

# 计算机概要与技术

在不关注具体过程的情况下完成更多的操作，这种方法促进了文明的进步。

——Alfred North Whitehead, 《An Introduction to Mathematics》, 1911

## 1.1 引言

欢迎阅读本书！非常高兴有机会与大家一起分享令人兴奋的计算机系统世界。这是一个进步飞快、新思想层出不穷、非常有趣的领域。事实上，计算机是极度充满生气的信息技术工业的产物，其相关产品几乎占全美国民生产总值的 10%。在摩尔定律的推动下，美国的经济已经与信息技术密不可分。这个不寻常的工业具有惊人的发展速度。在过去 30 年里，出现了许多导致计算产业革命的新型计算机，但是它们很快就被更好的计算机所取代。

电子计算机自 20 世纪 40 年代后期诞生以来，其创新性的竞争带来了史无前例的进步。如果运输业能够以计算机工业的速度发展，那么我们只需要花一美分就可以在一秒钟之内从纽约赶到伦敦。想象一下，这样的进步将如何改变社会——生活在南太平洋的塔希提岛，而工作在旧金山，傍晚去莫斯科吃夜宵——你能够想象得出这种进步意味着什么。

沿着农业革命、工业革命的发展方向，计算机促进了人类的第三次革命——信息革命。信息革命使人类的能力成倍增长，自然而深刻地影响着人类的日常生活，甚至改变了寻求新知识的方法。现在有一种科学探索的新方式，即计算科学家联合理论和实验科学家，共同探索天文学、生物学、化学和物理学的前沿问题。

计算机革命一直在向前推进。每当计算成本降低 10 倍，计算机的发展机遇就会增加 10 倍。原本出于经济考虑不可行的应用突然变得可行了。例如，下述各项应用在过去曾经是“计算机科学幻想”：

- **车载计算机：**在 20 世纪 80 年代初微处理器的性能和价格得到极大改进之前，用计算机来控制汽车几乎是天方夜谭。而今天，用计算机控制的汽车发动机普遍应用，车载计算机不仅改进了燃油效率、减轻了污染，还通过盲点警告、车道偏离警告、移动目标检测和安全气囊实现了碰撞时对乘客的保护。
- **手机：**谁曾想到计算机系统的发展会使全球一半以上的人口拥有手机，并让人们几乎在全球的各个角落都可以自由通信？
- **人类基因项目：**以前用于匹配和分析人类基因序列的计算机设备价格达几亿美元。在过去的 15~25 年里，用于该项目的计算机设备的价格降低了 10~100 倍。随着计算机设备价格的持续下降，人们可以获得自己的基因序列，并利用其来治疗疾病。
- **万维网：**在编写本书第 1 版时，万维网尚不存在，而现在万维网已经改变了整个社会。在许多地方，它已取代了传统的图书馆和报纸。
- **搜索引擎：**随着万维网规模的扩大和价值的与日俱增，如何快速精确地找到所需信息变得越来越重要。今天，如果没有搜索引擎，许多人在万维网中将寸步难行。

显而易见，计算机技术的进步几乎影响着社会的每一个方面。硬件的进步使得程序员可以编写出各种优秀的应用软件，进而证实计算机几乎是无所不能的。今天的科学幻想在未来就会成为现实，诸如虚拟现实、无现金社会和无人驾驶汽车等。

### 1.1.1 计算应用的分类及其特性

从智能家电到手机再到最大型超级计算机，它们虽然使用了一套通用的硬件技术（参见1.4和1.5节），但这些不同的应用有着不同的设计需求，并以不同的方式通过硬件实现。概括地说，计算机主要包括以下三类应用：

**个人计算机**（Personal Computer, PC）也许是最为人所知的应用方式，本书的读者几乎都在大量使用。个人计算机强调对单用户提供良好的性能，价格低廉，通常运行第三方软件。尽管此类应用的出现只有短短的35年，但它推动了许多计算技术的革新。

- ➊ 个人计算机：用于个人使用的计算机，通常包含图形显示器、键盘和鼠标等。

**服务器**（server）是过去被称为大型机的现代形式，通常借助网络访问。服务器适用于执行大负载任务，可以执行单个复杂应用（科学的或工程的），也可以处理大量的简单作业，如大型Web服务器。这些应用通常基于其他来源的软件（例如数据库或仿真软件），并且往往为了特别的需要而加以修改或定制。服务器的制造技术和桌面计算机差不多，但能够提供更强的计算、存储和I/O能力。通常情况下，当发生故障时，服务器比个人计算机恢复的代价高得多，因此服务器更加强调可靠性。

- ➋ 服务器：用于为多用户运行大型程序的计算机，通常由多个用户并行使用，并且一般通过网络访问。

服务器的功能和价格有很大的伸缩范围。低端服务器可能比桌面计算机稍微贵些，不带显示器和键盘的大约需要1000美元，一般用于文档存储、小型商务应用或者简单的Web服务（见6.10节）。高端服务器称为**超级计算机**（supercomputer），一般由成百上千台处理器组成，内存为**terabyte**级，其价格可高达数千万甚至上亿美元。它们主要用于高端科学和工程计算，如天气预报、石油勘探、蛋白质结构计算和其他大规模问题。虽然这类超级计算机代表了最高的计算能力，但是它们只占服务器相对很小的一部分，在整个计算机市场份额中所占比例也很小。

- ➌ 超级计算机：具有最高性能和最高成本的一类计算机，一般配置为服务器，需要花费数千万甚至数亿美元。
- ➍ terabyte：一般简写作TB，原始定义为 $1\,099\,511\,627\,776 (2^{40})$ 字节，但有些通信和辅助存储系统将其重新定义为 $1\,000\,000\,000\,000 (10^{12})$ 字节。为了避免混淆，使用术语tebibyte(TiB)表示 $2^{40}$ 字节，而terabyte指 $10^{12}$ 字节。图1-1表示了十进制和二进制术语的范围。

十进制术语	缩写	数值	二进制术语	缩写	数值	数值差别
kilobyte	KB	$10^3$	kibibyte	KiB	$2^{10}$	2%
megabyte	MB	$10^6$	mebibyte	MiB	$2^{20}$	5%
gigabyte	GB	$10^9$	gibibyte	GiB	$2^{30}$	7%
terabyte	TB	$10^{12}$	tebibyte	TiB	$2^{40}$	10%
petabyte	PB	$10^{15}$	pebibyte	PiB	$2^{50}$	13%
exabyte	EB	$10^{18}$	exbibyte	EiB	$2^{60}$	15%
zettabyte	ZB	$10^{21}$	zebibyte	ZiB	$2^{70}$	18%
yottabyte	YB	$10^{24}$	yobibyte	YiB	$2^{80}$	21%

图1-1 通过为常用容量加一个二进制注释解决 $2^x$ 与 $10^y$ 字节的模糊性。最后一列表示了二进制术语大于相应的十进制术语的具体数值。在以bit为单位时，这些表示方法同样适用，因此gigabit(Gb)是 $10^9$ bit，而gigabit(Gib)是 $2^{30}$ bit

嵌入式计算机（embedded computer）是数量最多的一类计算机，应用和性能范围十分广泛，包括汽车、电视中的微处理器以及用来控制飞机和货船的处理器网络。嵌入式计算系统的设计目标是运行单一应用程序或者一组相关的应用程序，并且通常和硬件集成在一起以单一系统的方式一并交付用户。因此，尽管嵌入式计算机的数量庞大，还是有很多用户从来没有意识到他们正在使用计算机。

● 嵌入式计算机：嵌入到其他设备中的计算机，一般运行预定义的一个或者一组应用程序。

面向单一应用需求的嵌入式应用通常对成本或功耗有严格限制。以音乐播放器为例，处理器只需尽量快速地执行有限的功能，除此之外，降低成本和功耗是最大的目标。除了低成本的要求之外，由于故障可能会让使用者感到不适（例如，新电视机无法正常收看节目），也可能导致安全事故（例如，飞机或货船失事），因此嵌入式计算机对故障非常敏感。在面向消费者的嵌入式应用中（如数字家电）一般通过简单设计来获得可靠性——其重点在于尽可能地保证一项功能的正常运转；而在大型嵌入式系统中，采用了在服务器领域应用的多种冗余技术。尽管本书将重点放在通用计算机上，但是大多数概念可直接或者稍微修改之后用于嵌入式计算机。

**01 精解** 本书中的精解是正文中的一些段落，主要用来对读者可能感兴趣的内容做深入介绍。由于它并不影响后续内容的学习，因此对此部分不感兴趣的读者可以直接跳过。

许多嵌入式处理器使用处理器核。处理器核是利用硬件描述语言（如 Verilog 或 VHDL，见第 4 章）描述的处理器版本，它使得设计者能够把其他专用硬件与之集成起来制造在一块芯片上。

### 1.1.2 欢迎来到后 PC 时代

技术的持续进步给计算机硬件带来了革命性的变化，对整个信息技术工业产生了震动。就像 30 年前开始出现的个人计算机对产业带来的变化一样，我们已经从本书的上一版开始感受到这种变化。代替 PC 的是个人移动设备（Personal Mobile Device，PMD）。PMD 由电池供电，通过无线方式连接到网络，价格通常只有几百美元。另外，与 PC 一样，PMD 可下载软件（App）并进行运行。与 PC 不同的是，PMD 不再有键盘和鼠标，而采用触摸屏甚至语音作为输入。当今的 PMD 可以是智能手机或平板电脑，而明天的 PMD 可能会包括电子眼镜。图 1-2 给出了平板电脑和智能手机与 PC 和传统手机的增长速度的对比。

