *software*. Neste caso, a informação pode ser dividida em pequenos módulos, às vezes chamados de pacotes de ajuda*, que são apresentados como partes do próprio sistema de *software*. Isto significa que o recurso de acesso a um dos pacotes de ajuda é uma opção construída como parte do sistema de *software*. Em alguns sistemas, os pacotes de ajuda podem entrar automaticamente em atividade na tela do monitor, caso o usuário dê, por muito tempo, a impressão de estar perdido entre os comandos do sistema.

O outro propósito da documentação é o de descrever a composição interna do *software* de forma que, mais tarde, seja possível efetuar a sua manutenção, durante o seu ciclo de vida. Conhecida como **documentação de sistema**, é inerentemente mais técnica do que a documentação de usuário. O componente principal da documentação é a versão fonte de todos os programas do sistema. É essencial

*N. de T. Em inglês, *help*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

CAPÍTULO 7

# *Estruturas de dados*

A memória principal de um computador é organizada na forma de células individuais, com endereços consecutivos. Entretanto, pode ser conveniente associar a tais células outros arranjos de dados. Por exemplo, registros de vendas semanais são mais facilmente visualizados em tabelas nas quais as vendas de diferentes produtos em diferentes dias são organizadas em linhas e colunas. Neste capítulo, analisaremos como são simuladas tais organizações abstratas de dados. O objetivo é permitir ao usuário dos dados raciocinar em termos de organizações abstratas, em vez de se preocupar com a organização real existente na memória principal do computador.



**Os asteriscos indicam sugestões de seções consideradas opcionais.*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

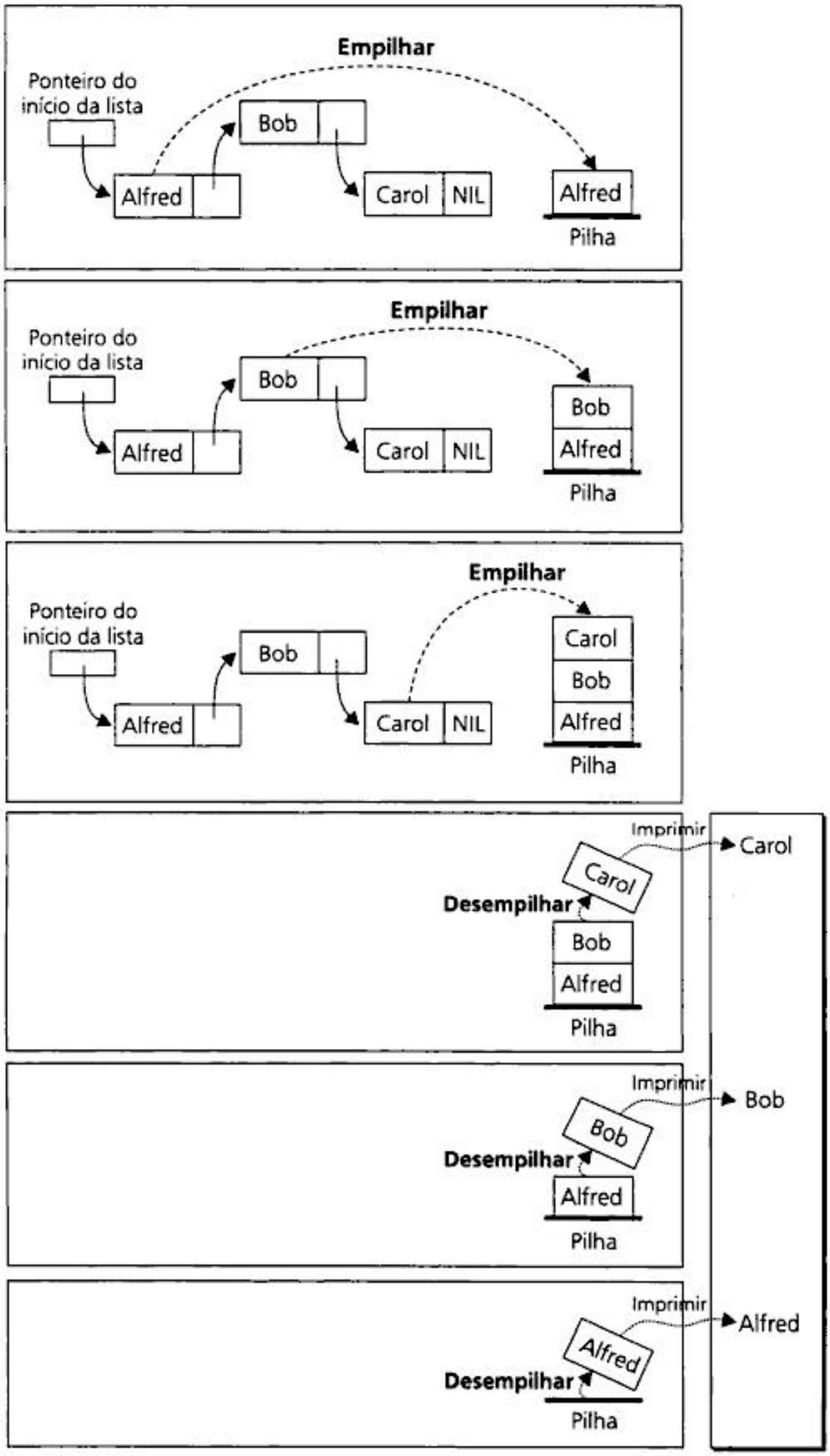



**Figura 7.10** Uso de uma pilha para imprimir, em ordem invertida, uma lista ligada.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

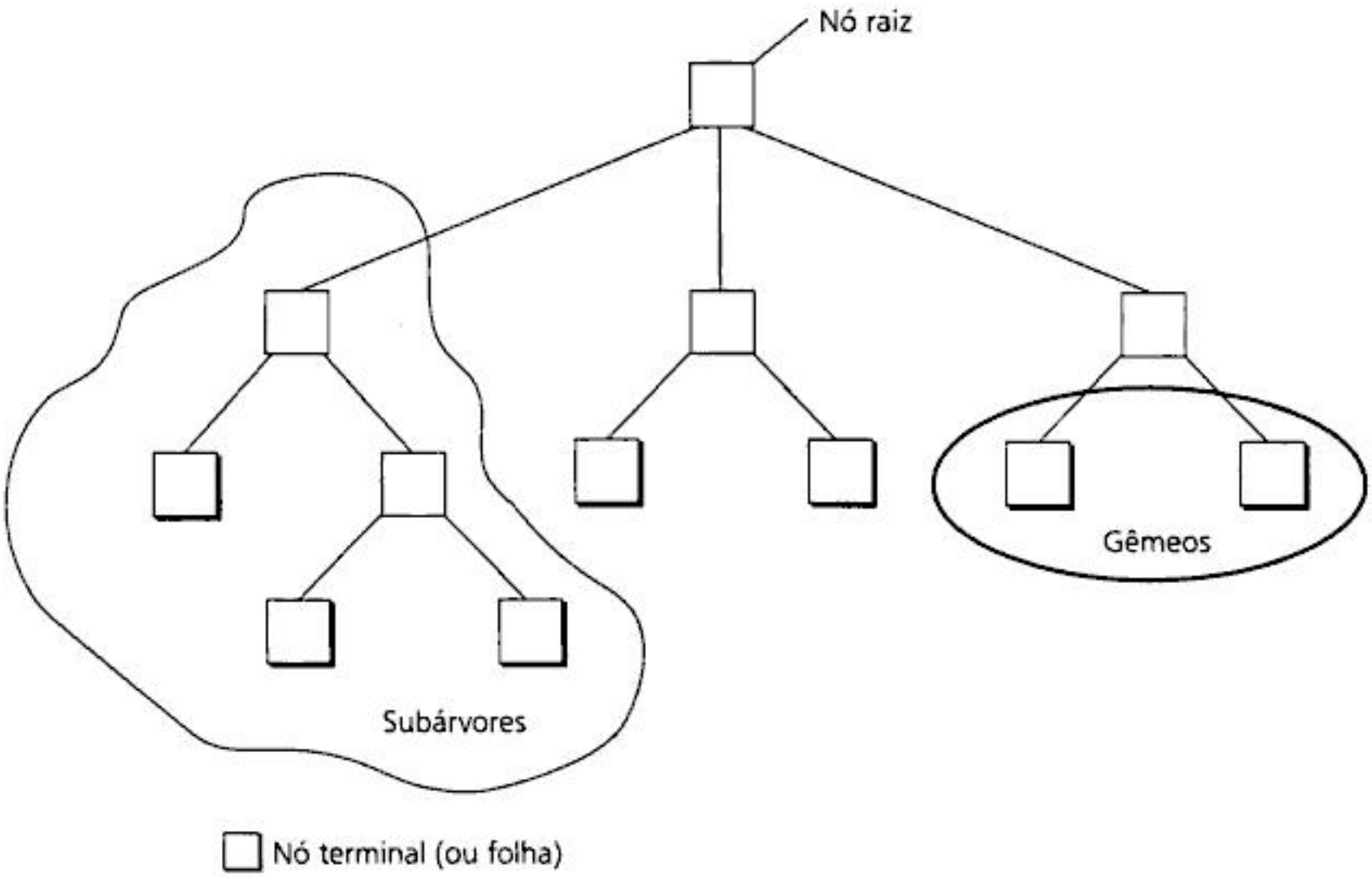



**Figura 7.17** Terminologia de árvores.

tes imediatos como seus **filhos** e a seu ancestral imediato como seu **pai***. Além disso, referimo-nos a dois ou mais nós descendentes do mesmo pai como **gêmeos****. Finalmente, mencionamos com freqüência a **profundidade***** de uma árvore, que é o número de nós compreendidos na mais longa trajetória da raiz até uma folha. Em outras palavras, a profundidade de uma árvore é o número de camadas horizontais nela compreendidas.

Encontraremos diversas vezes as estruturas de árvore nos capítulos subseqüentes, por isso, não apresentaremos qualquer aplicação no momento. Mais adiante, nesta seção, e também no nosso estudo sobre organização indexada, no Capítulo 8, verificaremos que uma informação que deva ser obtida com grande rapidez, na consulta a bancos de dados, muitas vezes é organizada em árvore. No Capítulo 10, verificaremos como os jogos eletrônicos podem ser analisados com o auxílio de árvores.

