Capítulo 1

Introdução

Um computador digital é uma máquina que pode resolver problemas para as pessoas, executando instruções que lhe são dadas. Uma sequência de instruções descrevendo como realizar determinada tarefa é chamada de **programa**. Os circuitos eletrônicos de cada computador podem reconhecer e executar diretamente um conjunto limitado de instruções simples, para o qual todos os programas devem ser convertidos antes que possam ser executados. Essas instruções básicas raramente são muito mais complicadas do que

- Some dois números.
- Verifique se um número é zero.
- Copie dados de uma parte da memória do computador para outra.

Juntas, as instruções primitivas de um computador formam uma linguagem com a qual as pessoas podem se comunicar com ele. Essa linguagem é denominada **linguagem de máquina**. Quem projeta um novo computador deve decidir quais instruções incluir em sua linguagem de máquina. De modo geral, os projetistas tentam tornar as instruções primitivas as mais simples possíveis, coerentes com os requisitos de utilização e desempenho idealizados para o computador e seus requisitos de desempenho, a fim de reduzir a complexidade e o custo dos circuitos eletrônicos necessários. Como a maioria das linguagens de máquina é muito simples, sua utilização direta pelas pessoas é difícil e tediosa.

Com o passar do tempo, essa observação simples tem levado a uma forma de estruturar os computadores como uma sequência de abstrações, cada uma baseada naquela abaixo dela. Desse modo, a complexidade pode ser dominada e os sistemas de computação podem ser projetados de forma sistemática e organizada. Denominamos essa abordagem **organização estruturada de computadores** – foi esse o nome dado a este livro. Na seção seguinte, descreveremos o que significa esse termo. Logo após, comentaremos alguns desenvolvimentos históricos, o estado atual da tecnologia e exemplos importantes.

1.1 Organização estruturada de computadores

Como já mencionamos, existe uma grande lacuna entre o que é conveniente para as pessoas e o que é conveniente para computadores. As pessoas querem fazer *X*, mas os computadores só podem fazer *Y*, o que dá origem a um problema. O objetivo deste livro é explicar como esse problema pode ser resolvido.

1.1.1 Linguagens, níveis e máquinas virtuais

O problema pode ser abordado de duas maneiras, e ambas envolvem projetar um novo conjunto de instruções que é mais conveniente para as pessoas usarem do que o conjunto embutido de instruções de máquina. Juntas, essas novas instruções também formam uma linguagem, que chamaremos de L1, assim como as instruções de máquina embutidas formam uma linguagem, que chamaremos de L0. As duas técnicas diferem no modo como os programas escritos em L1 são executados pelo computador que, afinal, só pode executar programas escritos em sua linguagem de máquina, L0.

Um método de execução de um programa escrito em L1 é primeiro substituir cada instrução nele por uma sequência equivalente de instruções em L0. O programa resultante consiste totalmente em instruções L0. O computador, então, executa o novo programa L0 em vez do antigo programa L1. Essa técnica é chamada de **tradução**.

A outra técnica é escrever um programa em L0 que considere os programas em L1 como dados de entrada e os execute, examinando cada instrução por sua vez, executando diretamente a sequência equivalente de instruções L0. Essa técnica não requer que se gere um novo programa em L0. Ela é chamada de **interpretação**, e o programa que a executa é chamado de **interpretador**.

Tradução e interpretação são semelhantes. Nos dois métodos, o computador executa instruções em L1 executando sequências de instruções equivalentes em L0. A diferença é que, na tradução, o programa L1 inteiro primeiro é convertido para um L0, o programa L1 é desconsiderado e depois o novo L0 é carregado na memória do computador e executado. Durante a execução, o programa L0 recém-gerado está sendo executado e está no controle do computador.

Na interpretação, depois que cada instrução L1 é examinada e decodificada, ela é executada de imediato. Nenhum programa traduzido é gerado. Aqui, o interpretador está no controle do computador. Para ele, o programa L1 é apenas dados. Ambos os métodos e, cada vez mais, uma combinação dos dois, são bastante utilizados.