● 个人移动设备：连接到网络上的小型无线设备。PMD 由电池供电，通过下载 App 的方式安装软件。智能手机和平板电脑是典型的 PMD。

云计算（cloud computing）替代了传统的服务器，它依赖于称为仓储规模计算机（Warehouse Scale Computer，WSC）的巨型数据中心。像 Amazon 和 Google 这样的公司构建了包含 100 000 台服务器的 WSC，一些公司可以租用其中一部分为 PMD 提供软件服务，而不用自己构建 WSC。事实上，与 PMD 和 WSC 是硬件工业的革命类似，通过云计算实现的软件即服务（Software as a Service，SaaS）是软件工业的革命。当今的软件开发者通常在 PMD 和云上各运行其应用的一部分。

● 云计算：在网络上提供服务的大服务器集群，一些运营商根据应用需求出租不同数量的服务器。

- ② 软件即服务：在网络上以服务的方式提供软件和数据。其运行方式通常不是在本地设备上运行所有的二进制代码，而是通过诸如运行在本地客户端的浏览器等小程序登录到远程服务器上执行。典型的例子是 Web 搜索和社交网络。

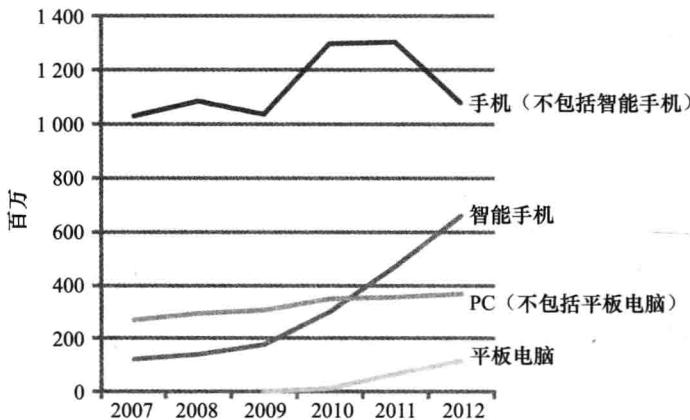


图 1-2 每年生产的平板电脑和智能手机与 PC 和传统手机的数量对比。平板电脑和智能手机代表着后 PC 时代。智能手机反映了手机工业的近期增长情况，并且在 2011 年超过了 PC 的产量。平板电脑产量增长最快，2012 年的产量几乎是 2011 年的两倍。PC 和传统手机的产量保持不变甚至在下降

### 1.1.3 你能从本书学到什么

成功的程序员总是关心其程序的性能，因为让用户快速得到结果对软件成功与否至关重要。7 在 20 世纪六七十年代，限制计算机性能的主要因素是内存容量。因而那时候程序员的信条是尽量少占用内存空间，以加速程序的运行速度。近十多年来，计算机和内存的设计技术有了长足进步。除了嵌入式系统以外，大多数用户对少占内存容量的需求大大减轻了。

现在，关心性能的程序员需要十分明确，20 世纪 60 年代的简单存储模型已经不复存在，现代计算机的特征是处理器的并行性和内存的层次性。另外，当今的程序员需要考虑运行在 PMD 或云上的程序的能效，这就要求他们了解自己的代码之下的很多细节（见 1.7 节）。因此，程序员为了创建有竞争力的软件版本，必须增加对计算机组成认知。

我们很荣幸有机会为你解释这些知识，阐述机箱覆盖之下的计算机内部软硬件是如何工作的。当你读完本书之后，我们相信，你将能够理解下面的问题：

- 用 C 或 Java 等高级语言编写的程序如何翻译成硬件之间的语言？硬件如何执行程序？领会这些概念是理解软硬件两者如何影响程序性能的基础。
- 什么是软硬件之间的接口，以及软件如何指导硬件完成其功能？这些概念对于许多软件的编写是十分重要的。
- 哪些因素决定了程序的性能？程序员如何才能改进其程序性能？从本书中我们将知道，程序性能取决于原始程序、将该程序转换为计算机语言的软件以及执行该程序的硬件的有效性。
- 什么技术可供硬件设计者用于改进性能？本书将介绍现代计算机设计的基本概念。有兴趣的读者可深入阅读我们的另一本进阶教材《Computer Architecture: A Quantitative Approach》。
- 硬件设计者可使用哪些技术提高能效？什么技术可供程序员提高或降低能效？
- 为什么串行处理近来发展为并行处理？这种发展带来的结果是什么？本书给出了解释，

并介绍了当今支持并行处理的硬件机制，全面评述了新一代的**多核微处理器**（multicore microprocessor）（见第6章）。

- 自1951年第一台商用计算机开始，计算机架构师们提出的哪些伟大的思想构成了现代计算机的基础？
- ② 多核微处理器：在一块集成电路上包含多个处理器（“核”）的微处理器。

8

如果无法理解这些问题，那么要在现代计算机上提升程序性能，或者要评估不同计算机解决特定问题的优劣将会是一个反复实验的复杂过程，而不是一个深入分析的科学过程。

本书第1章的目的是为其余各章奠定良好的基础。它介绍了各种基本概念和定义，指出如何正确地剖析软硬件，以及如何评价性能与功耗。它还介绍了集成电路（为计算机革命提供动力的技术），并在最后解释了向多核转移的原因。

在本章和后面几章里，读者会看到许多新的术语或者一些你听过却不知道其含义的术语。但是不用担心，在描述现代计算机时，确实会有很多专用术语，它们使我们能够精确描述计算机的功能或性能。另外，计算机设计人员（包括本书作者）喜欢用**首字母缩略词**（acronym），一旦知道了每个字母代表什么，就很容易理解了。为了帮助读者理解和记忆这些专用术语，在术语第一次出现时，我们会给出明确的定义。通过与这些术语的短时间接触，你将会熟练地正确使用这些术语的缩写，例如 BIOS、CPU、DIMM、DRAM、PCIe、SATA 和许多其他术语。

- ② 首字母缩略词：由一串单词中每个单词的首字母相连构成的单词。例如 RAM 是随机访问存储器（Random Access Memory）的缩略词，CPU 是中央处理单元（Central Process Unit）的缩略词。

为了加强对软件和硬件对于程序运行性能影响的理解，我们在全书中特别插入了“理解程序性能”，来对程序性能的理解加以概括。下面就是第一个。

**01 理解程序性能** 一个程序的性能取决于以下各因素的组合：程序所用算法的有效性，用来建立程序并将其翻译成机器指令的软件系统，计算机执行机器指令（可能包括 I/O 操作）的有效性。下表总结了硬件和软件是如何影响性能的。

软件或硬件组成元素	该元素如何影响性能	该论题出现的位置
算法	决定了源码级语句的数量和 I/O 操作的数量	其他书
编程语言、编译器和体系结构	决定了每条源码级语句对应的计算机指令数量	第2、3章
处理器和存储系统	决定了指令的执行速度	第4、5、6章
I/O 系统（硬件和操作系统）	决定了 I/O 操作可能的执行速度	第4、5、6章

9

为了说明本书中的思想的作用，在一连串章节中对完成一个矩阵与一个向量相乘的 C 程序进行了性能优化。每一步都可以帮助我们理解现代微处理器硬件如何能使性能提高 200 倍！

- 在第3章的数据级并行部分，使用C语言固有的子字并行使性能提高3.8倍。
- 在第4章的指令级并行部分，使用循环展开开发多指令发射和乱序执行硬件使性能再提高2.3倍。
- 在第5章的存储器层次优化部分，使用阻塞cache将大型矩阵处理性能再次提高2.5倍。

- 在第6章的线程级并行部分，在OpenMP中使用循环并行来开发多核硬件使性能再次提高14倍。

### 01 小测验

“小测验”的目的是帮助读者评估自己是否掌握了所学的概念，以及是否理解了这些概念的内涵。在这些小测验中，有些只有简单的答案，有些则是为了组内讨论。有些问题的答案可在章末找到。所有小测验只在节末出现，如果你确信自己对该部分内容完全理解，则可以跳过去。

- 每年嵌入式处理器的售出数量远远超过PC处理器甚至后PC处理器的数量。根据自己的经验，你是支持还是反对这种看法？列举你家中使用的嵌入式处理器，它与你家中桌面处理器的数量相比如何？
- 如前所述，软件和硬件都会影响程序的性能。请思考下述哪个例子属于性能瓶颈。
  - 所选算法
  - 编程语言或编译程序
  - 操作系统
  - 处理器
  - I/O系统和设备

10

## 1.2 计算机系统结构中的8个伟大思想

现在介绍计算机设计者在过去60年的计算机设计中提出的8个伟大思想。这些思想非常有用，以至于在首台应用它们的计算机之后的很长时间里，设计师在设计新的处理器时都会使用这些思想。这些思想将会贯穿本章和后续章节。为了说明它们的影响，本节对这些思想的含义以及亮点进行介绍，在本书的后续章节中将会明确使用它们近100次。

### 1.2.1 面向摩尔定律的设计

计算机设计者面临的一个永恒的问题就是摩尔定律（Moore's Law）驱动的快速变化。摩尔定律指出单芯片上的集成度每18~24个月翻一番。摩尔定律是Intel公司的创始人之一Gordon Moore在1965年对集成电路集成度做出的预测。由于计算机设计需要几年时间，因此在项目结束时，单芯片的集成度相对于设计开始时很容易翻一番甚至翻两番。像一个双向飞碟运动员一样，计算机设计者必须预测其设计完成时的工艺水平，而不是设计开始时的。