## Implementação de árvores

Com o propósito de discutir as técnicas de armazenamento de árvores, restringiremos a nossa atenção às **árvores binárias**, nas quais cada nó tem dois filhos no máximo. Tais árvores geralmente são armazenadas na memória usando uma estrutura semelhante à das listas ligadas. Em vez de cada elemento conter dois componentes (o dado e um ponteiro para o próximo elemento), cada elemento (ou nó) da árvore binária contém três: (1) o dado, (2) um ponteiro para o primeiro e (3) outro ponteiro para o segundo filho do nó. Embora não haja um lado esquerdo ou direito na memória de uma

*N. de T. Em inglês, usa-se o termo mais genérico *parent* (significa *genitor*).
**N. de T. Ou ainda, *irmãos*. Em inglês, *twins* ou *siblings*.
***N. de T. Alguns autores preferem o termo *altura* em vez de *profundidade*. Em inglês, *depth*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

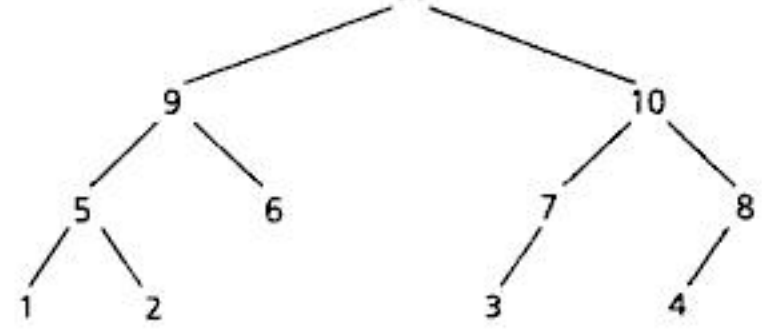
```
procedure Insere(Arvore,NovoValor)

if (Ponteiro do nó-raiz = NIL)
  then (ajustar o ponteiro do nó-raiz para apontar
        para uma nova folha que contém NovoValor)
  else (executar o bloco de instruções abaixo,
        associado com o caso apropriado)
         caso 1: NovoValor = valor do nó-raiz
                 (Nada a fazer)
         caso 2: NovoValor < valor do nó-raiz
                 (if (ponteiro para o filho da esquerda do nó-raiz = NIL)
                       then (ajustar esse ponteiro para apontar para
                             uma nova folha que contém o NovoValor)
                       else (aplicar o procedimento Insere para
                             instalar o NovoValor na subárvore
                             identificada pelo ponteiro para o
                             filho da esquerda)
           caso 3: NovoValor > valor do nó-raiz
                 (if (ponteiro para o filho da direita do nó-raiz = NIL)
                       then (ajustar esse ponteiro para apontar para
                             uma nova folha que contém o NovoValor)
                       else (aplicar o procedimento Insere para instalar
                             o NovoValor na subárvore identificada pelo ponteiro
                             para o filho da direita)
  ) end if
```

**Figura 7.28** Um procedimento para inserir um elemento em uma lista armazenada como árvore binária.

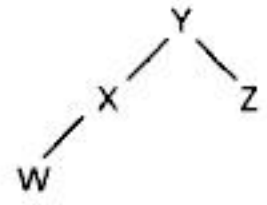
## *QUESTÕES/EXERCÍCIOS*

**1.** Identifique a raiz e os nós de folha da árvore seguinte. Identifique as subárvores abaixo do nó 9. Identifique os grupos de filhos dentro da árvore.




**2.** Qual condição indica que uma árvore ligada, implementada na memória de uma máquina, está vazia?

**3.** Faça um diagrama que represente como a árvore aparece na memória, quando armazenada com o uso de ponteiros para os filhos à esquerda e à direita, conforme foi descrito nesta seção. A seguir, faça um outro diagrama que mostre como tal árvore seria em armazenamento contíguo, utilizando o esquema de armazenamento alternativo, também apresentado nesta seção.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

CAPÍTULO 8

# *Estruturas de arquivo*

No Capítulo 7, discutimos as formas de organização interna dos dados na memória principal da máquina. Neste capítulo, iremos nos concentrar nas técnicas de guardar dados no armazenamento em massa. Um tópico importante é que a maneira como a informação será acessada desempenha um papel importante na forma como ela deve ser armazenada. De forma semelhante ao caso das estruturas de dados, verificamos que o formato apresentado finalmente ao usuário pode não ser o mesmo do sistema de armazenamento. Assim como foi feito no Capítulo 7, aqui discutiremos e compararemos as organizações conceituais e as reais.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
procedimento Intercala (ArquivoEntradaA, ArquivoEntradaB, ArquivoSaida)