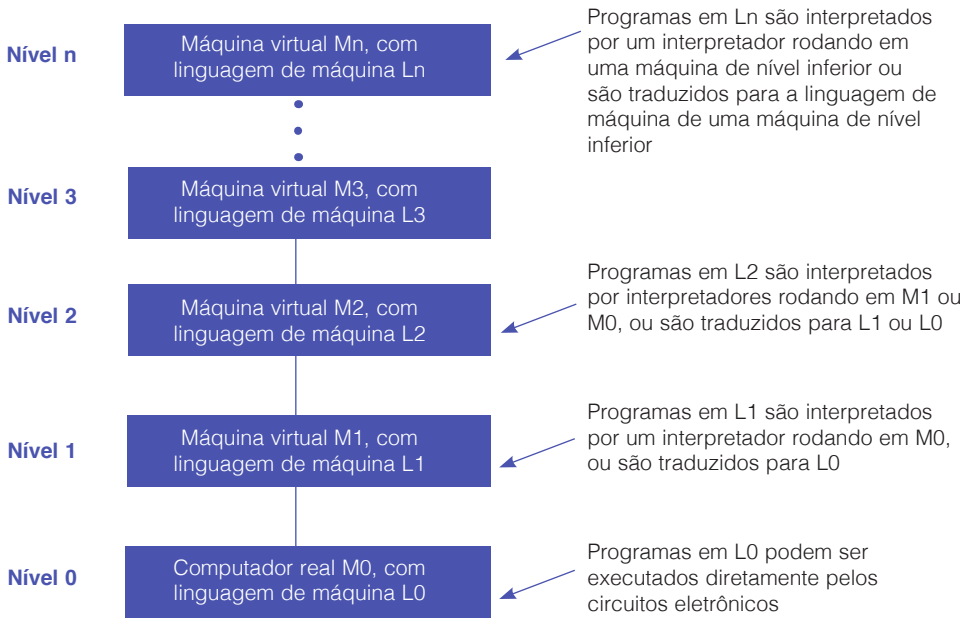
Em vez de pensar em termos de tradução ou interpretação, muitas vezes é mais simples imaginar a existência de um computador hipotético ou **máquina virtual** cuja linguagem seja L1. Vamos chamar essa máquina virtual de M1 (e de M0 aquela correspondente a L0). Se essa máquina pudesse ser construída de forma barata o suficiente, não seria preciso de forma alguma ter a linguagem L0 ou uma máquina que executou os programas em L0. As pessoas poderiam simplesmente escrever seus programas em L1 e fazer com que o computador os executasse diretamente. Mesmo que a máquina virtual cuja linguagem é L1 seja muito cara ou complicada de construir com circuitos eletrônicos, as pessoas ainda podem escrever programas para ela. Esses programas podem ser ou interpretados ou traduzidos por um programa escrito em L0 que, por si só, consegue ser executado diretamente pelo computador real. Em outras palavras, as pessoas podem escrever programas para máquinas virtuais, como se realmente existissem.

Para tornar prática a tradução ou a interpretação, as linguagens L0 e L1 não deverão ser "muito" diferentes. Tal restrição significa quase sempre que L1, embora melhor que L0, ainda estará longe do ideal para a maioria das aplicações. Esse resultado talvez seja desanimador à luz do propósito original da criação de L1 – aliviar o trabalho do programador de ter que expressar algoritmos em uma linguagem mais adequada a máquinas do que a pessoas. Porém, a situação não é desesperadora.

A abordagem óbvia é inventar outro conjunto de instruções que seja mais orientado a pessoas e menos orientado a máquinas que a L1. Esse terceiro conjunto também forma uma linguagem, que chamaremos de L2 (e com a máquina virtual M2). As pessoas podem escrever programas em L2 exatamente como se de fato existisse uma máquina real com linguagem de máquina L2. Esses programas podem ser traduzidos para L1 ou executados por um interpretador escrito em L1.