### 1.2.2 使用抽象简化设计

计算机架构师和程序员必须发明能够提高产量的技术，否则设计时间也将会向资源规模一样按照摩尔定律增长。提高硬件和软件生产率的主要技术之一是使用抽象（abstraction）来表示不同的设计层次，在高层次中看不到低层次的细节，只能看到一个简化的模型。

### 1.2.3 加速大概率事件

加速大概率事件（common case fast）远比优化小概率事件更能提高性能。大概率事件通常比小概率事件简单，从而易于提高。大概率事件规则意味着设计者需要知道什么事件是经常发生的，这只有通过仔细的实验与评估才能得出（见1.6节）。可以把加速大概率事想象成一辆赛车，由于通常情况下只有一两名乘客，因此提高赛车的速度要比提高小型货车的速度容易。

11

### 1.2.4 通过并行提高性能

从计算的诞生开始，计算机设计者就通过并行执行操作来提高性能。在本书中将会看到许多并行性能（parallel performance）的例子。

### 1.2.5 通过流水线提高性能

在计算机系统结构中，一个特别的并行性场景就是流水线（pipelining）。例如许多西部片中，一些坏人在制造火灾，在消防车出现之前会有一个“消防队列”来灭火——小镇的居民们排成一排通过水桶接力快速将水桶从水源传至火场，而不是每个人都在来回奔跑。可以把流水线想象成一系列水管，其中每一块代表一个流水级。

### 1.2.6 通过预测提高性能

遵循谚语“求人准许不如求人原谅”，最后一个伟大的思想就是预测（prediction）。在某些情况下，如果假定从误预测恢复执行代价不高并且预测的准确率相对较高，则通过猜测的方式提前开始某些操作，要比等到确切知道这些操作应该启动时才开始要快一些。

### 1.2.7 存储器层次

由于存储器的速度通常影响性能、存储器的容量限制了解题的规模、当今计算系统中存储器的代价占了主要部分，因此程序员希望存储器速度更快、容量更大、价格更便宜。设计师们发现可以通过存储器层次（hierarchy of memory）来解决这些相互矛盾的需求。在存储器层次中，速度最快、容量最小并且每位价格最昂贵的存储器处于顶层，而速度最慢、容量最大且每位价格最便宜的存储器处于最底层。在第5章将会看到，程序员看到的主存同时具有存储器层次中顶层的高速度和底层中的大容量和便宜的特征。可以把存储器层次想象成一个堆叠的三角形，该形状表示速度、价格和容量：越靠近顶端，存储器速度越快、每位价格越高；底层宽度越大，存储器容量越大。

### 1.2.8 通过冗余提高可靠性

计算机不仅需要速度快，还需要工作可靠。由于任何一个物理器件都有可能失效，因此可以通过使用冗余部件的方式提高系统的可靠性（dependable），冗余部件可以替代失效部件并可以帮助检测错误。可以通过牵引式挂车来理解可靠性：牵引式挂车后轴两边具有双轮胎，在一个轮胎出问题时卡车仍然可以继续工作。（在一个轮胎出问题时，卡车司机立即直接开往修理厂进行修理，从而恢复冗余性。）

12

## 1.3 程序概念入门

在巴黎，我对当地人讲法语，他们只是瞪着我看；我从来没能让这些白痴理解他们自己的语言。

——马克·吐温，《异国奇遇》，1869

一个典型的应用程序，如字处理程序或大型数据库系统，可以由数百万行代码构成，并依靠软件库来实现异常复杂的功能。众所周知，计算机中的硬件只能执行极为简单的低级指令。从复杂的应用程序到简单的指令需要经过几个软件层次来将复杂的高层次操作逐步解释或翻译

成简单的计算机指令，这可以作为伟大思想抽象的一个例子。

图 1-3 给出了这些软件的层次结构，外层是应用软件，中心是硬件，**系统软件**（systems software）位于两者之间。

- ➊ 系统软件：提供常用服务的软件，包括操作系统、编译程序、加载程序和汇编程序等。

系统软件有很多种，其中有两种对于现代计算机系统来说是必需的：操作系统和编译程序。**操作系统**（operating system）是用户程序和硬件之间的接口，为用户提供各种服务和监控功能。操作系统最为重要的作用是：

- 处理基本的输入和输出操作。
- 分配外存和内存。
- 为多个应用程序提供共享计算机资源的服务。

当前我们使用的操作系统主要有 Linux、iOS 和 Windows。

- 13 ➋ 操作系统：为了使程序更好地在计算机上运行而管理计算机资源的监控程序。

**编译程序**（compiler）完成另外一项重要功能：把用高级语言（如 C、C++、Java 或 Visual Basic 等）编写的程序翻译成硬件能执行的指令。这个翻译过程是相当复杂的，这里仅作简要介绍，第 2 章和附录 A 将作深入介绍。

- ➌ 编译程序：将高级语言翻译为计算机所能识别的机器语言的程序。

## 从高级语言到硬件语言

谈到电子硬件，首先需要谈到电信号的发送。对于计算机来说，最简单的信号是“通”和“断”。因此，计算机只用 2 个字母来表示。正如英语 26 个字母写多少不受限制一样，计算机的 2 个字母写多少也不受限制。代表两个字母的符号是 0 和 1，我们通常认为计算机语言就是二进制数。每个字母就是二进制元数字中的一个**二进制位**（binary digit）或一位（bit）。计算机服从于我们的命令，即计算机术语中的**指令**（instruction）。指令是能被计算机识别并执行的位串，可以将其视为数字。例如，位串

1000110010100000

告诉计算机将 2 个数相加。第 2 章将解释为什么数字元既表示指令又表示数据。我们不希望在此处涉及第 2 章的具体内容，但是使用数字既表述指令又表示数据是计算机的基础。

- ➍ 二进制位：也称为位。基数为 2 的数字中的 0 或 1，它是信息的基本组成元素。
- ➎ 指令：计算机硬件所能理解并服从的命令。

第一代程序员是直接使用二进制数与计算机通信的，这是一项非常乏味的工作。所以他们很快发明了助记符，以符合人类的思维方式。最初助记符是手工翻译成二进制的，其过程显然过于烦琐。随后设计人员开发了一种称为**汇编程序**（assembler）的软件，可以将助记符形式的指令自动翻译成对应的二进制。例如，程序员写下

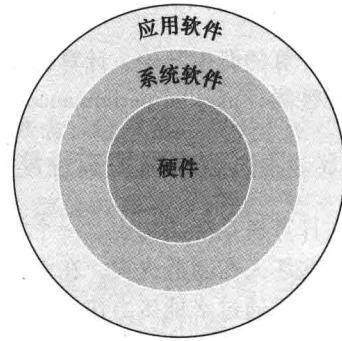


图 1-3 简化的硬件和软件层次图，将硬件作为同心圆的中心，应用程序软件作为最外层。在复杂的应用中，通常存在多层应用软件层。例如，一个数据库系统可运行于系统软件之上，而驻留在该系统软件上的某应用又反过来运行在该数据库之上

add A,B

汇编程序会将该符号翻译成

1000110010100000

该指令告诉计算机将 A 和 B 两个数相加。这种符号语言的名称今天还在用，即汇编语言（assembly language）。而机器可以理解的二进制语言是机器语言（machine language）。

- 汇编程序：将指令由助记符形式翻译成二进制形式的程序。
  - 汇编语言：以助记符形式表示的机器指令。
  - 机器语言：以二进制元形式表示的机器指令。

虽然这是一个巨大的进步，但汇编语言仍然与科学家用来模拟液体流动或会计师用来结算账目所使用的符号相去甚远。汇编语言需要程序员写出计算机执行的每条指令，要求程序员像计算机一样思考。

认识到可以编写一个程序来将需要更强大的高级语言翻译成计算机指令是计算机早期的一个重大突破。高级编程语言（high-level programming language）及其编译程序大大地提高了软件的生产率。图 1-4 表示了这些程序和编程语言之间的关系，这是抽象的另外一个例子。

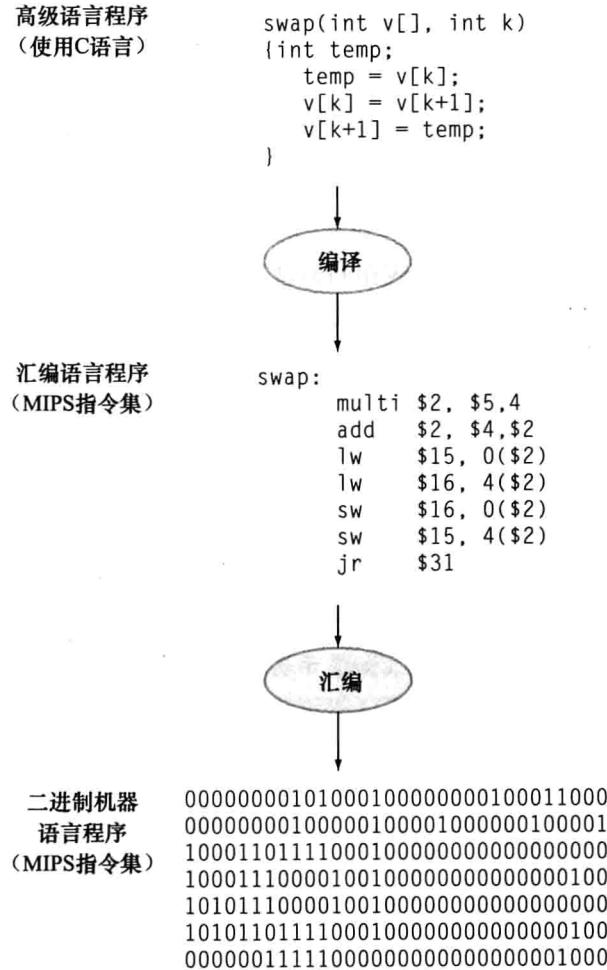


图 1-4 C 程序编译为汇编语言程序，再汇编为二进制机器语言程序。尽管将高级语言翻译成二进制的机器语言仅需要两步，但一些编译器将“中间人”砍掉，直接产生二进制的机器语言。这些语言和本图中列举的程序将在第 2 章详细检测