se (dois arquivos chegaram ao fim) então (Parar, com ArquivoSaida vazio)
se (ArquivoEntradaA não estiver no fim) então (Declarar seu primeiro registro como registro corrente)
se (ArquivoEntradaB não estiver no fim) então (Declarar seu primeiro registro como registro corrente)
enquanto (nenhum dos arquivos de entrada chegar ao fim) faça
   (Colocar o registro corrente com o "menor" valor do campo chave no ArquivoSaida;
    se (esse registro corrente for o último em seu arquivo)
        então (Declarar que esse arquivo de entrada chegou ao fim)
        senão (Declarar o próximo registro nesse arquivo de entrada como registro corrente)
   )
Iniciando com o registro corrente do arquivo que não chegou ao fim,
    copiar os restantes para o ArquivoSaida.
```

**Figura 8.3** Um procedimento para intercalar dois arquivos seqüenciais.

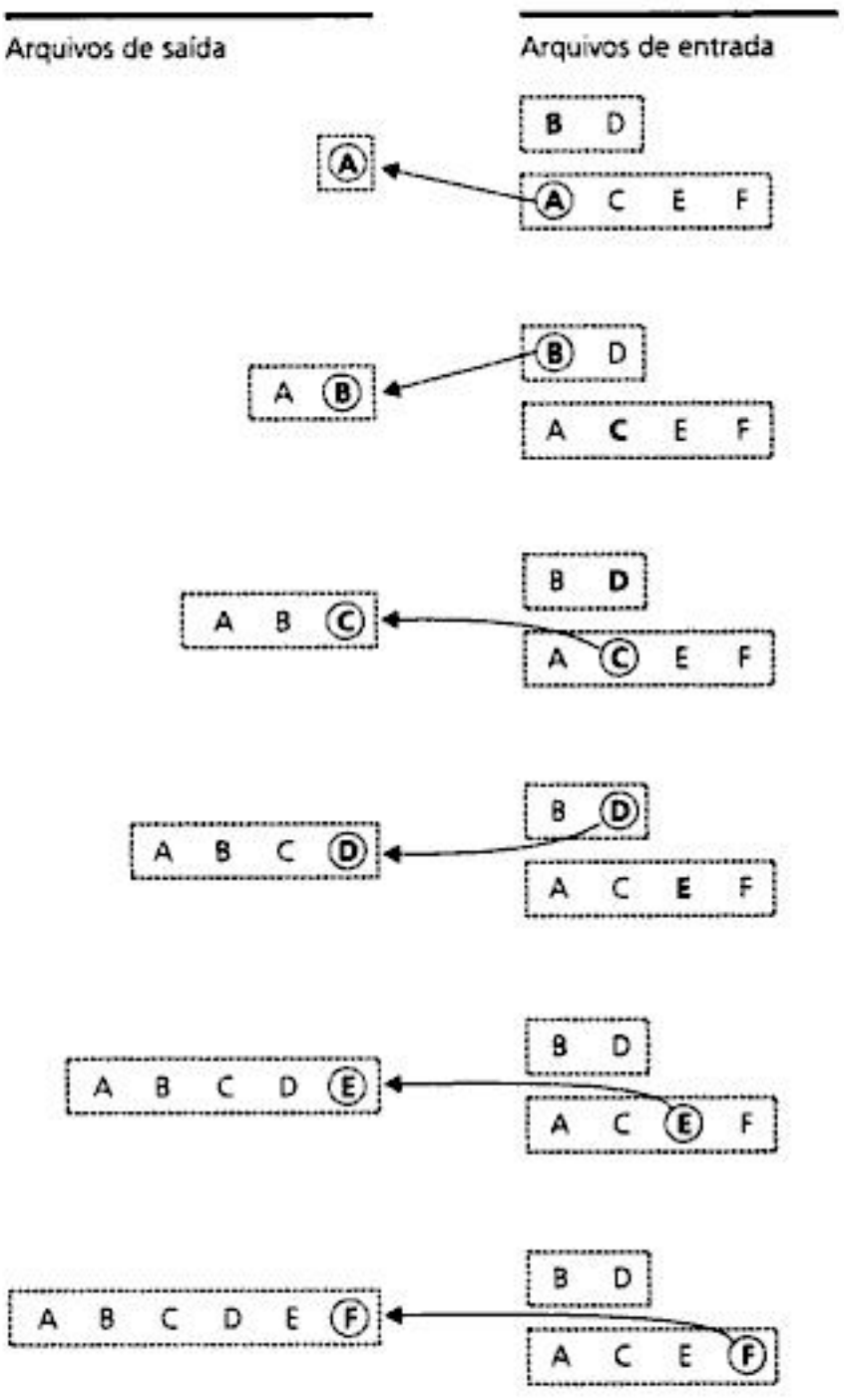



**Figura 8.4** Aplicação do algoritmo de intercalação. (As letras são usadas para representar registros completos. Cada letra representa o valor do campo chave no registro.)

Os registros lógicos em um arquivo freqüentemente são identificados por um único campo. Por exemplo, em um arquivo de funcionários, o campo pode ser o que contém o número de Seguro Social ou talvez o de identificação do funcionário. Tal campo é chamado **campo chave**. Armazenando um arquivo seqüencial de acordo com o campo chave, pode-se reduzir drasticamente o tempo de processamento. Por exemplo, suponha que o processamento da folha de pagamentos necessite que cada registro de funcionário seja atualizado para refletir a informação de horas trabalhadas por ele. Se o arquivo que contém as horas trabalhadas está armazenado de acordo com o mesmo campo chave dos registros de funcionários, então esse processo de atualização pode ser feito acessando os arquivos seqüencialmente, usando as horas trabalhadas recuperadas de um arquivo para atualizar o registro correspondente no outro. Se os arquivos não estivessem arranjados por campo chave, o processo de atualização implicaria a leitura de cada registro de um arquivo e na busca repetida ao registro correspondente no outro.

Assim, a atualização de arquivos seqüenciais clássicos normalmente é feita em múltiplos passos. Primeiro, a nova informação (como a coleção de horas trabalhadas) é gravada em um arquivo seqüencial conhecido como arquivo de transações e este arquivo é ordenado para se compatibilizar com o arquivo a ser atualizado, que é chamado arquivo-mestre. Então, os registros do arquivo-mestre são atualizados lendo-se os registros dos dois arquivos seqüencialmente.

Outro exemplo clássico de processamento em arquivos seqüenciais é o de intercalar dois arquivos para formar um novo que contenha os registros de ambos. Os registros nos arquivos de entrada são dispostos em ordem crescente de acordo com um campo chave comum, e o arquivo de saída deve possuir essa mesma propriedade. O algoritmo clássico de intercalação está resumido na Figura 8.3. O tema subjacente é construir o arquivo de saída à medida que os arquivos de entrada são varridos seqüencialmente (Figura 8.4).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

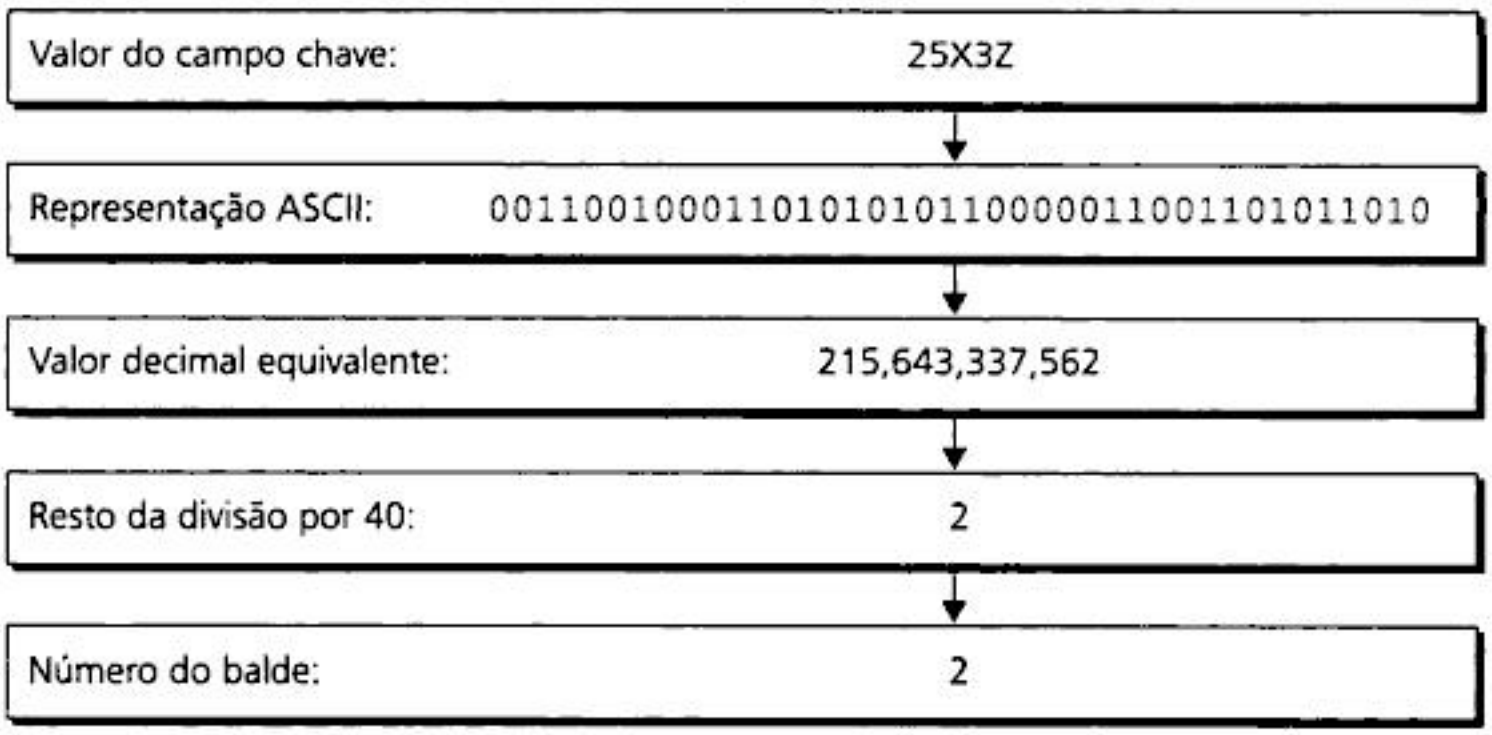



**Figura 8.12** *Hashing* do valor de campo chave 25X3Z, sobre um dos 40 baldes.

observação deste fato sugere uma solução parcial: selecionar um número primo de baldes — minimizando assim a possibilidade de fatores comuns. Por exemplo, a chance de aglomerações no exemplo do arquivo de funcionários pode ser drasticamente reduzida usando-se 41 baldes, em vez de 40.

O fenômeno de duas chaves serem transformadas no mesmo valor é chamado **colisão**. As colisões são o primeiro passo para os aglomerados e, portanto, devem ser evitadas. Infelizmente, a teoria das probabilidades nos diz que as colisões são altamente prováveis, do mesmo quando o número de baldes é maior do que o de elementos armazenados. Para entender essa aparente contradição, vamos considerar o que acontece quando os elementos são inseridos em um sistema *hashing* de armazenamento com 41 baldes vazios.

Suponha que encontremos uma função *hash* que distribua arbitrariamente os registros entre os baldes, que nosso sistema de armazenamento esteja vazio e que iremos inserir os registros um por vez. Quando inserirmos o primeiro, este deverá ser guardado em um balde vazio. No entanto, quando inserirmos o próximo, só 40 dos 41 baldes estarão vazios, e assim a probabilidade de o segundo registro ser colocado em um balde vazio é de apenas $^{40}/_{41}$. Supondo que o segundo registro seja colocado em um balde vazio, o terceiro encontrará apenas 39 baldes vazios, e a probabilidade de ser colocado em um deles será de $^{39}/_{41}$. Prosseguindo neste processo, se os sete primeiros registros forem colocados em baldes vazios, o oitavo terá a probabilidade de $^{34}/_{41}$ de ser colocado em um dos baldes vazios restantes.

Esta análise nos permite calcular a probabilidade de os primeiros oito registros serem colocados em baldes vazios, pois é o produto das probabilidades de cada registro ser colocado em um balde vazio, supondo que os registros anteriores tenham sido colocados desta forma. Esta probabilidade é então:

$$(^{41}/_{41})\,(^{40}/_{41})\,(^{39}/_{41})\,(^{38}/_{41})\ldots(^{34}/_{41}) = 0{,}482$$

O resultado obtido é menor que meio. Isto significa que é mais provável que pelo menos dois dos primeiros oito

## *Autenticação por meio do* hashing

Embora tenhamos introduzido as técnicas de *hashing* no contexto da construção de estruturas eficientes de armazenamento, o *hashing* não se restringe a esse ambiente. Por exemplo, ele pode ser usado como um meio de autenticação de mensagens transmitidas pela Internet. O tema subjacente é transformar a mensagem (de uma maneira secreta) e criptografar o valor obtido. Esse valor é então transferido junto com a mensagem. Para autenticá-la, o receptor transforma a mensagem recebida (da mesma maneira secreta) e confirma que o valor produzido concorda com o original. Se eles não concordarem, pressupor-se-á que a mensagem é corrompida. Sob esse foco, a maioria das técnicas de detecção de erros pode ser considerada como aplicação de *hashing*. Por exemplo, o sistema de paridade simplesmente transforma cada padrão de *bits* em 0 ou 1 e envia esse resultado juntamente como o padrão de *bits* para ser usado no processo de autenticação.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

### *QUESTÕES/EXERCÍCIOS*

1. Qual a diferença entre uma transação que atingiu o seu ponto de comprometimento e outra que não chegou a tanto?
2. Como um SGBD poderia resguardar-se de extensas anulações em cascata?
3. Mostre de que forma a alternância descontrolada entre duas transações, sendo que uma debita $100 de uma conta e a outra debita $200 da mesma conta, poderia produzir saldos finais de $100, $200 e $300, supondo que o saldo inicial seja de $400.
4. a. Resuma os possíveis resultados de uma transação que solicita acesso compartilhado a um elemento de um banco de dados.
   b. Resuma os possíveis resultados de uma transação que solicita acesso exclusivo a um elemento de um banco de dados.
5. Descreva uma seqüência de eventos capaz de levar a um enlace mortal entre transações que estejam executando operações em um sistema de banco de dados.
6. Descreva como o enlace mortal da sua resposta à questão 5 poderia ser eliminado. Sua solução faz uso do diário do sistema de gerência do banco de dados? Justifique a sua resposta.

## 9.6 Impacto social da tecnologia de banco de dados

No passado, as coleções de dados eram vistas como entidades inertes e passivas. Cada uma era projetada e usada com um propósito específico. Uma biblioteca local mantinha uma lista física com nomes e endereços de seus usuários. Cada livro continha um cartão removível com o nome do livro. Quando ele era emprestado, o cartão era removido, o nome do usuário era registrado no cartão e este era arquivado na biblioteca. Quando o livro retornava, o cartão era colocado de volta; se o livro não retornasse no prazo, os funcionários da biblioteca poderiam identificar o usuário por meio de sua lista física.

Com esse sistema manual, era possível obter uma lista de todos os livros emprestados a um único indivíduo. Entretanto, isso exigia procurar em todos os livros da biblioteca para ver que cartões continham o nome do indivíduo, e o custo dessa pesquisa a tornava impraticável. Assim, embora os registros da biblioteca contivessem informações sobre os hábitos de leitura de seus usuários, estes ficavam seguros de que tal informação era privada. Atualmente, porém, a maioria das bibliotecas é informatizada e o perfil do hábito de leitura de um indivíduo é fácil de acessar. Agora é possível que as bibliotecas forneçam tais informações a empresas comerciais, agências locais, partidos políticos, empregadores e indivíduos diversos. As ramificações são enormes.

Esse exemplo da biblioteca é representativo do potencial que permeia o espectro completo das aplicações de banco de dados. A tecnologia facilitou a coleta de dados e a fusão ou comparação de diferentes coleções de dados para obter relacionamentos que de outra forma ficariam escondidos.

As coleções de dados agora são conduzidas em larga escala. Em alguns casos, o processo é prontamente aparente; em outros, é mais sutil. Exemplos dos primeiros ocorrem quando as solicitações de informação são explícitas. Isso pode ser feito de maneira voluntária, por meio de formulários de pesquisa, ou de maneira involuntária, quando imposto por exigência legal. Algumas vezes, o fato de ser ou não voluntário depende do ponto de vista. O fornecimento de informação pessoal, quando se reivindica um empréstimo, é voluntário ou involuntário? A distinção depende de o empréstimo ser uma conveniência ou uma necessidade. Para usar cartão de crédito em algumas lojas, atualmente se exige que a assinatura seja registrada em um formato digitalizado. Mais uma vez, o fornecimento da informação é voluntário ou involuntário, dependendo da situação do comprador.

Casos mais sutis de coleções de dados evitam a comunicação direta com os envolvidos. Exemplos incluem uma companhia de crédito que registra as práticas de compra dos usuários de seus cartões, um

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

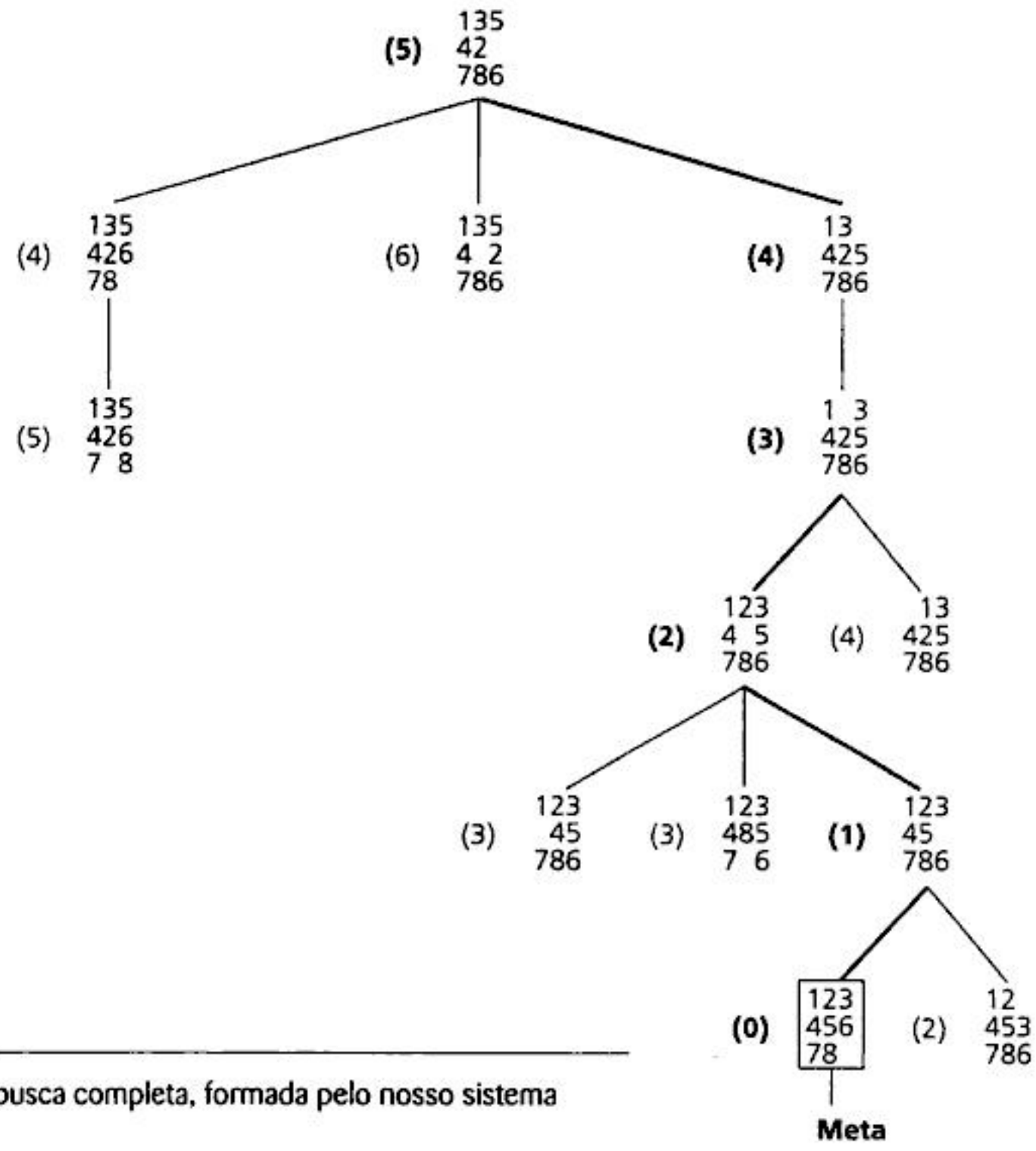



**Figura 10.13** A árvore de busca completa, formada pelo nosso sistema heurístico.

## QUESTÕES/EXERCÍCIOS

1. Qual é a importância dos sistemas de produções na inteligência artificial?
2. Faça uma parte do grafo de estados do quebra-cabeças de oito peças que cerca o nó que representa o seguinte estado:

| | | |
|---|---|---|
| 4 | 1 | 3 |
| | 2 | 6 |
| 7 | 5 | 8 |

3. Utilizando o método de busca em largura, desenhe a árvore de busca construída por um sistema de controle ao resolver o quebra-cabeças de oito peças com o seguinte estado inicial:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 8 | 5 |