A invenção de toda uma série de linguagens, cada uma mais conveniente que suas antecessoras, pode prosseguir indefinidamente, até que, por fim, se chegue a uma adequada. Cada linguagem usa sua antecessora como base, portanto, podemos considerar um computador que use essa técnica como uma série de **camadas** ou **níveis**, um sobre o outro, conforme mostra a Figura 1.1. A linguagem ou nível mais embaixo é a mais simples, e a linguagem ou nível mais em cima é a mais sofisticada.

Figura 1.1 Máquina multinível.



Há uma relação importante entre uma linguagem e uma máquina virtual. Cada máquina tem uma linguagem de máquina, consistindo em todas as instruções que esta pode executar. Com efeito, uma máquina define uma linguagem. De modo semelhante, uma linguagem define uma máquina – a saber, aquela que pode executar todos os programas escritos na linguagem. Claro, pode ser muito complicado e caro construir a máquina definida por determinada linguagem diretamente pelos circuitos eletrônicos, mas, apesar disso, podemos imaginá-la. Uma máquina que tivesse C ou C++ ou Java como sua linguagem seria de fato complexa, mas poderia ser construída usando a tecnologia de hoje. Porém, há um bom motivo para não construir tal computador: ele não seria econômico em comparação com outras técnicas. O mero fato de ser factível não é bom o suficiente: um projeto prático também precisa ser econômico.

De certa forma, um computador com *n* níveis pode ser visto como *n* diferentes máquinas virtuais, cada uma com uma linguagem de máquina diferente. Usaremos os termos "nível" e "máquina virtual" para indicar a mesma coisa. Apenas programas escritos na linguagem L0 podem ser executados diretamente pelos circuitos eletrônicos, sem a necessidade de uma tradução ou interpretação intervenientes. Os programas escritos em L1, L2, ..., L*n* devem ser interpretados por um interpretador rodando em um nível mais baixo ou traduzidos para outra linguagem correspondente a um nível mais baixo.

Uma pessoa que escreve programas para a máquina virtual de nível *n* não precisa conhecer os interpretadores e tradutores subjacentes. A estrutura de máquina garante que esses programas, de alguma forma, serão executados. Não há interesse real em saber se eles são executados passo a passo por um interpretador que, por sua vez, também é executado por outro interpretador, ou se o são diretamente pelos circuitos eletrônicos. O mesmo resultado aparece nos dois casos: os programas são executados.

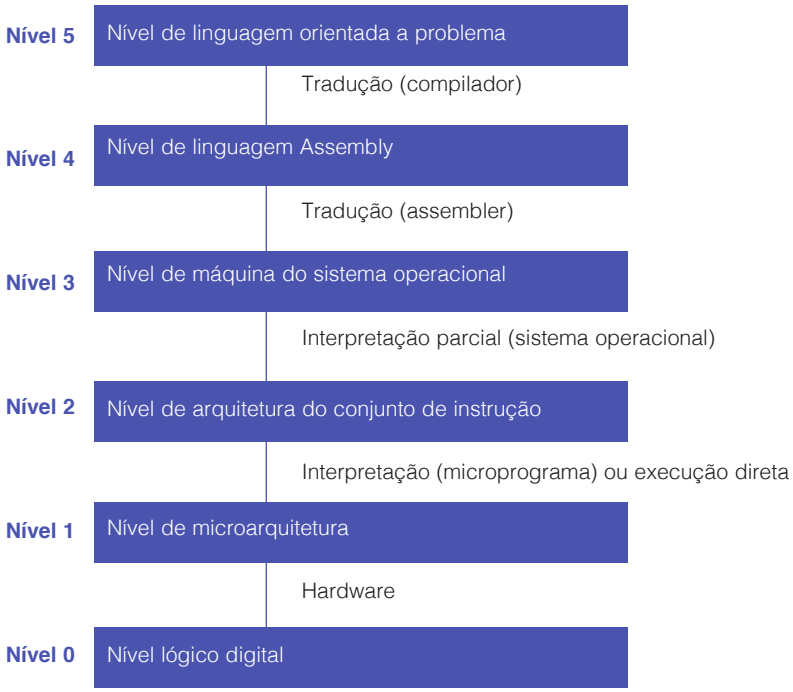
Quase todos os programadores que usam uma máquina de nível *n* estão interessados apenas no nível superior, aquele que menos se parece com a linguagem de máquina do nível mais inferior. Porém, as pessoas