- ② 高级编程语言：如 C、C++、Java、Visual Basic 等可移植的语言，由一些单词和代数符号组成，可以由编译器转换为汇编语言。

编译程序使得程序员可以写出高级语言表达式：

A + B

编译程序将其编译为如下的汇编语言语句：

add A,B

然后，汇编程序将此语句翻译为二进制元指令，告诉计算机将这两个数 A 和 B 相加。

使用高级编程语言有以下几个好处。第一，可以使程序员用更自然的语言来思考，用英文和代数符号来表示，形成的程序看起来更像文字而不是密码表（见图 1-4）。而且，它们可按用途进行设计。例如，Fortran 是为科学计算设计的，Cobol 是为商业数据操作设计的，Lisp 是为符号操作设计的，等等。还有一些特定领域的语言，只为少数专业人群设计，如流体仿真的研究人员等。

第二，高级语言提高了程序员的生产率。如果使用较少行数的编程语言即可表示出设计用意，则可加速程序的开发，这是软件开发方面少有的共识之一。简明性是高级语言相对汇编语言最为明显的优势。

第三，采用高级语言编写程序提高了程序相对于计算机的独立性，因为编译程序和汇编程序能够把高级语言程序翻译成任何计算机的二进制元指令。高级编程语言的这些好处，使其直到今天仍应用广泛。

## 1.4 硬件概念入门

我们已经在上节通过程序揭示了计算机软件，在本节中我们将打开机箱盖学习其中的硬件。任何一台计算机的基础硬件都要完成相同的基本功能：输入数据、输出数据、处理数据和存储数据。本书的主题就是描述这些功能是怎样完成的，随后各章将分别讨论这 4 项任务。

本书在遇到重要知识点时，都会用“重点”标题加以强调，希望读者对其重点记忆。全书大致有 10 多个重要知识点，这里是第一个，即计算机是由完成输入、输出、处理和存储数据任务的 5 个部件构成的。

计算机的两个关键部件是输入设备（input device）和输出设备（output device），例如麦克风是输入设备，而扬声器是输出设备。输入为计算机提供数据，输出将计算结果送给用户。像无线网络等设备既是输入设备又是输出设备。

- ② 输入设备：为计算机提供信息的装置，如键盘。

16

- ② 输出设备：将计算结果输出给用户（如显示器）或其他计算机装置。

第 5 章和第 6 章将详细介绍 I/O 设备，这里由外部 I/O 设备开始先对计算机硬件做一些基本的介绍。

- 01 重点** 组成计算机的 5 个经典部件是输入、输出、存储器、数据通路（在计算机中也称运算器）和控制器，其中最后两个部件通常合称为处理器。图 1-5 表示了一台计算机的标准组成。该组成与硬件技术无关，你总能够把任何计算机（无论是现在的还是过去的）中的任何部件归于这 5 种之一。为了加深读者对这一重点的印象，我们将在每章开始都给出此图。

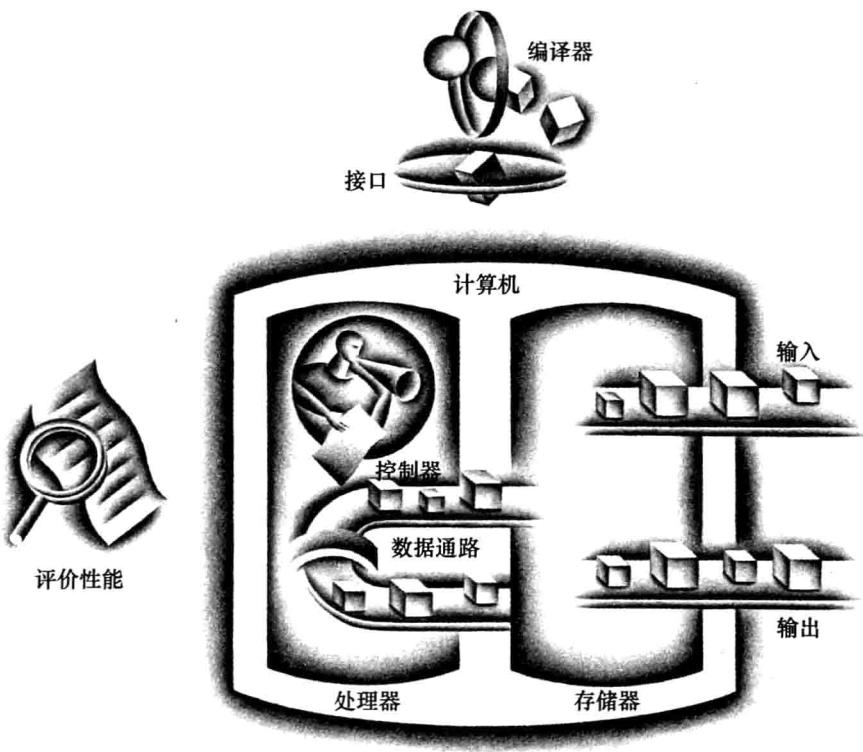


图 1-5 组成计算机的 5 个经典部件。处理器从存储器中得到指令和数据，输入部件将数据写入存储器，输出部件从内存中读出数据，控制器向数据通路、存储器、输入和输出部件发出命令信号

17

### 1.4.1 显示器

最吸引人的 I/O 设备应该是图形显示器了。大多数个人移动设备都用液晶显示 (Liquid Crystal Display, LCD) 来获得轻巧、低功耗的显示效果。LCD 并非光源，而是控制光的传输。典型的 LCD 内含棒状液态分子团形成的转动螺旋线，用来弯曲来自显示器后方的光线或者少量的反射光线。当电流通过时，液态分子棒不再弯曲，也不再使光线弯曲，由于两层相互垂直的偏光板之间充满液晶材料，如果它不弯曲则光线不能通过。（在不施加任何电压的情况下，液晶处于初始状态，并将入射光的方向扭转 90°，让背光源的入射光能够通过整个结构，在显示屏上呈现白色；而当施加电压时，光线不再弯曲，显示屏呈现为黑色。）今天，大多数 LCD 显示器采用动态矩阵显示 (active matrix display) 技术，其每个像素 (pixel) 都由一个晶体管精确地控制电流，使图像更清晰。在彩色动态矩阵 LCD 中，还有一个红 - 绿 - 蓝屏决定三种颜色分量的强度，每个点需要 3 个晶体管开关。

- ➊ 液晶显示：这是一种显示技术，用液体聚合物薄层的带电或者不带电来传输或者阻止光线的传输。
- ➋ 动态矩阵显示：一种液晶显示技术，使用晶体管控制单个像素上光线的传输。
- ➌ 像素：图像元素的最小单元。屏幕由成千上万的像素组成的矩阵而形成。

通过计算机显示器，我将飞机降落航空母舰的甲板上，观察到一个原子打到势阱中，乘着火箭以接近光的速度飞翔，同时我了解到计算机最深层的工作原理。

——Ivan Sutherland, 计算机图形学之父, 《Scientific American》, 1984

图像由像素矩阵组成，可以表示成二进制位的矩阵，称为位图（bit map）。针对不同的屏幕尺寸及分辨率，典型的屏幕上显示矩阵的大小可以从  $1024 \times 768$  到  $2048 \times 1536$ 。彩色显示器使用 8 位来表示每个三原色（红、绿和蓝），每个像素用 24 位表示，可以显示百万种不同的颜色。

计算机硬件采用光栅刷新缓冲区（又称为帧缓冲区）来保存位图以支持图像。要显示的图像保存在帧缓冲区中，每个像素的二进制值以刷新频率读出到显示设备。图 1-6 显示了一个用 4 位表示一个像素的简化设计的帧缓冲区。

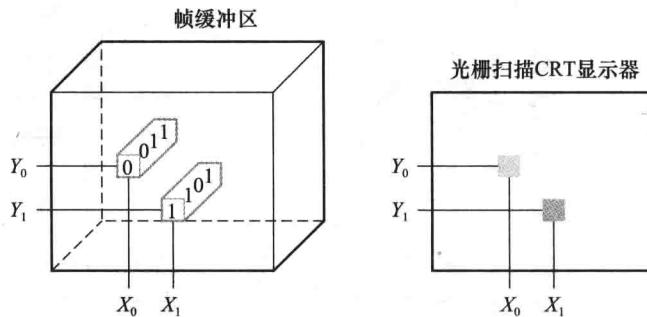


图 1-6 左边的帧缓冲区中每个坐标决定了右边光栅扫描 CRT 显示中相应坐标的灰度。像素  $(X_0, Y_0)$  的灰度值是 0011，小于像素  $(X_1, Y_1)$  的灰度值， $(X_1, Y_1)$  的灰度值是 1101