| 7 | 6 | |

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

aproximando da saída desejada. Como este processo é repetido sobre uma amostra de entradas, espera-se que os pesos precisem ser cada vez menos reajustados, até que a rede comece a operar corretamente sobre todos os dados da amostra. É necessária uma estratégia de reajuste desses pesos para que, a cada novo ajuste, a rede vá se aproximando da meta final, em vez de destruir eventuais avanços obtidos a partir das amostras anteriores.

## Memória associativa

Uma característica impressionante da mente humana é a habilidade de recuperar a informação associada a um tema corrente. Quando experimentamos certos aromas, rapidamente relembramos nossa

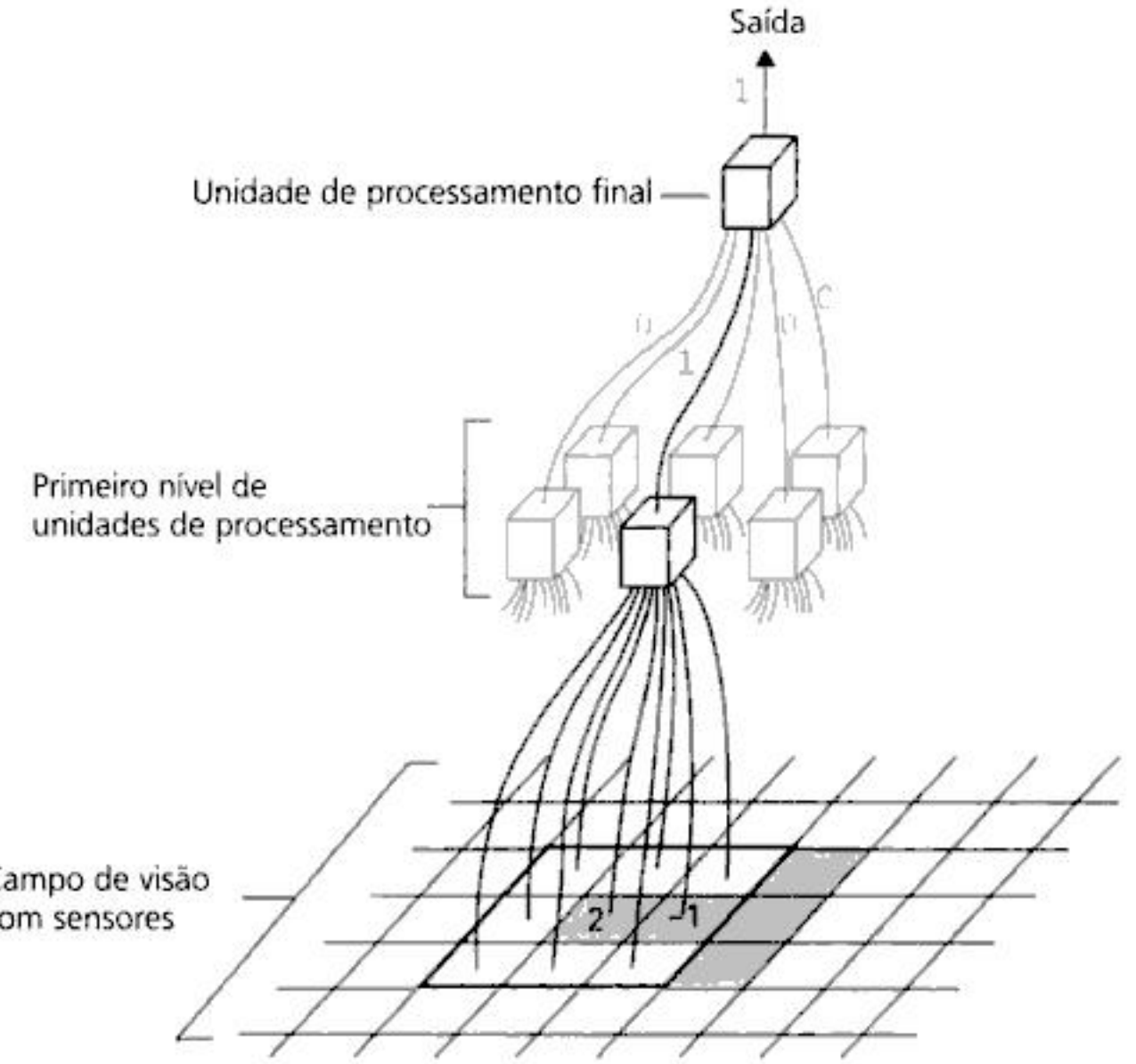



**Figura 10.22** A letra T no campo de visão.

### *Pequena escala à grande escala*

Um fenômeno recorrente no desenvolvimento e teste de teorias é a transição da experimentação em pequena escala para as aplicações em grande escala. A experimentação inicial de uma nova teoria freqüentemente envolve casos pequenos e simples. Se o sucesso é alcançado, então o ambiente experimental é estendido para sistemas mais realistas, em grande escala. Algumas teorias são aptas a sobreviver nessa transição; outras não. Às vezes, o sucesso na pequena escala é grande o suficiente para encorajar os proponentes da teoria a persistir após as falhas na grande escala terem desencorajado outros pesquisadores. Em alguns casos, essa persistência finalmente vale a pena; em outros, representa esforço desperdiçado.

Tais cenários são prontamente observados no campo da inteligência artificial. Um exemplo está na área de processamento de linguagens naturais, na qual sucessos antigos em situações limitadas levaram muitos a acreditar que o entendimento geral de linguagens naturais estava apenas pouco acima no horizonte. Infelizmente, estender o sucesso para a grande escala demonstrou ser muito mais difícil, e tem-se vencido morosamente como resultado de esforços significativos. Outro exemplo é o tema das redes neurais artificiais, tópico que entrou em cena com uma pompa significativa, declinou durante anos quando as suas capacidades de grande escala foram questionadas e agora retorna em uma atmosfera mais conquistadora. Como indicado no texto, o tema dos algoritmos genéticos atualmente está realizando o teste da transição. Se a abordagem evolucionária provará ser ferramenta útil no futuro, é uma questão em aberto.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Assim, um problema fundamental da inteligência artificial é de banco de dados, e o progresso em cada campo é aplicado ao outro. As técnicas da inteligência artificial são aplicadas aos sistemas tradicionais de bancos de dados para proporcionar melhores serviços, e as técnicas de bancos de dados, aos projetos de inteligência artificial, em uma tentativa de tratar as enormes massas de conhecimento do mundo real subjacentes aos processos de decisão envolvidos. Em outros casos, os esforços de pesquisa em inteligência artificial e em sistemas de bancos de dados atacam o mesmo problema a partir de perspectivas diferentes.

Um exemplo é o problema de desenvolver os sistemas de memória associativa (sistemas de bancos de dados associativos). Já vimos a necessidade de tais sistemas no campo do processamento de linguagens naturais, onde a informação relevante do mundo real deve ser aplicada para entender sentenças de acordo com seus contextos, e discutimos como as redes neurais artificiais têm sido aplicadas ao problema.

Visto da perspectiva mais tradicional de banco de dados, o problema é identificar e recuperar a informação relacionada a um tópico, em vez da informação que está sendo explicitamente solicitada. Considere, por exemplo, o problema de quem navega na Web procurando informação na rede. A abordagem tradicional é exigir que a pessoa identifique palavras-chave e frases que devem aparecer nos documentos relevantes. O sistema então procura em seu banco de dados e recupera os documentos que contêm tais palavras e frases. Infelizmente, esse sistema nada mais é do que uma peneira baseada em sintaxe, e não na semântica. Ele pode desprezar os documentos mais importantes porque lidam com "carros" em vez de "automóveis". O que o navegante realmente deseja é um sistema de recuperação com inteligência para identificar matérias relacionadas (ou associadas), em vez de simplesmente recuperar o que foi explicitamente solicitado.

Como outro exemplo, suponha que tenhamos um banco de dados com os cursos conduzidos pelos professores de uma universidade, juntamente com as notas dadas aos estudantes. Consideremos a seguinte seqüência de eventos: solicitamos ao banco de dados o número de notas A atribuídas pelo professor Johnson no semestre passado. O banco de dados responde "nenhuma". Concluímos que o professor Johnson era um instrutor muito exigente. A seguir, solicitamos o número de notas F por ele atribuídas no semestre passado. Novamente, o banco de dados responde "nenhuma". Concluímos que o professor Johnson considera médios todos os estudantes, sem que haja alguém nos extremos. Então solicitamos o número de notas C dadas por ele neste mesmo período. O banco de dados novamente responde "nenhuma". Neste momento, surge a dúvida, e perguntamos se o professor Johnson lecionou algum curso no semestre passado. O banco de dados responde "não". Se ao menos tivesse dito isso logo no início! Gostaríamos de ter um banco de dados que recuperasse a informação que necessitamos, em vez da que solicitamos.

Outro tópico perseguido pela pesquisa tanto em banco de dados como na inteligência artificial é o desenvolvimento de sistemas de armazenamento e recuperação de dados que podem fornecer a informação derivada dos dados armazenados, em vez de meramente responder com a informação explicitamente guardada. Em outras palavras, gostaríamos que o banco de dados fosse capaz de raciocinar com a informação nele contida. Considere, por exemplo, um banco de dados que contenha informações sobre os presidentes dos Estados Unidos. Quando consultado sobre a existência de algum presidente com 3 m de altura, um sistema tradicional não seria capaz de fornecer a resposta, a menos que a altura de cada presidente estivesse armazenada no banco de dados. Por outro lado, um sistema inteligente poderia responder corretamente, sem saber a altura de cada um. A linha de raciocínio poderia ser a seguinte: se tivesse existido um presidente com 3 m de altura, isto teria sido importante e, por isso, armazenado no banco de dados. Portanto, desde que nenhum presidente foi registrado com 3 m de altura, então tal presidente nunca existiu.

A conclusão de que não houve presidentes com 3 m de altura envolve um conceito importante no projeto de banco de dados — a diferença entre bancos de dados em universo fechado e bancos de dados em universo aberto. Falando abertamente, um **banco de dados em universo fechado** é o que assume a postura de conter todos os fatos verdadeiros sobre o tópico em questão, enquanto um banco de dados em universo aberto não adota tal hipótese. A capacidade de rejeitar a hipótese de um presidente

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

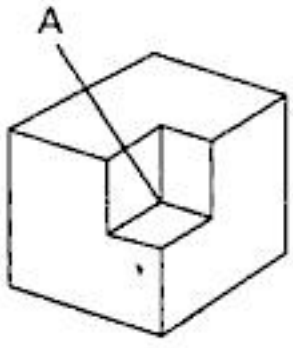
**5.** Se um pesquisador utilizar modelos computacionais para estudar as capacidades e os processos de memorização da mente humana, os programas desenvolvidos para a máquina necessariamente realizarão o trabalho de memorização no limite das suas possibilidades? Explique.

**6.** Quais das seguintes atividades você espera que sejam orientadas ao desempenho e quais à simulação?
   a. o projeto de um simulador de vôo
   b. o projeto de um sistema de piloto automático
   c. o projeto de um banco de dados que lida com materiais de biblioteca
   d. o projeto de um modelo econômico de uma nação para testar teorias
   e. o projeto de um programa para monitorar os sinais vitais de um paciente

**7.** Identifique um pequeno conjunto de propriedades geométricas a serem usadas para distinguir entre os símbolos O, G, C e Q.

**8.** Descreva as semelhanças entre a técnica de identificar características, comparando-as com padrões, e os códigos de correção de erros discutidos no Capítulo 1.

**9.** Descreva duas interpretações do seguinte desenho, com base em se o "vértice" A é convexo ou côncavo:




**10.** No contexto de um sistema de produções, qual a diferença entre um grafo de estados e uma árvore de busca?

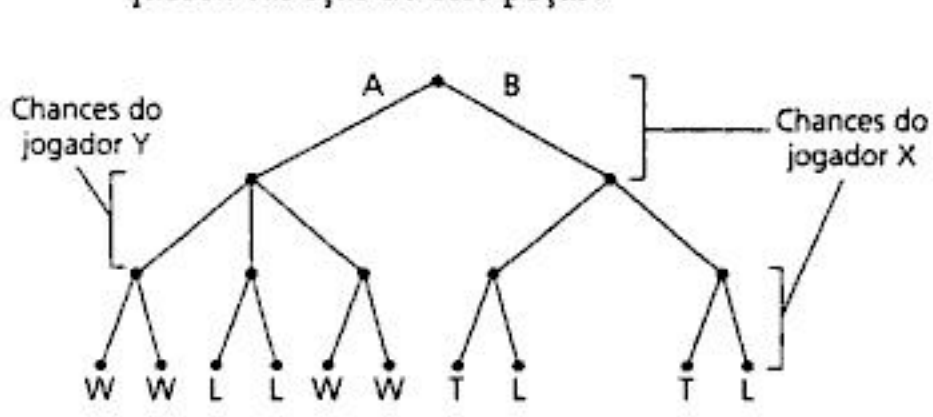
**11.** Analise a tarefa de resolver o cubo de Rubik como um sistema de produções. (Quais são os estados, as produções etc?)

**12.** Analise a tarefa de investir no mercado de capitais como um sistema de produções. (Quais são os estados, as produções etc?) Que heurística poderia ser usada para ajudar o sistema de controle a tomar decisões?

**13.** Analise a tarefa de jogar uma partida de golfe como um sistema de produções. (Quais são os estados, as produções etc?) Que heurística poderia ser usada para ajudar na decisão de que produção aplicar?

**14.** No texto, mencionamos que um sistema de produções geralmente é usado como técnica para tirar conclusões a partir de fatos conhecidos. Os estados do sistema são fatos tidos como verdadeiros em cada fase do processo de raciocínio, e as produções, as regras lógicas destinadas a manipular tais fatos. Identifique algumas regras lógicas que levam à seguinte conclusão: *John is tall** a partir dos fatos: *John is a basketball player*, *Basketball players are not short* e *John is either short or tall.***

**15.** A seguinte árvore representa os possíveis movimentos em um jogo competitivo, mostrando que o jogador X pode, no momento, escolher entre os movimentos A e B. Após o movimento do jogador X, o jogador Y tem direito a um movimento e, em seguida, o jogador X tem direito ao último movimento do jogo. Os nós-folha da árvore estão etiquetados com W, L ou T, conforme o resultado obtido pelo jogador X seja ganho, perda ou empate, respectivamente. O jogador X deveria escolher o movimento A ou B? Por quê? Em que sentido a tarefa de selecionar uma "produção" em uma atmosfera competitiva difere da de um jogo individual, como é o caso do quebra-cabeças de oito peças?




**16.** Analise o jogo de damas como um sistema de produções e descreva uma heurística que poderia ser utilizada para determinar qual de dois estados está mais próximo do objetivo. Como o sistema de controle nessa situação difere de um

---

*N. de T. Em português, *John é alto.*

**N. de T. Em português, *John é um jogador de basquete, Jogadores de basquete não são baixos, John é baixo ou é alto.*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ser realizado limpando primeiro a variável-destino e, em seguida, incrementando-a um número apropriado de vezes. De fato, já tivemos a oportunidade de observar que a seqüência