interessadas em entender como um computador realmente funciona deverão estudar todos os níveis. Quem projeta novos computadores ou novos níveis também deve estar familiarizado com outros níveis além do mais alto. Os conceitos e técnicas de construção de máquinas como uma série de níveis e os detalhes dos próprios níveis formam o assunto principal deste livro.

1.1.2 Máquinas multiníveis contemporâneas

A maioria dos computadores modernos consiste de dois ou mais níveis. Existem máquinas com até seis níveis, conforme mostra a Figura 1.2. O nível 0, na parte inferior, é o hardware verdadeiro da máquina. Seus circuitos executam os programas em linguagem de máquina do nível 1. Por razões de precisão, temos que mencionar a existência de outro nível abaixo do nosso nível 0. Esse nível, que não aparece na Figura 1.2 por entrar no domínio da engenharia elétrica (e, portanto, estar fora do escopo deste livro), é chamado de **nível de dispositivo**. Nele, o projetista vê transistores individuais, que são os primitivos de mais baixo nível para projetistas de computador. Se alguém quiser saber como os transistores funcionam no interior, isso nos levará para o campo da física no estado sólido.

Figura 1.2 Um computador com seis níveis. O método de suporte para cada nível é indicado abaixo dele (junto com o nome do programa que o suporta).



No nível mais baixo que estudaremos, o **nível lógico digital**, os objetos interessantes são chamados de **portas** (ou ***gates***). Embora montadas a partir de componentes analógicos, como transistores, podem ser modeladas com precisão como dispositivos digitais. Cada porta tem uma ou mais entradas digitais (sinais representando 0 ou 1) e calcula como saída alguma função simples dessas entradas, como AND (E) ou OR (OU). Cada porta é composta de no máximo alguns transistores. Um pequeno número de portas podem ser combinadas para formar uma memória de 1 bit, que consegue armazenar um 0 ou um 1. As memórias de 1 bit podem ser combinadas em grupos de (por exemplo) 16, 32 ou 64 para formar registradores. Cada **registrador** pode manter um único número binário até algum máximo. As portas também podem ser combinadas para formar o próprio mecanismo de computação principal. Examinaremos as portas e o nível lógico digital com detalhes no Capítulo 3.

O próximo nível acima é o **nível de microarquitetura**. Aqui, vemos uma coleção de (em geral) 8 a 32 registradores que formam uma memória local e um circuito chamado **ULA – Unidade Lógica e Artitmética (em inglês Arithmetic Logic Unit**), que é capaz de realizar operações aritméticas simples. Os registradores estão conectados à ULA para formar um **caminho de dados**, sobre o qual estes fluem. A operação básica do caminho de dados consiste em selecionar um ou dois registradores, fazendo com que a ULA opere sobre eles (por exemplo, somando-os) e armazenando o resultado de volta para algum registrador.

Em algumas máquinas, a operação do caminho de dados é controlada por um programa chamado **microprograma**. Em outras, o caminho de dados é controlado diretamente pelo hardware. Nas três primeiras edições deste livro, chamamos esse nível de "nível de microprogramação", pois no passado ele quase sempre era um interpretador de software. Como o caminho de dados agora quase sempre é (em parte) controlado diretamente pelo hardware, mudamos o nome na quarta edição.