使用位图的目的是如实地在屏幕上进行显示。因为人眼可以分辨出屏幕上的细小变化，所以图像系统面临着挑战。

18

### 1.4.2 触摸屏

PC 使用 LCD，而后 PC 时代的平板电脑和智能手机使用接触敏感的显示设备替代了键盘和鼠标，拥有良好的用户界面，用户直接指向感兴趣的内容，而不需要使用鼠标。

触摸屏可采用多种方式实现，许多平板电脑采用电容感应实现。如果绝缘玻璃上覆盖一层透明的导体，人的手指接触到屏幕范围时，由于人是导体，将会使屏幕的电场发生变化，进而导致电容的变化。这种技术允许同时接触多个点，可提供非常好的用户界面。

### 1.4.3 打开机箱

图 1-7 给出了 Apple iPad 2 平板电脑的内部结构。不难看出，在计算机的五大传统部件中的 I/O 是该设备的主要部分。iPad 2 的 I/O 设备包括一个电容性的多触点 LCD、前置摄像头、后置摄像头、麦克风、耳机插孔、扬声器、加速计、陀螺仪、Wi-Fi 网络和蓝牙网络。其数据通路、控制器和存储器只占很小一部分。

图 1-8 中的小长方形是集成电路（integrated circuit），俗称芯片（chip）。其中心标有 A5 的芯片中含有两个运行频率为 1GHz 的 ARM 处理器。处理器是计算机中最活跃的部分。它严格按照程序中的指令运行，将数字相加，测试结果，并按结果发出控制信号使 I/O 设备做出动作，等等。有时候，人们把处理器称为中央处理单元（central processor unit），即 CPU。

- ② 集成电路：也叫芯片，一种将几十个至几百万个晶体管连接起来的设备。
- ② 中央处理器单元：也称为处理器，处理器是计算机中最活跃的部分，它包括数据通路和控制器，能将数字相加，测试结果，并按结果发出控制信号使 I/O 设备动作等。

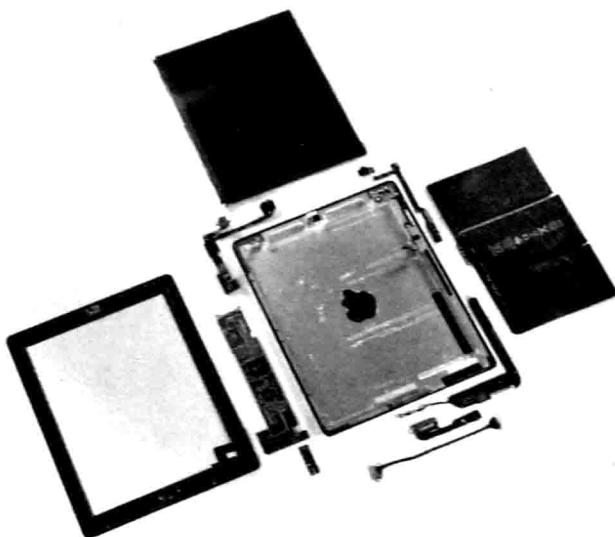


图 1-7 Apple iPad2 A1395 的组成。中间是 iPad 的金属背板（中心是倒置的 Apple 标志），顶部是电容性触摸屏和 LCD。最右端是 3.8V、25W·h 的聚合物电池，它包含三块锂离子电池芯，可以供电 10 小时。最左端是将 LCD 固定在背板上的金属外壳。在金属背板周围的小部件组成了我们熟知的计算机，它们在金属壳内位于电池旁边，呈 L 型排布。图 1-8 显示了靠近金属外壳左下部 L 型的逻辑主板的详细情况，上面有处理器和存储器，其下面的小方块中包含了提供无线通信的芯片，即 Wi-Fi、蓝牙和调频调谐器，它可以插在逻辑主板左下角的插槽中。外壳左上角是另外一个 L 型部件，它是前置摄像头组件，包括摄像头、耳机插孔和麦克风。外壳右上角的电路板除了加速计和陀螺仪，还包含了音量控制和静音/屏幕旋转锁定按钮。加速计和陀螺仪使得 iPad 可识别 6 向移动。旁边的小方块是后置摄像头。外壳右下角是 L 型的扬声器组件。底部的电缆连接逻辑主板和摄像/声音控制电路板。电缆和扬声器组件之间的电路板是电容性触摸屏的控制器（iFixit 友情提供，[www.ifixit.com](http://www.ifixit.com)）

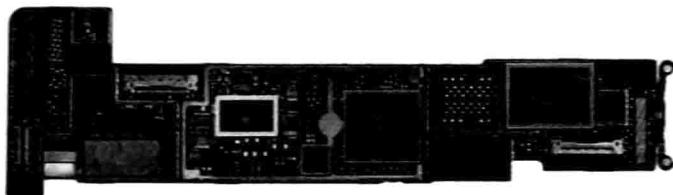


图 1-8 图 1-7 中 Apple iPad 2 的逻辑主板。图中突出了 5 块集成电路。中部的大 IC 是 Apple A5 芯片，包含了一个主频 1GHz 的双核 ARM 处理器和 512MB 的主存。图 1-19 是 A5 中处理器芯片的照片。左边大小相当的芯片是 32GB 的非易失性的闪存芯片。两块芯片之间的空间可以安装第二块存储器来扩展 iPad 的存储容量。A5 右边的芯片包含了电源控制和 I/O 控制芯片（iFixit 友情提供，[www.ifixit.com](http://www.ifixit.com)）

为进一步理解硬件，图 1-9 展示了一款微处理器的内部细节。处理器从逻辑上包括两个主要部件：数据通路和控制器，分别相当于处理器的肌肉和大脑。数据通路（datapath）负责完成算术运算，控制器（control）负责指导数据通路、存储器和 I/O 设备按照程序的指令正确执行。第 4 章将进一步详细说明数据通路和控制器。

- 数据通路：是处理器中执行算术操作的部分。
- 控制器：处理器中根据程序的指令指挥数据通路、存储器和 I/O 设备的部分。



图 1-9 A5 内部的处理器集成电路。芯片尺寸为  $12.1\text{mm} \times 10.1\text{mm}$ ，采用 45nm 工艺制造（见 1.5 节）。左半部分靠中间的位置是两个相同的 ARM 处理器，左上角的四分之一部分是具有 4 条数据通路的图形处理器单元（Graphic Processor Unit, GPU），左下角和底部是与主存的接口（Chipworks 友情提供，[www.chipworks.com](http://www.chipworks.com)）

图 1-8 中的 A5 芯片中还有两块存储器芯片，每块容量 2 gigabit，共 512MiB。内存（memory）是程序运行时的存储空间，它同时也用于保存程序运行时所使用的数据。内存由 DRAM 芯片组成。DRAM 是 dynamic random access memory（动态随机访问存储器）的缩写。内存由多片 DRAM 芯片组成，用来承载程序的指令和数据。与串行访问内存（如磁带）不同的是，无论数据存储在什么位置，DRAM 访问内存所需的时间基本相同。

- ➊ 内存：程序运行时的存储空间，同时还存储程序运行时所需的数据。
- ➋ DRAM：动态随机访问存储器，集成电路形式的存储器，可随机访问任何地址的内存。在 2012 年，其访问时间为 50ns，每 gigabyte 的价格为 5~10 美元。

进一步深入了解任何一个硬件部件会加深对计算机的理解。在处理器内部使用的是另外一

种存储器——缓存。缓存（cache memory）是一种小而快的存储器，一般作为 DRAM 的缓冲（缓存的一个非技术性的定义是一个隐藏事物的安全地方）。cache 采用的是另一种存储技术，称为静态随机访问存储器（Static Random Access Memory, SRAM），其速度更快而且不那么密集，因此价格比 DRAM 更贵（见第 5 章）。SRAM 和 DRAM 是存储器层次中的两层。

- ② 缓存：缓存是一种小而快的存储器，一般作为大而慢的存储器的缓冲。
- ③ 静态随机访问存储器：一种集成电路形式的存储器，但是比 DRAM 更快，集成度更低。

19  
21

如前所述，改进设计的一个伟大思想是抽象。最重要的抽象之一是硬件和底层软件之间的接口。鉴于其重要性，该抽象被命名为计算机的指令集体体系结构（instruction set architecture），或简称体系结构（architecture）。计算机体系结构包括程序员正确编写二进制机器语言程序所需的全部信息，如指令、I/O 设备等。一般来说，操作系统需要封装 I/O 操作、存储器分配和其他低级的系统功能细节，以便应用程序员无需在这些细节上分心。提供给应用程序员的基本指令集和操作系统接口合称为应用二进制接口（Application Binary Interface, ABI）。

- ④ 指令集体体系结构：也叫体系结构，是低层次软件和硬件之间的抽象接口，包含了需要编写正确运行的机器语言程序所需要的全部信息，包括指令、寄存器、存储访问和 I/O 等。
- ⑤ 应用二进制接口：用户部分的指令加上应用程序员调用的操作系统接口，定义了二进制层次可移植的计算机的标准。

计算机体系结构可以让计算机设计者独立地讨论功能，而不必考虑具体硬件。例如，我们讨论数字时钟的功能（如计时、显示时间、设置闹钟）时，可以不涉及时钟的硬件（如石英晶体、LED 显示、按钮）。计算机设计者将体系结构与体系结构的实现（implementation）分开考虑也是沿着同样的思路：硬件的实现方式必须依照体系结构的抽象。这些概念产生了另一个重点。