```
clear Z;
while X not 0 do;
   incr Z;
   decr X;
end;
```

transfere para Z o valor associado a X. No entanto, esta seqüência tem o efeito colateral de destruir o valor original de X. Para corrigir isso, podemos introduzir uma variável auxiliar, para a qual transferimos primeiramente o valor em questão a partir de sua posição inicial. Utilizamos, então, esta variável auxiliar para restabelecer a original, ao mesmo tempo em que depositamos no destino desejado o valor em questão. Desta maneira, o movimento de Hoje para Amanha pode ser realizado pela seqüência descrita na Figura 11.5.

Adotamos a sintaxe

copy *nome1* to *nome2*;

(onde *nome1* e *nome2* representam nomes de variáveis) como uma notação abreviada para uma estrutura de instruções da forma mostrada na Figura 11.5. Assim, embora Bare Bones não tenha uma instrução explícita de cópia de dados (*copy*), freqüentemente escrevemos programas desta forma, sempre levando em conta que, para converter tais programas informais em programas escritos em Bare Bones, é preciso substituir os comandos *copy* pelas suas estruturas while-end equivalentes, utilizando uma variável auxiliar, cujo nome deverá ser diferente de qualquer outro já utilizado no programa.

## A universalidade da linguagem Bare Bones

Vamos agora aplicar a tese de Church-Turing para confirmar a nossa afirmação de que Bare Bones é uma linguagem de programação universal. Primeiro, observamos que qualquer programa escrito em Bare Bones pode ser entendido como especificação da computação de uma função. A entrada da função consiste nos valores atribuídos às variáveis antes da execução do programa, e a saída, nos valores das variáveis quando o programa termina. Para computar a função, simplesmente executamos o programa, atribuindo inicialmente os valores apropriados às variáveis, e então observamos os valores destas quando o programa termina.

Sob estas condições, o programa