Em máquinas com controle do caminho de dados por software, o microprograma é um interpretador para as instruções no nível 2. Ele busca, examina e executa instruções uma por vez, usando o caminho de dados. Por exemplo, para uma instrução ADD, a instrução seria buscada, seus operandos localizados e trazidos para registradores, a soma calculada pela ULA e, por fim, o resultado retornado para o local a que pertence. Em uma máquina com controle do caminho de dados por hardware, haveria etapas semelhantes, mas sem um programa armazenado explícito para controlar a interpretação das instruções desse nível.

Chamaremos o nível 2 de **nível de arquitetura do conjunto de instrução,** ou nível **ISA** (**Instruction Set Architecture**). Os fabricantes publicam um manual para cada computador que vendem, intitulado "Manual de Referência da Linguagem de Máquina", ou "Princípios de Operação do Computador Western Wombat Modelo 100X", ou algo semelhante. Esses manuais, na realidade, referem-se ao nível ISA, e não aos subjacentes. Quando eles explicam o conjunto de instruções da máquina, na verdade estão descrevendo as instruções executadas de modo interpretativo pelo microprograma ou circuitos de execução do hardware. Se um fabricante oferecer dois interpretadores para uma de suas máquinas, interpretando dois níveis ISA diferentes, ele precisará oferecer dois manuais de referência da "linguagem de máquina", um para cada interpretador.

O próximo nível costuma ser híbrido. A maior parte das instruções em sua linguagem também está no nível ISA. (Não há motivo pelo qual uma instrução que aparece em um nível não possa estar presente também em outros.) Além disso, há um conjunto de novas instruções, uma organização de memória diferente, a capacidade de executar dois ou mais programas simultaneamente e diversos outros recursos. Existe mais variação entre os projetos de nível 3 do que entre aqueles no nível 1 ou no nível 2.

As novas facilidades acrescentadas no nível 3 são executadas por um interpretador rodando no nível 2, o qual, historicamente, tem sido chamado de sistema operacional. Aquelas instruções de nível 3 que são idênticas às do nível 2 são executadas direto pelo microprograma (ou controle do hardware), e não pelo sistema operacional. Em outras palavras, algumas das instruções de nível 3 são interpretadas pelo sistema operacional e algumas o são diretamente pelo microprograma. É a isso que chamamos de nível "híbrido". No decorrer deste livro, nós o chamaremos de **nível de máquina do sistema operacional**.

Há uma quebra fundamental entre os níveis 3 e 4. Os três níveis mais baixos não servem para uso do programador do tipo mais comum. Em vez disso, eles são voltados principalmente para a execução dos interpretadores e tradutores necessários para dar suporte aos níveis mais altos. Esses interpretadores e tradutores são escritos pelos **programadores de sistemas**, profissionais que se especializam no projeto e execução de novas máquinas virtuais. Os níveis 4 e acima são voltados para o programador de aplicações, que tem um problema para solucionar.

Outra mudança que ocorre no nível 4 é o método de suporte dos níveis mais altos. Os níveis 2 e 3 são sempre interpretados. Em geral, mas nem sempre, os níveis 4, 5 e acima são apoiados por tradução.

Outra diferença entre níveis 1, 2 e 3, por um lado, e 4, 5 e acima, por outro, é a natureza da linguagem fornecida. As linguagens de máquina dos níveis 1, 2 e 3 são numéricas. Os programas nessas linguagens consistem em uma longa série de números, muito boa para máquinas, mas ruim para as pessoas. A partir do nível 4, as linguagens contêm palavras e abreviações cujo significado as pessoas entendem.

O nível 4, o da linguagem de montagem (*assembly*), na realidade é uma forma simbólica para uma das linguagens subjacentes. Esse nível fornece um método para as pessoas escreverem programas para os níveis 1, 2 e 3 em uma forma que não seja tão desagradável quanto às linguagens de máquina virtual em si. Programas em linguagem de montagem são primeiro traduzidos para linguagem de nível 1, 2 ou 3, e em seguida interpretados pela máquina virtual ou real adequada. O programa que realiza a tradução é denominado **assembler**.