- ⑥ 实现：遵循体系结构抽象的硬件。

**01 重点** 无论硬件还是软件都可以使用抽象分成多个层次，每个较低的层次把细节对上层隐藏起来。抽象层次中的一个关键接口是指令集体体系结构——硬件和底层软件之间的接口。这一抽象接口使得同一软件可以由成本不同、性能也不同的实现方法来完成。

22

#### 1.4.4 数据安全

目前为止，我们已经理解了如何输入数据，如何使用这些数据进行计算，以及如何显示结果。然而，一旦关掉电源，所有数据就丢失了，因为计算机中的内存是易失性存储器（volatile memory）。与之不同的是，如果关掉 DVD 机的电源，所记录的内容将不会丢失，因为 DVD 采用的是非易失性存储器（nonvolatile memory）。

- ⑦ 易失性存储器：类似 DRAM 的存储器，仅在加电时保存数据。
- ⑧ 非易失性存储器：在掉电时仍可保持数据的存储器，用于存储运行间的程序，例如 DVD。

为了区分易失性存储器与非易失性存储器，我们将前者称为主存储器（main memory 或 primary memory），将后者称为二级存储器（secondary memory）。二级存储器形成了存储器层次中下面更低的一层。DRAM 自 1975 年起在主存储器中占主导地位，而磁盘（magnetic disk）在

二级存储器中占主导地位的时间更早。由于器件尺寸和前面所述的特点，非易失性半导体存储器——闪存（flash memory）在个人移动设备中替代了磁盘。图1-8所示的iPad 2中的芯片上包含了闪存。除了非易失性外，闪存比DRAM慢，但却便宜很多。虽然每位的价格高于磁盘，但是闪存在体积、电容、可靠性和能耗方面都优于磁盘。因此闪存是PMD的二级存储器的标准。遗憾的是，与硬盘和DRAM不同的是，闪存具有写100 000~1 000 000次后老化或损坏的弱点。因此，文件系统必须记录写操作的数目，而且具备避免存储器损坏的策略，例如，避免移动经常使用的数据。第5章将会详细介绍磁盘和闪存。

- ② 主存储器：也叫主要存储器。这个存储器用来保持运行中的程序，在现代计算机中一般由DRAM组成。
- ③ 二级存储器：非易失性存储器，用来保存两次运行之间的程序和数据。在个人移动设备中一般由闪存组成，在服务器中由磁盘组成。
- ④ 磁盘：也叫硬盘（hard disk），是使用磁介质材料构成的以旋转盘片为基础的非易失性二级存储设备。因为是旋转的机械设备，所以磁盘的定位时间大约是5~20毫秒，2012年每g字节的价格大约为0.05~0.1美元。
- ⑤ 闪存：一种非易失性半导体内存，单位价格和速度均低于DRAM，但单位价格比磁盘高，速度比磁盘快。其访问时间大约为5~50毫秒，2012年每g字节的价格大约为0.75~1美元。

#### 1.4.5 与其他计算机通信

我们已经介绍了如何输入、计算、显示和保存数据，但对于今天的计算机来说，还有一项不可缺少的功能：计算机网络。如图1-5所示，处理器与存储器和I/O设备连接。通过网络，计算机可以与其他计算机通信，从而扩展计算能力。当今网络已经十分普遍，逐步成为了计算机系统的主干。一台新型个人移动设备或服务器如果没有网络接口将是十分可笑的。联网的计算机具有如下几个主要优点：

- 通信：在计算机之间高速交换信息。
- 资源共享：有些I/O设备可以由网络上的计算机共享，不必每台计算机都配备。
- 远距离访问：用户可以不必在计算机的旁边，而是在很远的地方使用计算机。

根据传输速度以及信息传输的距离，通信代价随之增长，网络的传输距离和性能是多种多样的，最为普遍的网络类型是以太网。它的传输距离可达到1 000千米，传输速率可达到40Gbps。以太网的传输距离和速率可以将一个建筑物中同一层的计算机连接起来，这就形成了通常称为局域网（Local Area Network，LAN）的一个例子。局域网通过交换机进行连接，可以提供路由与安全服务。广域网可支持万维网（World Wide Web），作为因特网的骨干网，以光纤为基础并向通信公司租用。

- ② 局域网：一种在一定地理区域（例如在同一栋大楼内）使用的传输数据的网络。
- ③ 广域网：一种可将区域扩展到几百千米范围的网络。

在过去的30年间，因为广泛的使用和性能的大幅度提升，网络已经改变了计算的方式。在20世纪70年代，个人很难接触到电子邮件，网络和Web还不存在，物理上的邮件介质磁带成为传输两地之间大容量数据的主要载体。局域网根本不存在，少数几个广域网限制了容量和访问。

随着网络技术的进步，网络变得越来越便宜，速度越来越快。在30多年前，第一个标准局域网的最大带宽为10Mbps，支持数十台计算机共享工作。今天，局域网技术已能提供从100Mbps~10Gbps的带宽。光通信技术已经使广域网有了类似的发展，从几百Kbps到Gbps的

带宽，支持几百台到几百万台计算机与全球网络互连。网络规模的飞速扩大，伴随着带宽的急剧增长，使得网络技术成为最近 30 年来信息革命的中心。

最近 10 年来，新的联网创新变革了计算机通信的方式。推动后 PC 时代（PostPC Era）的无线技术广泛应用，加上原本用来生产无线电的廉价的半导体（CMOS）技术被用来生产存储器和微处理器，使其价格大幅度降低，产量剧增。当前无线通信技术（IEEE 标准 802.11）支持从 1Mbps 到近 100Mbps 的传输速率。无线技术和基于线路的网络相当不同，因为所有用户可以在最近的区域里共享电波。

### 01 小测验

半导体 DRAM、闪存和磁盘存储有很大差别。试从易失性、访问时间和价格三方面进行比较。

## 1.5 处理器和存储器制造技术

处理器和存储器正在以难以置信的速度进步，因为计算机设计者一直采用最新的电子技术进行设计，以期在竞争中取得优势。图 1-10 描述了不断进步的各种新型技术，包括其出现的时间和性价比。这些技术确定了计算机能够做什么，以及以多快的速度发展变化。我们相信，所有计算机专业人员都应该熟悉集成电路的基础知识。

年代	计算机中采用的技术	相对性价比
1951	真空管	1
1965	晶体管	35
1975	集成电路	900
1995	超大规模集成电路	2 400 000
2013	甚大规模集成电路	250 000 000 000

图 1-10 随着时间的发展，不同计算机实现技术的性价比。来源：波士顿计算机博物馆，其中 2013 年的数据由作者推断得到（见 1.12 节）

**晶体管**（transistor）仅仅是一种受电流控制的开关。集成电路（IC）是由成千上万个晶体管组成的芯片。当戈登·摩尔预测资源持续翻番时，他是在预测单芯片上晶体管数量的增长速度。为了描述这些晶体管从几百个增长到成千上万的情形，形容词“超大规模”被添加到术语中，简写为 VLSI，即超大规模集成电路（very large-scale integrated circuit）。

- 晶体管：一种由电信号控制的简单开关。
- 超大规模集成电路：由数十万到数百万晶体管组成的电路。

集成度的增长率是相当稳定的。图 1-11 表示自 1977 年以来 DRAM 容量的发展情况。近 20 多年以来，每隔 3 年 DRAM 的容量就增长 4 倍，累积增长已超过 16 000 倍。

为了理解集成电路的制造过程，我们从头开始介绍。芯片的制造从硅（silicon）开始，硅是沙子中的一种物质。因为硅的导电能力不强，因此称为**半导体**（semiconductor）。用特殊的化学方法对硅添加某些材料，可以把其细微的区域转变为以下三种类型之一：

- 良好的导电体（类似于细微的铜线或铝线）。
  - 良好的绝缘体（类似于塑料或玻璃膜）。
  - 可控的导电体或绝缘体（类似开关）。
- 硅：一种自然元素，它是一种半导体。
  - 半导体：一种导电性能不好的物质。

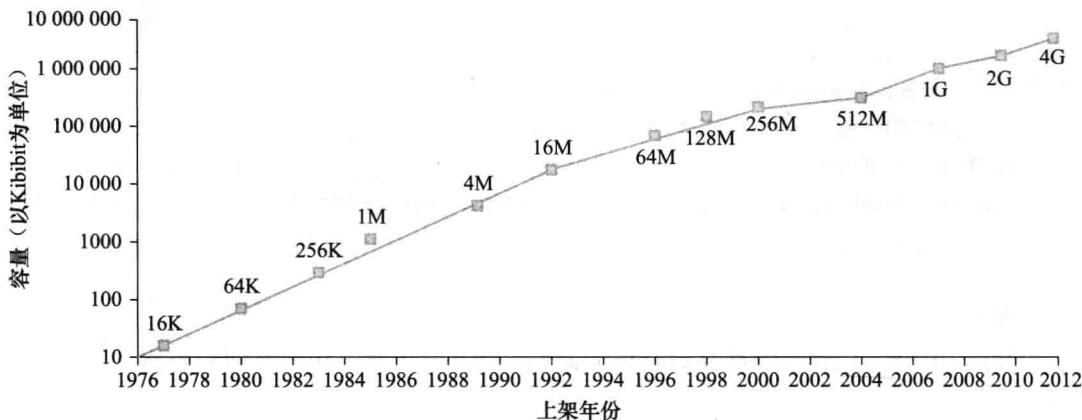


图 1-11 单片 DRAM 容量随时间的增长。纵轴单位为 Kb，其中 K 为 1024 ( $2^{10}$  位)。在近 20 多年中，平均每隔 3 年 DRAM 容量扩大 4 倍，即每年增长约 60%。在最近几年中，增长速度有所下降，接近每 2~3 年翻一番的水平。