```
incr X;
```

descreve a mesma função (a função sucessor) descrita pelo exemplo da máquina de Turing da Seção 11.2. Na verdade, ela aumenta em uma unidade o valor associado a X. Do mesmo modo, considerando as variáveis X e Y como entradas e a variável Z como a saída, o programa abaixo descreve a função adição:

```
copy Y to Z;
while X not 0 do,
   incr Z;
   decr X;
end;
```

Os pesquisadores mostraram que a linguagem de programação Bare Bones pode ser utilizada para expressar algoritmos que computam todas as funções Turing-

```
clear Aux;
clear Amanha;
while Hoje not 0 do;
 incr Aux;
 decr Hoje;
end;
while Aux not 0 do;
 incr Hoje;
 incr Amanha;
 decr Aux;
end;
```

**Figura 11.5** Uma implementação da instrução "copy Hoje to Amanha" na linguagem Bare Bones.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
while X not 0 do;
end;
```

produzindo assim um novo programa. Dessa maneira, o programa deverá obrigatoriamente terminar ou não por si mesmo. Estamos, porém, prestes a verificar que não acontecerá uma nem outra coisa.

Se este fosse um programa que terminasse por si mesmo, e se o processássemos com a entrada igual à sua própria versão codificada, então quando sua execução alcançasse a instrução while que acabamos de adicionar, a variável X conteria o valor 1. (Neste ponto, o novo programa seria idêntico ao original, que produzia o valor 1 se sua entrada fosse a representação de um programa que terminasse por si mesmo.) Aqui, a execução do programa estaria presa para sempre na estrutura while-end porque não tomamos qualquer providência para decrementar a variável X no interior desta iteração. Entretanto, isso contradiz a nossa hipótese de que o novo programa termina por si mesmo. Então devemos concluir que ele não termina por si mesmo.

Entretanto, se admitirmos que este novo programa não termina por si mesmo e o executarmos com suas variáveis iniciadas com a sua própria representação codificada, ele alcançará a instrução while adicionada, com o valor 0 na sua variável X. (Isto acontece porque as instruções que precedem a instrução while constituem o programa original que produz uma saída 0 quando sua entrada representa um programa que não termina por si mesmo.) Neste caso, a iteração representada pela estrutura while-end não será ativada e, portanto, o programa deverá parar. No entanto, esta é a propriedade dos programas que terminam por si mesmos, portanto, seríamos forçados a concluir que o novo programa é um programa que termina por si mesmo, contradizendo a nossa hipótese, da mesma forma que ocorreu no caso anterior.

Em resumo, estamos às voltas com uma situação impossível de um programa que por um lado deve simultaneamente terminar e não terminar e, por outro, não deve terminar nem não terminar. Conseqüentemente, a hipótese que conduziu a esse dilema tem de ser falsa.

Concluímos que a função de parada é incomputável, e uma vez que a solução do problema da parada depende da computação dessa função, devemos concluir que a resolução do problema da parada permanece além da capacidade de qualquer sistema algorítmico. Tais problemas são chamados problemas insolúveis.

Para terminar, devemos relacionar com as idéias do Capítulo 10 aquilo que acabamos de discutir. Uma importante questão subjacente é se as capacidades das máquinas de computação incluem ou não aquelas exigidas pela própria inteligência. Lembre-se de que as máquinas podem resolver apenas os problemas com solução algorítmica, e agora vimos que existem problemas sem solução algorítmica. Portanto, a questão é se a inteligência natural incorpora algo mais que a execução de processos algorítmicos. Se não incorpora, então os limites que identificamos aqui também são os do pensamento humano. É desnecessário dizer que esta é uma questão altamente polêmica e, às vezes, emocional. Se, por exemplo, a mente humana nada mais é do que uma máquina programada, poderíamos concluir que os homens não possuem livre arbítrio.

## QUESTÕES/EXERCÍCIOS

**1.** O programa abaixo, em Bare Bones, termina por si mesmo? Justifique a sua resposta

```
incr X;
decr Y;
```

**2.** O programa abaixo, em Bare Bones, termina por si mesmo? Justifique a sua resposta.

```
copy X to Y;
incr Y;
incr Y;
while X not 0 do;
   decr X;
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
procedimento OrdenaIntercala(Lista)

se (Lista possuir mais de um elemento)
   então (Aplicar o procedimento OrdenaIntercala para ordenar a primeira metade da Lista;
          Aplicar o procedimento OrdenaIntercala para ordenar a segunda metade da Lista;
          Aplicar o procedimento IntercalaListas para intercalar a primeira e segunda
             metades da Lista, produzindo a versão ordenada da Lista
         )
```

**Figura 11.9** Algoritmo de ordenação por intercalação implementado como procedimento OrdenaIntercala.

estrutura de árvore mostrada na Figura 11.10, onde cada nó da árvore apresenta um único problema no processo recursivo e os ramos abaixo dele representam os problemas menores derivados do ancestral. Por isso, encontramos o número total de comparações que ocorrem no processo completo de ordenação somando os números de comparações que ocorrem em cada nó da árvore.

Vamos inicialmente determinar o número de comparações feitas em cada nível da árvore. Observe que cada nó que aparece em qualquer nível da árvore possui a tarefa de ordenar um único segmento da lista original. Isso é feito pelo processo de intercalação, portanto requer um número de comparações igual ao número de elementos do segmento, como já demonstramos. Por isso, cada nível da árvore requer um número de comparações igual ao número total de elementos do segmento, e como os segmentos em um mesmo nível da árvore representam partes disjuntas da lista original, esse total não é maior do que o comprimento da lista original. Conseqüentemente, cada nível da árvore envolve não mais que $n$ comparações. (Obviamente, o nível mais baixo envolve a ordenação de listas com um único elemento, que não necessita comparação alguma.)