O nível 5 normalmente consiste em linguagens projetadas para ser usadas por programadores de aplicações que tenham um problema a resolver. Essas linguagens costumam ser denominadas **linguagens de alto nível**. Existem literalmente centenas delas. Algumas das mais conhecidas são C, C++, Java, Perl, Python e PHP. Programas escritos nessas linguagens em geral são traduzidos para nível 3 ou nível 4 por tradutores conhecidos como **compiladores**, embora às vezes sejam interpretados, em vez de traduzidos. Programas em Java, por exemplo, costumam ser primeiro traduzidos para uma linguagem semelhante à ISA denominada código de bytes Java, ou bytecode Java, que é então interpretada.

Em alguns casos, o nível 5 consiste em um interpretador para o domínio de uma aplicação específica, como matemática simbólica. Ele fornece dados e operações para resolver problemas nesse domínio em termos que pessoas versadas nele possam entendê-lo com facilidade.

Resumindo, o aspecto fundamental a lembrar é que computadores são projetados como uma série de níveis, cada um construído sobre seus antecessores. Cada nível representa uma abstração distinta na qual estão presentes diferentes objetos e operações. Projetando e analisando computadores desse modo, por enquanto podemos dispensar detalhes irrelevantes e assim reduzir um assunto complexo a algo mais fácil de entender.

O conjunto de tipos de dados, operações e características de cada nível é denominado **arquitetura**. Ela trata dos aspectos que são visíveis ao usuário daquele nível. Características que o programador vê, como a quantidade de memória disponível, são parte da arquitetura. Aspectos de implementação, como o tipo da tecnologia usada para executar a memória, não são parte da arquitetura. O estudo sobre como projetar as partes de um sistema de computador que sejam visíveis para os programadores é denominado **arquitetura de computadores**. Na prática, contudo, arquitetura de computadores e organização de computadores significam basicamente a mesma coisa.

1.1.3 Evolução de máquinas multiníveis

Para colocar as máquinas multiníveis em certa perspectiva, examinaremos rapidamente seu desenvolvimento histórico, mostrando como o número e a natureza dos níveis evoluíram com o passar dos anos. Programas escritos em uma verdadeira linguagem de máquina (nível 1) de um computador podem ser executados diretamente pelos circuitos eletrônicos (nível 0) do computador, sem qualquer interpretador ou tradutor interveniente. Esses circuitos eletrônicos, junto com a memória e dispositivos de entrada/saída, formam o **hardware** do computador. Este consiste em objetos tangíveis – circuitos integrados, placas de circuito impresso, cabos, fontes de alimentação, memórias e impressoras – em vez de ideias abstratas, algoritmos ou instruções.

Por outro lado, o **software** consiste em **algoritmos** (instruções detalhadas que dizem como fazer algo) e suas representações no computador – isto é, programas. Eles podem ser armazenados em disco rígido, CD-ROM, ou outros meios, mas a essência do software é o conjunto de instruções que compõe os programas, e não o meio físico no qual estão gravados.

Nos primeiros computadores, a fronteira entre hardware e software era nítida. Com o tempo, no entanto, essa fronteira ficou bastante indistinta, principalmente por causa da adição, remoção e fusão de níveis à medida que os computadores evoluíam. Hoje, muitas vezes é difícil distingui-la (Vahid, 2003). Na verdade, um tema central deste livro é

Hardware e software são logicamente equivalentes.

Qualquer operação executada por software também pode ser embutida diretamente no hardware, de preferência após ela ter sido suficientemente bem entendida. Como observou Karen Panetta: “Hardware é apenas software petrificado”. Claro que o contrário é verdadeiro: qualquer instrução executada em hardware também pode ser simulada em software. A decisão de colocar certas funções em hardware e outras em software é baseada em fatores

Tanenbaum • Austin

Organização estruturada de computadores

6ª EDIÇÃO

ALWAYS LEARNING

PEARSON