晶体管属于第三种 VLSI 电路，是由数亿个上述三种材料组合起来并封装在一起所制成的。

集成电路的制造过程对芯片的价格非常关键，因此对计算机设计者十分重要。图 1-12 表示了集成电路制造的整个过程。集成电路的制造是从硅锭 (silicon crystal ingot) 开始的，它像一根巨大的香肠。目前使用的硅锭直径约 8~12 英寸，长度约 12~24 英寸。硅锭经切片机切成厚度不超过 0.1 英寸的晶圆 (wafer)。这些晶圆经过一系列化学加工过程最终产生之前所讨论的晶体管、导体和绝缘体。如今的集成电路只包含一层晶体管，但是可能具有多个绝缘层间隔的 2~8 层金属导体。

- 硅锭：一块由硅晶体组成的棒。直径大约在 8~12 英寸，长度约 12~24 英寸。
- 晶圆：厚度不超过 0.1 英寸的硅锭片，用来制造芯片。

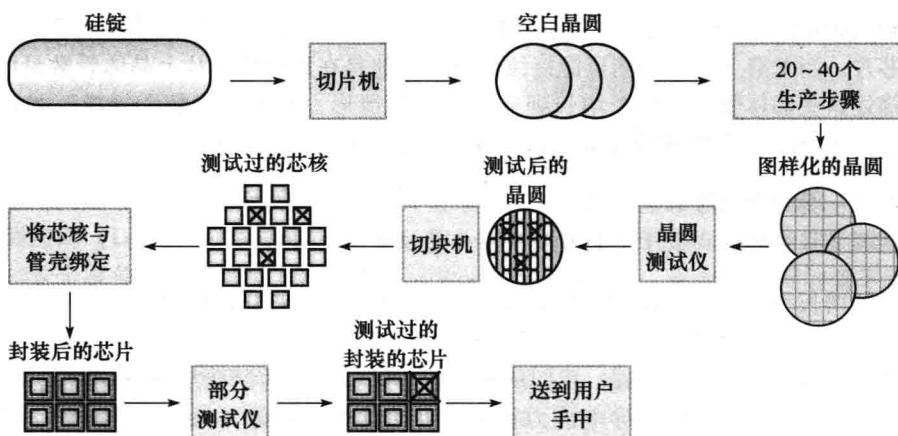


图 1-12 芯片制造的全过程。从硅锭切下来之后，空白的晶圆经过大约 20~40 步的加工，产生图样化的晶圆（见图 1-13）。这些图样化的晶圆由晶圆测试器进行测试，测试后生成一张图，表明哪些部分是合格的。之后，这些晶圆被进一步切成芯片（见图 1-9）。在本图中，一个晶圆能生产 20 个芯片，其中有 17 个通过测试（X 意味着这个芯片是坏的）。本例中芯片的良率（又称成品率）是 17/20，也就是 85%。这些合格芯片被封装起来并且在发布给用户之前经过多次测试。不合格的封装会在最终测试中被发现。

晶圆中或是在图样化的几十个步骤中出现一个细微的瑕疵就会使其附近的电路损坏，这些瑕疵（defect）使得制成一个完美的晶圆几乎是不可能的。有几种策略可以解决这一问题，最简单的策略是把晶圆切分成许多独立的晶圆，也就是现在所称的芯片（die），更正式的叫法是 chip。图 1-13 所示就是切分前的微处理器晶圆，而图 1-9 则是单个微处理器芯片。

26

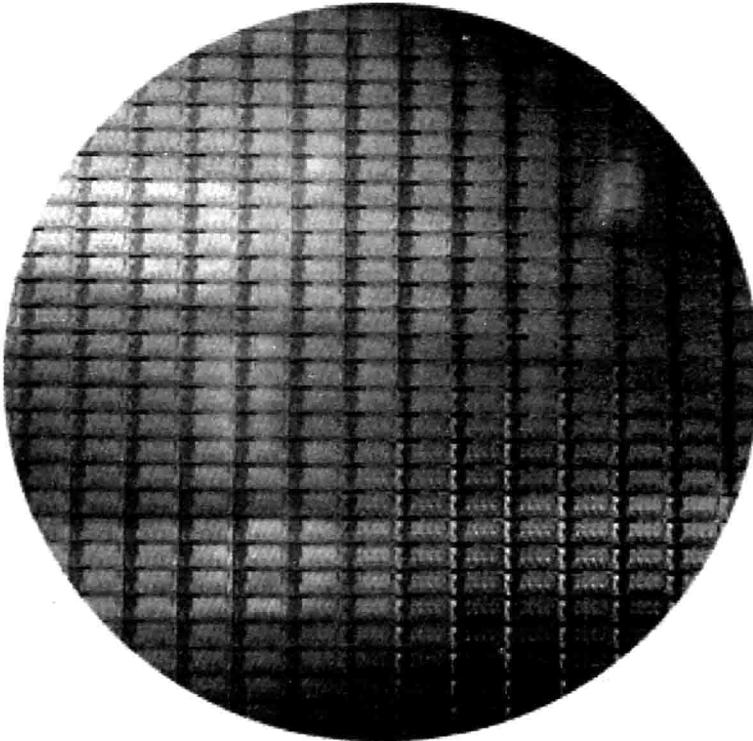


图 1-13 Intel Core i7 芯片的 12 英寸（300mm）晶圆（Intel 提供）。良率为 100% 的圆片中的晶圆的数目是 280，每个为  $20.7\text{mm} \times 10.5\text{mm}$ 。晶圆边缘几十个不完整的芯片是没用的。之所以包含它们，是因为这样给硅片生产掩膜相当容易。芯片使用 32nm 的工艺，这意味着最小的晶体管的尺寸几乎接近 32nm，尽管它们通常比实际的特征尺寸还要小，这个特征尺寸是将晶体管“图纸尺寸”和最终的生产尺寸相比得出的

通过切分，可以只淘汰那些有瑕疵的芯片，而不必淘汰整个晶圆。对这一过程的量化描述可以用成品率（yield）来表示，其定义为合格芯片数占总芯片数的百分比。

- ② 瑕疵：晶圆上一个微小的缺陷，或者在图样化的过程中因为包含这个缺陷而导致芯片失效。
- ③ 芯片：从晶圆中切割出来的一个单独的矩形区域，更加正式的英文名称是 chip。
- ④ 成品率：合格芯片数占总芯片数的百分比。

当芯片尺寸增大时，集成电路的价格会快速上升，因为成品率和晶圆中芯片的总数都下降了。为了降低价格，大芯片常采用下一代工艺进行尺寸收缩（包括晶体管和导线），从而改进每晶圆的芯片数和成品率。2012 年的典型工艺尺寸为 32nm，这意味着芯片上的最小特征尺寸是 32nm。

合格芯片要连接到 I/O 引脚上，这一过程称为封装。在封装之后，必须进行最后一次测试，因为封装过程也可能出错。最后芯片被交付给用户。

27

01 精解 集成电路的成本可以用下面 3 个简单公式来表示：

每芯片的价格 = 每晶圆的价格 / (每晶圆的芯片数 × 成品率)

每晶圆的芯片数 ≈ 晶圆面积 / 芯片面积

成品率 =  $1/(1 + (\text{单位面积的瑕疵数} \times \text{芯片面积}/2))^2$

第1个公式是直接导出的。第2个公式是近似的，因为没有减去晶圆边上不满足芯片矩形要求的面积（参见图1-13）。第3个公式是基于集成电路工厂的成品率经验，与重要加工步骤的数量呈指数关系。

因此，芯片的成本取决于成品率以及芯片和晶圆的面积，与芯片面积之间的关系一般不是线性的。

## 01 小测验

产量是决定集成电路价格的关键因素。下列哪些理由说明了芯片产量越高成本就越低？

1. 高产量使得在制造过程中能够适当调节设计，从而提高成品率。
2. 设计高产量芯片的工作量比设计低产量芯片小。
3. 制造芯片用的掩膜很贵，产量高时每芯片的掩膜成本就低。
4. 工程开发的成本高，并且基本与产量无关，故产量高时每芯片的开发成本较低。
5. 产量高时，通常每芯片的面积比产量低时小，因此成品率较高。

## 1.6 性能

对计算机的性能进行评价是富有挑战性的。由于现代软件系统的规模及其复杂性，加上硬件设计者采用了大量先进的性能改进方法，使性能评价极为困难。

28

在不同的计算机中挑选合适的产品，性能是极其重要的因素之一。精确地测量和比较不同计算机之间的性能对于购买者和设计者都很重要。销售计算机的人也需要知道这些。销售人员通常希望用户看到他们的计算机表现最好的一面，无论这一面是否能准确地反映购买者的应用需求。因此，理解怎样才能更合理地测量性能以及测定所选择的计算机的性能限制相当重要。

本节将首先介绍性能评价的不同方法，然后分别从计算机用户和设计者的角度描述性能测量的度量标准，最后还要分析这些度量标准之间有什么联系，并提出经典的处理器性能方程式，我们在全书中都要使用它进行性能分析。