Agora vamos determinar o número de níveis na árvore. Para isso, observe que o processo de dividir problemas em problemas menores continua até que as listas de comprimento menor que dois sejam obtidas. Assim, o número de níveis na árvore é determinado pelo número de vezes que, iniciado com o valor $n$, podemos repetidamente dividir por dois até que o resultado não seja maior que um, que é $\lg n$. Mais precisamente, existem não mais do que $\lceil \lg n \rceil$ níveis na árvore que envolvam comparações, onde $\lceil \lg n \rceil$ representa o valor de $\lg n$ arredondado para cima até o próximo inteiro.

Finalmente, o número total de comparações feitas pelo algoritmo de ordenação por intercalação quando ordena uma lista de comprimento $n$ é obtido multiplicando-se o número de comparações feitas em cada nível da árvore pelo número de níveis nos quais as comparações são feitas. Concluímos que ele não será maior que $n \lceil \lg n \rceil$. Uma vez que o gráfico de $n \lceil \lg n \rceil$. possui a mesma forma geral do gráfico de $n \lg n$, concluímos que o algoritmo de ordenação por intercalação pertence à classe $O(n \lg n)$. Combinando isto com o fato de os pesquisadores garantirem que o problema da ordenação possui complexidade $\Theta(n \lg n)$, podemos afirmar que o algoritmo de ordenação por intercalação representa uma solução ótima para o problema da ordenação.

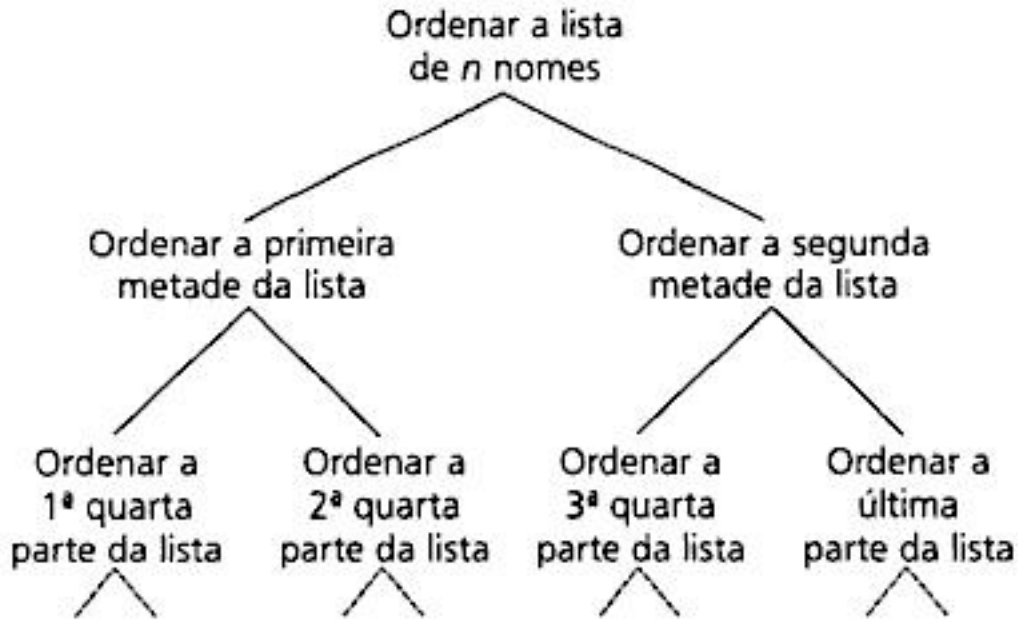



**Figura 11.10** A hierarquia dos problemas gerados pelo algoritmo de ordenação por intercalação.

## Problemas polinomiais *versus* não-polinomiais

Suponha que $f(n)$ e $g(n)$ sejam expressões matemáticas. Dizer que $g(n)$ é limitada por $f(n)$ significa que, conforme aplicamos essas expres-

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

sistema modular, fenômeno que não ocorre no sistema tradicional de números inteiros. Por exemplo, no sistema modular baseado no módulo 7, temos $3 \times 5 = 1$ (uma vez que $3 \times 5 = 15$ e $15 \div 7$ produz resto 1).

Dois números que resultam no produto 1 são chamados inversos multiplicativos um do outro. No sistema tradicional, o número 3 não possui inverso multiplicativo. Em vez disso, o seu inverso multiplicativo tradicional, que é 1/3, está fora do sistema de valores inteiros. No entanto, no sistema de inteiros módulo 7, o valor 3 possui um inverso multiplicativo que, como já vimos, é 5.

Quando um valor $x$ tem um inverso multiplicativo no sistema modular baseado no módulo $m$? Os matemáticos nos dizem que se $x$ e $m$ são dois inteiros positivos tais que $x < m$ e $x$ e $m$ são relativamente primos (ou seja, o único inteiro que divide ambos exatamente é 1), então o valor $x$ terá um inverso multiplicativo no sistema modular baseado no módulo $m$. Por exemplo, 6 é menor do que 13, e os dois valores não possuem divisor comum além do 1. Assim, o 6 terá um inverso multiplicativo no sistema de módulo 13. De fato, seu inverso é 11, pois $6 \times 11 = 66$, que quando dividido por 13 produz resto 1, isto é, $6 \times 11 \equiv 1 \pmod{13}$.

O fato de alguns inteiros terem outros inteiros como seus inversos multiplicativos em um sistema modular pode parecer um fenômeno estranho à primeira vista. Por exemplo, uma vez que 6 e 11 são inversos multiplicativos em um sistema módulo 13, vemos que $6 \times 11 \times x = x$, para qualquer valor de $x$. Afinal de contas, $6 \times 11 \times x = (6 \times 11) \times x = 1 \times x = x$.

## De volta à criptografia

Note que se o valor $x$ é não-negativo e menor que o módulo $m$, então $x \pmod{m}$ é o próprio $x$. Isso significa que se realizarmos operações aritméticas cujos resultados tradicionais estejam na faixa de 0 até $m - 1$, então esses resultados serão os mesmos obtidos no sistema modular. Assim, se tomarmos um módulo extremamente grande, poderemos realizar nossas computações aritméticas usuais sem saber sequer se estamos trabalhando no sistema aritmético tradicional ou em um sistema modular. Por exemplo, uma vez que a soma de todos os valores na lista

1 4 6 12 25 51 105 210 421 850

é 1685, as adições realizadas quando se tenta resolver um problema da mochila baseado nessa lista jamais produzirão resultados maiores que 1685. Quando resolvemos esses problemas, não precisamos nos preocupar se estamos trabalhando no sistema aritmético tradicional ou em um sistema modular cujo módulo é maior que 1685.

Entretanto, se pretendemos que o nosso problema da mochila fácil seja posto em um sistema modular com grande módulo, podemos obter um método de convertê-lo em um problema mais difícil e trazê-lo de volta novamente. Para esclarecer, suponha que tenhamos uma lista de números

$$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \quad a_9 \quad a_{10}$$

tal que cada elemento na lista seja maior que a soma de seus antecedentes. Ela é, portanto, uma lista em cujos termos os problemas da mochila podem ser facilmente resolvidos. Tomemos um módulo $m$ que seja maior que a soma de todos os valores na lista e dois outros valores $x$ e $y$ que sejam inversos multiplicativos no sistema modular baseado em $m$.

Se multiplicarmos cada elemento na lista original por $x$, obteremos a lista

$$a_1x \quad a_2x \quad a_3x \quad a_4x \quad a_5x \quad a_6x \quad a_7x \quad a_8x \quad a_9x \quad a_{10}x$$

em cujos termos os problemas da mochila novamente são resolvidos com facilidade. (Cada elemento continua sendo maior que a soma de seus antecedentes.) Entretanto, agora vamos substituir cada elemento em nossa nova lista por um valor equivalente a ele mod $m$. Especificamente, no lugar de $a_1x$, colocaremos o valor $a_1x \pmod{m}$; no lugar de $a_2x$, o valor $a_2x \pmod{m}$ e assim por diante. Isso produzirá uma nova lista

$$b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7 \quad b_8 \quad b_9 \quad b_{10}$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# APÊNDICES

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

| Código de operação | Operando | Descrição |
|---|---|---|
| | | *Exemplo*: 95F3 coloca no registrador 5 o resultado da operação EXCLUSIVE OR entre os conteúdos dos registradores F e 3. |
| A | R0X | ROTATE (*gira*) o padrão de *bits* do registrador R, de X *bits* para a direita. Sempre coloca o *bit* que está na extremidade de mais baixa ordem na de mais alta ordem. |
| | | *Exemplo*: A403 gira em 3 bits para a direita o conteúdo do registrador 4, de forma circular. |
| B | RXY | JUMP (*salta*) para a instrução localizada na posição de memória de endereço XY se o padrão de *bits* do registrador R coincidir com o padrão de *bits* do registrador 0. Caso contrário, prossegue na seqüência normal de execução. (O salto é implementado copiando o valor de XY no contador de instruções durante a fase de execução.) |
| | | *Exemplo*: B43C primeiro compara o conteúdo do registrador 4 com o do registrador 0. Se os dois forem iguais, o padrão 3C será colocado no contador de instruções, de modo que a próxima instrução a ser executada será a localizada naquele endereço de memória. Caso contrário, a execução do programa continuará em sua seqüência normal. |
| C | 000 | HALT (*pára*) a execução. |
| | | *Exemplo*: C000 pára a execução do programa. |

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
// Programa para manipular uma lista

// Todos os objetos do tipo lista contêm uma lista de nomes e
// três métodos públicos chamados ObtemNomes, OrdenaLista e
// ImprimeNomes.

class Lista {
  const int ComprimentoDaLista = 10;
  private String[] Nomes;

  List() {
    Nomes = new String[ComprimentoDaLista];
  }

  public void ObtemNomes() {
    for (int i = 0; i < ComprimentoDaLista; i++)
      Nomes[i] = System.Console.ReadLine();
  }

  public void OrdenaLista() {
    int j:
    String Pivot;
    for (int i = 1; i < ComprimentoDaLista; i++) {
      Pivot = Nomes[i];
      j = i - 1;
      while ((j>= o) && String.Compare(Pivot, Nomes [j], true) < 0)) {
        Nomes[j+1] = Nomes[j];
        j--;
      }
    }
    Nomes[j+1] = Pivot;
  }

  public void ImprimeNomes() {
    for (int i = 0; i < ComprimentoDaLista; i++)
      System.Console.WriteLine(Nomes[i]);
  }
}

// Definir um objeto chamado ListaDeNomes e solicitar-lhe que
// colete alguns nomes, os ordene e imprima essa lista.

class Ordena {
  public static void main() {
    Lista ListaDeNomes = new Lista();
    ListaDeNomes.ObtemNomes();
    ListaDeNomes.OrdenaLista();
    ListaDeNomes.ImprimeNomes();
    Return 0;
}
```

**FIGURA D.4** Exemplo de programa escrito na linguagem C#.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(onde *nome* representa qualquer nome de variável e *unidade*, qualquer nome de unidade do programa, diferente de MAIN). Além disso, permitimos que as unidades diferentes de MAIN chamem a si próprias, recursivamente.

Com esses recursos adicionais, podemos simular a estrutura while-end, encontrada na Bare Bones original. Por exemplo, um programa escrito em Bare Bones da forma

```
while X not 0 do;
  S;
end;
```

(em que S representa qualquer seqüência de instruções Bare Bones) pode ser substituído pela estrutura de unidades

```
      MAIN: begin;
              if X not 0 perform unidadeA;
            end;
unidadeA: begin;
              S;
              if X not 0 perform unidadeA;
            return;
```

Como conseqüência, podemos concluir que a linguagem modificada preserva todas as capacidades da linguagem Bare Bones original.

Também é possível demonstrar que qualquer problema que admita solução com o uso da linguagem modificada também pode ser resolvido usando Bare Bones. Um método para provar isso é mostrar como qualquer algoritmo expresso na linguagem modificada pode ser escrito em Bare Bones original. Isso, porém, envolve uma descrição explícita de como as estruturas recursivas podem ser simuladas com a estrutura while-end de Bare Bones.

Para o nosso propósito, é mais simples confiar na tese de Church-Turing, apresentada no Capítulo 11. Essa tese e o fato de Bare Bones apresentar a mesma capacidade de uma máquina de Turing confirmam que nenhuma linguagem pode ser mais capaz do que a nossa Bare Bones original. Assim, concluímos, de imediato, que qualquer problema solúvel com a nossa linguagem modificada também pode ser resolvido utilizando Bare Bones.

Concluímos que o poder da linguagem modificada é igual ao da Bare Bones original. A única diferença entre ambas é que uma apresenta estrutura de controle iterativa e a outra fornece recursão. Devemos concluir, então, que as duas estruturas de controle são de fato equivalentes em termos de poder computacional.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

**5.** a. 0111 (5 + 2 = 7) b. 0100 (3 + 1 = 4) c. 1111 (5+ (–6) = –1)
d. 0001 (–2 + 3 = 1) e. 1000 (–6 + (–2) = –8)

**6.** a. 0111 b. 1011 (estouro) c. 0100 (estouro)
d. 0001 e. 1000 (estouro)

**7.**

| a. | b. | c. | d. | e. |
|---|---|---|---|---|
| 0110 | 0011 | 0100 | 0010 | 0001 |
| + 0001 | + 1110 | + 1010 | + 0100 | + 1011 |
| 0111 | 0001 | 1110 | 0110 | 1100 |

**8.** Não. O estouro ocorre quando é feita uma tentativa de armazenar um número muito grande para o sistema em questão. Ao somar um valor positivo com um negativo, o valor do resultado deve estar compreendido entre esses dois valores. Assim, como os valores originais são pequenos o suficiente para ser representados, o resultado também será.

**9.** a. 6 dado que 1110 → 14 – 8
b. –1 dado que 0111 → 7 – 8
c. 0 dado que 1000 → 8 – 8
d. –6 dado que 0010 → 2 – 8
e. –8 dado que 0000 → 0 – 8
f. 1 dado que 1001 → 9 – 8

**10.** a. 1101 dado que 5 + 8 = 13 → 1101
b. 0011 dado que 25 + 8 = 3 → 0011
c. 1011 dado que 3 + 8 = 11 → 1011
d. 1000 dado que 0 + 8 = 8 → 1000
e. 1111 dado que 7 + 8 = 15 → 1111
f. 0000 dado que 28 + 8 = 0 → 0000

**11.** Não. O maior valor que pode ser armazenado na notação de excesso de oito é 7, representado por 1111. Para representar um valor maior, deve ser utilizado, no mínimo, excesso de 16 (que usa padrões de 5 *bits*). Do mesmo modo, 6 não pode ser representado na notação de excesso de quatro. (O maior valor que pode sê-lo é 3.)

## *Seção 1.7*

**1.** a. $^{5}/_{8}$ b. $3^{1}/_{4}$ c. $^{9}/_{32}$
d. $-1^{1}/_{2}$ e. $-^{11}/_{64}$

**2.** a. 01101011 b. 01111010 (erro de truncamento)
c. 01001100 d. 11101110
e. 11111000 (erro de truncamento)

**3.** 01001001 ($^{9}/_{16}$) é maior que 00111101 ($^{13}/_{32}$). Segue abaixo um método simples de determinar qual dos dois padrões representa o maior valor:

*Caso 1.* Se os *bits* de sinal forem diferentes, o maior é aquele com *bit* de sinal igual a 0.

*Caso 2.* Se os dois *bits* de sinal forem 0, varrer o restante dos padrões, da esquerda para a direita, até que seja encontrada uma posição de *bit* em que os dois padrões diferem. O padrão que contiver um 1 nesta posição representa o maior valor.

*Caso 3.* Se os dois *bits* de sinal forem 1, varrer o restante dos padrões, da esquerda para a direita, até que seja encontrada uma posição de *bit* em que os dois padrões diferem. O padrão que contiver um 0 nesta posição representa o maior valor.

A simplicidade deste processo de comparação é uma das razões da utilização da notação de excesso para representar o expoente em sistemas de vírgula flutuante, no lugar da notação de complemento de dois.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

2. O processador completa o seu ciclo corrente, armazena o estado do processo atual e ajusta o seu contador de instruções com um valor predeterminado (que é a localização do programa de tratamento de interrupções). Assim, a instrução a ser executada em seguida será a primeira instrução do programa de tratamento de interrupções.

3. Podem ser-lhes atribuídas prioridades maiores, de forma que recebam preferência da parte do despachante. Outra opção seria dar ao processo de maior prioridade fatias de tempo mais longas.

4. Se cada processo consumir a sua fatia de tempo completa, a máquina proverá um quantum de tempo para no máximo 20 processos em um segundo. Se os processos não consumirem suas fatias completas, esse valor pode ser bem maior, mas então o tempo necessário para realizar o chaveamento de contexto talvez se torne mais significativo. (Veja o Problema 5.)

5. Na verdade, um total de 5000/5001 do tempo de máquina seria gasto executando processos. Contudo, quando um processo solicita uma atividade de entrada/saída, a sua fatia de tempo é terminada, enquanto o controlador executa a solicitação. Assim, se cada processo fizesse tal solicitação depois de apenas 1 microssegundo do início de seu quantum, a eficiência da máquina cairia para ½. Assim, a máquina gastaria a mesma parcela de tempo fazendo chaveamentos de contexto e executando processos.

6. Que tal um negócio de vendas por correspondência e seus clientes, um corretor de títulos e seus clientes, ou um farmacêutico e seus clientes?

### *Seção 3.4*

1. Este sistema garante que o recurso não seja utilizado por mais de um processo de cada vez, porém determina que o recurso seja alocado apenas de forma alternada entre os processos. Uma vez que um processo tiver utilizado e devolvido o recurso, deverá esperar que o outro termine de utilizá-lo, antes que o original possa acessá-lo novamente. Isto é verdadeiro mesmo se o primeiro processo necessitar do recurso imediatamente, ainda que o outro não precise dele por algum tempo.

2. Se dois carros entrarem pelas extremidades opostas do túnel ao mesmo tempo, um não estará ciente da presença do outro. O processo de entrar e acender as luzes é um outro exemplo de uma região crítica. Neste caso, poderíamos chamá-lo de processo crítico. Nesta terminologia, resumiríamos a falha dizendo que os carros em extremidades opostas do túnel executariam um processo crítico ao mesmo tempo.

3. a. Isto garante que o recurso não-compartilhável não seja solicitado nem alocado parcialmente, ou seja, cede-se ao carro toda a ponte ou nada.
   b. Isto significa que o recurso não-compartilhável pode ser recuperado à força.
   c. Isto transforma o recurso não-compartilhável em compartilhável, o que elimina a competição.

4. Uma seqüência de setas que formam um caminho fechado no grafo orientado. Foi com base nesta observação que se desenvolveram técnicas para que alguns sistemas operacionais conseguissem reconhecer a existência de enlace mortal e, por conseguinte, ativar a ação corretiva apropriada.

### *Seção 3.5*

1. Uma rede aberta é aquela cujas especificações e protocolos são públicos, permitindo que diferentes vendedores possam produzir produtos compatíveis.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4. As linguagens de terceira geração permitem que o programa seja expresso mais nos termos do ambiente do problema e menos nos das ininteligibilidades do computador, como faziam as linguagens das primeiras gerações.

### *Seção 5.2*

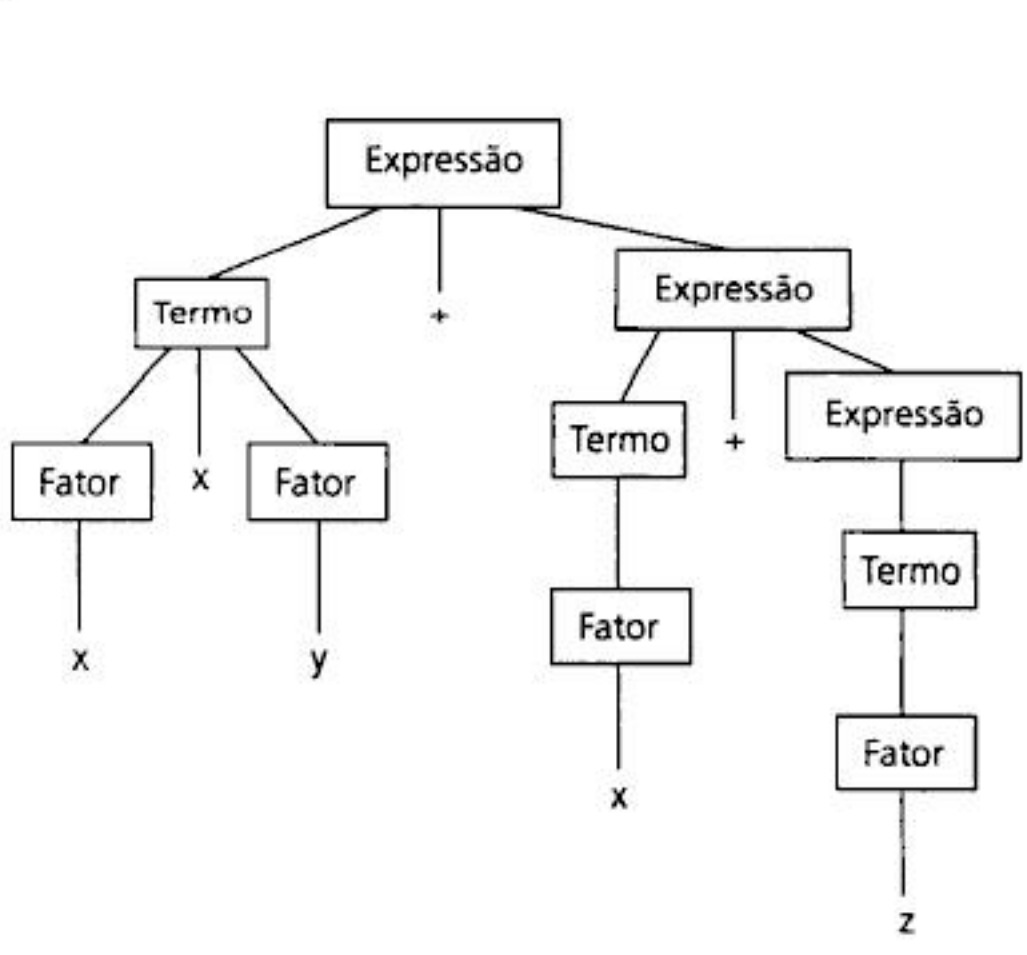
1. Usar constantes descritivas pode tornar o programa mais acessível.
2. Uma instrução declarativa descreve uma terminologia, enquanto uma imperativa descreve os passos de um algoritmo.
3. Inteiro, real, caractere e booleano.
4. As estruturas *if-then-else* e *while* são muito comuns.
5. Todos os elementos de uma matriz homogênea têm o mesmo tipo.

### *Seção 5.3*

1. Uma variável local só é acessível dentro de uma unidade de programa, como um procedimento, enquanto uma variável global for acessível por toda a extensão do programa.
2. Uma função é um procedimento que retorna um valor associado ao nome da função.
3. Porque isto é o que elas são. Operações de entrada ou saída são, na realidade, chamadas das rotinas do sistema operacional da máquina.
4. Um parâmetro formal é um identificador interno de um procedimento. Serve como reserva de área para o valor (argumento) que é passado ao procedimento quando este for ativado.
5. Um procedimento é projetado para realizar uma ação, enquanto uma função, para produzir um valor. Assim, o programa ficará mais legível se o nome do procedimento refletir a ação que ele realiza, e o nome da função refletir o valor que ela retorna.

### *Seção 5.4*

1. *Análise lexical:* o processo de identificar símbolos.
   *Análise sintática:* o processo de reconhecer a estrutura gramatical do programa.
   *Geração de código:* o processo de produzir as instruções do programa-objeto.
2. Uma tabela de símbolos é o registro da informação extraída das instruções declarativas do programa pelo analisador sintático.
3. Veja figura.




4. Eles são uma ou mais instâncias das subcadeias
   para a frente, para trás, cha, cha, cha,
   para trás, para a frente, cha, cha, cha,
   balançar para a direita, cha, cha, cha,
   balançar para a esquerda, cha, cha, cha,

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# Parte III

## Capítulo 7

### *Seção 7.1*

1. Se você fosse escrever um programa para jogar damas, a estrutura de dados para representar o tabuleiro provavelmente seria estática, uma vez que o seu tamanho não muda durante o jogo. Entretanto, se fosse escrever um programa para jogar dominó, a estrutura de dados que representa o padrão de dominós construído sobre a mesa provavelmente seria dinâmica, uma vez que esse padrão varia em tamanho e não pode ser predeterminado.
2. Um catálogo telefônico é essencialmente uma coleção de ponteiros para pessoas. Os vestígios encontrados na cena do crime são ponteiros para o criminoso (talvez codificados).

### *Seção 7.2*

1. 5 3 7 4 2 8 1 9 6
2. Se R for o número de linhas da matriz, a fórmula será R(J – 1)+(I – 1).
3. A partir do endereço inicial 25, devemos saltar 11(3 – 1)+(6 – 1) = 27 elementos da matriz, sendo que cada um ocupa duas células de memória. Assim, devemos saltar 54 células de memória. Portanto, o endereço final pode ser encontrado somando-se 54 ao endereço do primeiro elemento, o que nos fornecerá o endereço 79.
4. (C – I)+J

### *Seção 7.3*

1. Como exemplo, para encontrar o quinto elemento de uma lista densa, multiplique o número de células de cada elemento por 4 e some o resultado ao endereço do primeiro elemento. A situação é bem diferente no caso da lista ligada, pois o endereço do quinto elemento não tem relação alguma com o do primeiro. Assim, para encontrar o quinto elemento, deve-se, de fato, percorrer cada um dos elementos que o precedem.
2. O ponteiro para o início da lista contém o valor NIL.
3. Ultimo ← Último nome a ser impresso
   Terminado ← Falso
   Ponteiro Corrente ← ponteiro para o início da lista;
   enquanto (Ponteiro Corrente não for NIL e Terminado = falso) faça
    (imprimir o elemento apontado pelo Ponteiro Corrente,
     se (o nome que acabou de ser impresso = Ultimo)
        então (Terminado ← verdadeiro)
     Ponteiro Corrente ← o valor contido na célula de ponteiro do
         elemento apontado pelo Ponteiro Corrente)
4. procedimento apaga (Elemento)
   Corrente ← InícioDaLista
   Anterior ← NIL
   Encontrado ← falso
   enquanto (Corrente não é NIL e Encontrado = falso) faça
    se (elemento apontado por Corrente for o elemento desejado)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

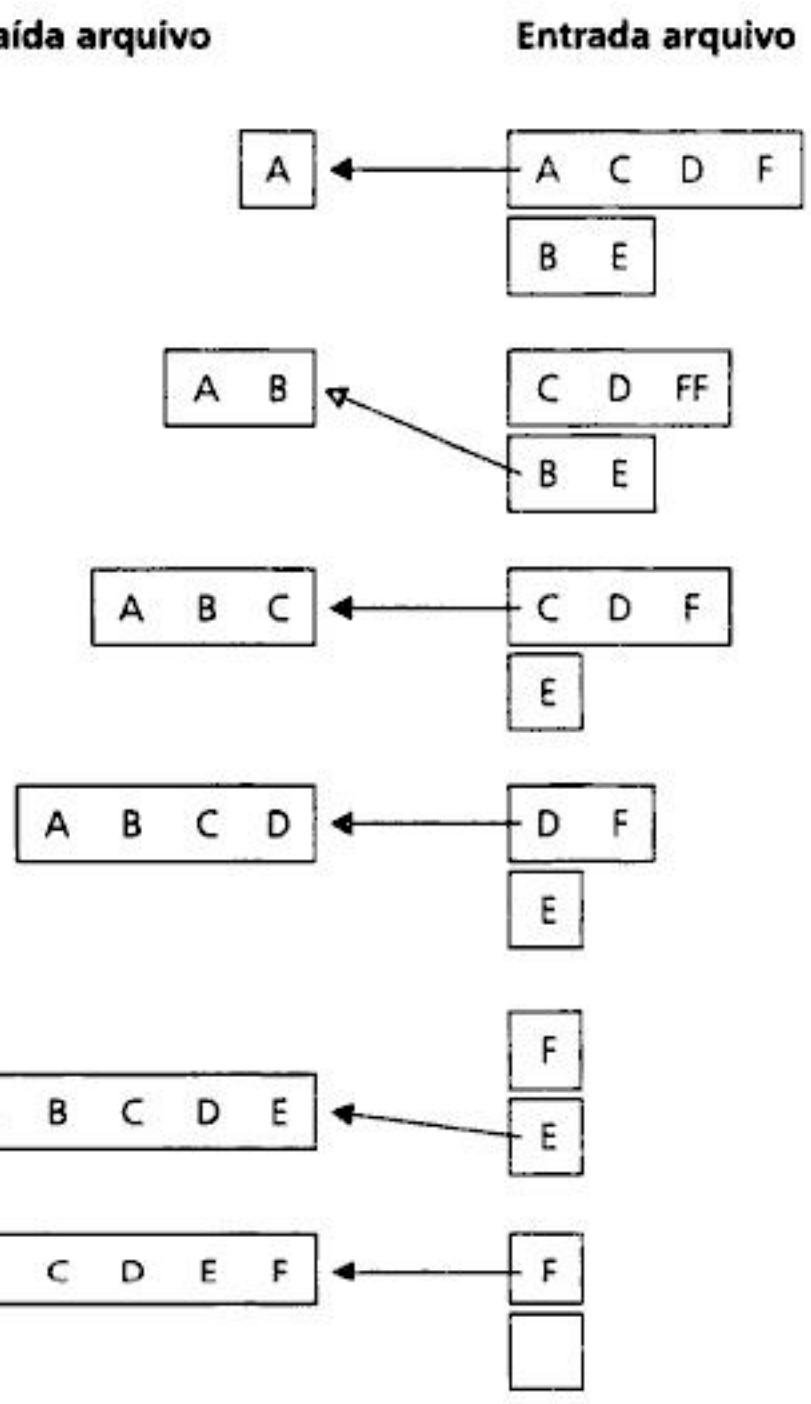
## Capítulo 8

### *Seção 8.1*

1. Os registros físicos são lidos do armazenamento em massa, em uma área de retenção de onde o programa de aplicação tem acesso aos dados em termos de registros lógicos.
2. Um descritor de arquivo é uma tabela que mantém a informação necessária a um sistema operacional para manipular o arquivo.
3. O gerente de arquivos é que constrói o descritor.

### *Seção 8.2*

1. Você deveria passar pelos seguintes estágios:




2. A idéia é primeiramente dividir o arquivo a ser ordenado em vários arquivos separados, sendo que cada qual contém um registro. A seguir, agrupar aos pares os arquivos com um registro e, para cada par, aplicar o algoritmo de intercalação. Isto diminuiria para a metade o número de arquivos, cada qual com dois registros. Além disso, cada arquivo de dois registros está ordenado. Podemos agrupá-los em pares e novamente aplicar o algoritmo de intercalação para os pares. Mais uma vez, teremos menos arquivos, porém maiores e ordenados. Continuando desta forma, teremos, em última instância, apenas um arquivo com todos os registros originais ordenados. (Se ocorrer um número ímpar de arquivos em alguma fase deste processo,

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

### *Seção 10.5*

1. 00101001100000100001000000
2. Que tal o problema de desenvolver uma estratégia para investir no mercado de ações?
3. A estrutura de um programa baseado no paradigma funcional é de funções dentro de funções. Isto é, a estrutura é homogênea em todos os níveis. Contudo, um programa baseado no paradigma orientado a objeto consiste em objetos que podem conter métodos, bem como outros objetos. Assim, a estrutura não é tão homogênea quanto a do paradigma funcional. Por isso, é mais complicado misturar componentes de programas orientados a objeto para formar novos programas.

### *Seção 10.6*

1. A oração está descrevendo o tipo de cavalo ou está contando o que algumas pessoas estão fazendo?
2. O processo de análise sintática produz estruturas idênticas, mas a análise semântica reconhece que a frase preposicionada da primeira oração conta onde a cerca foi construída, enquanto a frase da segunda oração conta quando a cerca foi construída.
3. Eles são irmãos.
4. Ele usa a suposição de universo fechado.
5. Os dois são iguais em muitos aspectos. Contudo, os bancos de dados tradicionais tendem a conter somente fatos como o nome do funcionário, endereço e assim por diante, enquanto bases de conhecimento tendem a incluir regras como "se estiver chovendo, confira o pluviômetro", que podem ser usadas para orientar o processo de raciocínio.

### *Seção 10.7*

1. Não existe resposta correta ou errada.
2. Não existe resposta correta ou errada.
3. Não existe resposta correta ou errada.

## Capítulo 11

### *Seção 11.1*

1. O cálculo da dívida referente a um empréstimo, a área de um círculo ou a quilometragem de um carro.
2. Os matemáticos as chamam de funções transcendentais. Como exemplos, citam-se as funções logarítmicas e trigonométricas. Tais exemplos particulares ainda podem ser computados, mas não por meios algébricos. Por exemplo, as funções trigonométricas podem ser calculadas desenhando-se o triângulo em questão, medindo-se os seus lados e só então usando-se a operação algébrica de divisão.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# ÍNDICE

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

# CIÊNCIA DA COMPUTAÇÃO, UMA VISÃO ABRANGENTE

## J. GLENN BROOKSHEAR

Nesta edição, J. Glenn Brookshear dá continuidade à abordagem utilizada nas edições anteriores de fazer um texto compreensível, acessível e atualizado sobre a dinâmica área da *Ciência da Computação*. A obra progride do concreto para o abstrato, uma ordenação que resulta em uma apresentação pedagógica consistente, na qual cada tópico leva ao próximo.

Destaques desta edição:

- atualizações para refletir as últimas tendências da área, incluindo segurança de rede, desenvolvimento de código aberto, memória associativa, criptografia de chave pública, XML, Java e C#
- maior cobertura de redes e Internet
- o uso da linguagem C#, bem como C, C++ e Java, como exemplo de linguagem; a abordagem permanece independente da linguagem utilizada
- inclusões de textos independentes sobre pontos ou questões para maior exploração
- aproximadamente 1000 problemas e exercícios
- discussões sobre ética e aspectos legais que os cientistas da computação devem se preocupar, incluindo segurança na Internet, propriedade de software e responsabilidade, e implicações sobre sistemas de banco de dados

ISBN 978-85-363-0438-0
9 788536 304380
artmed®
EDITORA
RESPEITO PELO CONHECIMENTO
Bookman®
www.bookman.com.br