### 1.6.1 性能的定义

当我们说一台计算机比另一台计算机具有更好的性能时意味着什么？虽然这个问题看起来很简单，但实际上却内藏玄机。我们可以先用客机问题模拟一下。图1-14表示若干典型客机的型号、载客量、航程、航速等参数。如果我们要指出表中哪架客机的性能最好，那么我们首先要对性能进行定义。如果考虑不同的性能度量，那么性能最佳的客机是不同的。我们可以看到，巡航速度最高的是Concorde，航程最远的是DC-8-50，载客量最大的是747。

飞机	载客量	航程 (英里)	航速 (英里/小时)	乘客吞吐率 (载客量 × 巡航速度)
波音777	375	4 630	610	228 750
波音747	470	4 150	610	286 700
BAC/Sud Concorde	132	4 000	1 350	178 200
Douglas DC-8-50	146	8 720	544	79 424

图1-14 若干商用飞机的载客量、航程和航速。最后一列展示的是飞机运载乘客的速度，它等于容量乘以航行速度（忽略距离、起飞和降落次数）

即使假定用速度来定义性能，这里仍然有两种可能的定义。如果你关心点对点的到达时间，那么可以将只搭载一名旅客的巡航速度最快的客机认为是性能最好的。如果你关心的是运输 450 名旅客，那么如图中最后一列所示，747 的性能是最好的。与此类似，我们可以用若干不同的方法来定义计算机性能。

如果你在两台不同的桌面计算机上运行同一个程序，那么你可以说首先完成作业的那台计算机更快。如果你运行的是一个数据中心，它有好几台服务器供很多用户投放作业，那你应该说在一天之内完成作业最多的那台计算机更快。个人计算机用户会对降低响应时间（response time）感兴趣，响应时间是指从开始一个任务到该任务完成的时间，又称为执行时间。而数据中心感兴趣的常常是吞吐率（throughput）。因此，在很多情形下，和关注吞吐率的服务器相比，我们需要对嵌入式以及台式计算机采用不同的应用程序作为测试基准和不同的性能度量标准。

- ① 响应时间：也叫执行时间（execution time），是计算机完成某任务所需的总时间，包括硬盘访问、内存访问、I/O 活动、操作系统开销和 CPU 执行时间等。
- ② 吞吐率：也叫带宽（bandwidth），性能的另一种度量参数，表示单位时间内完成的任务数量。

### 01 例题·吞吐率和响应时间

下面两种改进计算机系统的方式能否增加其吞吐率或减少其响应时间，或既增加其吞吐率又减少其响应时间？

1. 将计算机中的处理器更换为更高速的型号。
2. 增加多个处理器来分别处理独立的任务，如搜索万维网。

### 01 答案

一般来说，降低响应时间几乎都可以增加吞吐率。因此，方式 1 同时改进了响应时间和吞吐率。方式 2 不会使任务完成得更快，只会增加其吞吐率。□

但是，当需要处理更多的任务时，系统可能需要令后续请求排队。在这种情况下，随着吞吐率的增加，可同时改进响应时间，因为这缩小了排队等待时间。所以，在实际的计算机系统中，响应时间和吞吐率往往相互影响。

在讨论计算机性能时，本书前几章将主要考虑响应时间方面。为了使性能最大化，我们希望任务的响应时间或执行时间最小化。对于某个计算机 X，我们可以表达为：

$$\text{性能}_x = 1 / \text{执行时间}_x$$

如果有两台计算机 X 和 Y，X 比 Y 性能更好，则

$$\begin{aligned} \text{性能}_x &> \text{性能}_y \\ 1 / \text{执行时间}_x &> 1 / \text{执行时间}_y \\ \text{执行时间}_y &> \text{执行时间}_x \end{aligned}$$

也就是说，如果 X 比 Y 快，那么 Y 的执行时间比 X 长。

在讨论计算机设计时，经常要定量地比较两台不同计算机的性能。我们将使用“X 是 Y 的 n 倍快”的表态方式，即

$$\text{性能}_x / \text{性能}_y = n$$

如果 X 比 Y 快 n 倍，那么在 Y 上的执行时间是在 X 上执行时间的 n 倍，即

$$\text{性能}_X / \text{性能}_Y = \text{执行时间}_Y / \text{执行时间}_X = n$$

### 01 例题·相对性能

如果计算机 A 运行一个程序只需要 10 秒，而计算机 B 运行同样的程序需要 15 秒，那么计

算机 A 比计算机 B 快多少？

### 01 答案

我们知道，A 是 B 的  $n$  倍快，则

$$\text{性能}_A / \text{性能}_B = \text{执行时间}_B / \text{执行时间}_A = n$$

故性能比为

$$15/10 = 1.5$$

因此 A 是 B 的 1.5 倍快。 □

在以上的例子中，我们可以说，计算机 B 比计算机 A 慢 1.5 倍，因为

$$\text{性能}_A / \text{性能}_B = 1.5$$

意味着

$$\text{性能}_A / 1.5 = \text{性能}_B$$

简单地说，当我们试图将计算机的比较结果量化时，我们通常使用术语“比什么快”。因为性能和执行时间是倒数关系，提高性能就需要减少执行时间。为了避免对术语“增加”和“降低”潜在的误解，当我们想说“改善性能”和“改善执行时间”的时候，我们通常说“增加性能”或者“降低执行时间”。

## 1.6.2 性能的度量

如果用时间来度量计算机的性能，那么完成同样的计算任务，需要时间最少的计算机是最快的。程序的执行时间一般以秒为单位。然而，时间可以根据我们的计量方法选用不同的表示方法。对时间最直接的定义是墙上时钟时间（wall clock time），也叫响应时间（response time）、消逝时间（elapsed time）等。这些术语均表示完成任务所需的总时间，包括了硬盘访问、内存访问、I/O 操作和操作系统开销等一切时间。

多用户经常共享同一计算机，一个处理器需要同时运行几个程序。在这种情况下，系统可能更侧重于优化吞吐率，而不是最小化一个程序的响应时间。因此，我们往往要把运行我们自己的任务的时间与一般的响应时间区别开来。我们可以使用 CPU 执行时间（CPU execution time），简称 CPU 时间，它只表示在 CPU 上花费的时间，而不包括等待 I/O 或运行其他程序的时间。（需要注意的是，用户所感受到的是程序的响应时间，而不是 CPU 时间。）CPU 时间还可进一步分为用于用户程序的时间和操作系统为用户服务花去的 CPU 时间。前者称为用户 CPU 时间（user CPU time），后者称为系统 CPU 时间（system CPU time）。要精确区分这两种 CPU 时间是困难的，因为通常难以分清哪些操作系统的活动是属于哪个用户程序的，而且不同操作系统的功能也千差万别。

- CPU 执行时间：简称 CPU 时间，执行某一任务在 CPU 上所花费的时间。
- 用户 CPU 时间：在程序本身所花费的 CPU 时间。
- 系统 CPU 时间：为执行程序而花费在操作系统上的时间。

为了一致性，我们保持基于响应时间和基于 CPU 执行时间的性能差异。我们使用术语系统性能（system performance）表示空载系统的响应时间，并用术语 CPU 性能（CPU performance）表示用户 CPU 时间。本章我们概括介绍了计算机性能，既适用于响应时间的度量，也适用于 CPU 时间的度量，但本章的重点将放在 CPU 性能上。

### 01 理解程序性能 不同的应用关注计算机系统性能的不同方面。许多应用，特别是那些运行

在服务器上的应用，主要关注 I/O 性能，所以此类应用既依赖硬件又依赖软件，对墙上时钟时间最感兴趣。而在其他一些应用中，用户可能对吞吐率、响应时间或两者的复杂组合更为关注（例如，最差响应时间下的最大吞吐率）。要改进一个程序的性能，必须明确性能的定义，然后通过测量程序执行时间来寻找可能的性能瓶颈。在后面的章节中，我们将介绍如何在系统的各个部分寻找瓶颈，以改进性能。

虽然作为计算机用户我们关心的是时间，但当我们深入研究计算机的细节时，使用其他的度量可能更为方便。对计算机设计者来说，他们需要考虑如何度量计算机硬件完成基本功能的速度。几乎所有计算机都用时钟来驱动硬件中发生的各种事件。时钟间隔的时间称为时钟周期 (clock cycle)。也可用它的倒数来描述，称为时钟频率 (clock rate)。例如，时钟周期为 250ps，对应的时钟频率为 4GHz。在下一节，我们将形式化地定义硬件设计者的时钟周期和计算机使用者所指的秒之间的关系。

- ⌚ 时钟周期：也叫 tick、clock tick、clock period、clock 或 cycle，为计算机一个时钟周期的时间，通常是处理器时钟，一般为常数。
- ⌚ 时钟长度：每个时钟周期持续的时间长度。

### 01 小测验

1. 假设某个使用个人移动设备和云的应用受网络性能限制。那么对于下列 3 种方法，哪种只改进了吞吐率？哪种同时改进了响应时间和吞吐率？哪种都没有改进？
  - a. 在个人移动设备和云之间增加一条额外的网络信道，从而增加总的网络吞吐率，并减少获得网络访问的延迟（现在已经存在 2 条网络信道）。
  - b. 改进网络软件，从而减少网络通信延迟，但并不增加吞吐率。
  - c. 增加计算机的内存。
2. 计算机 B 运行给定的应用需要 28 秒，而计算机 C 的性能是计算机 B 的 4 倍。请问计算机 C 运行同样的应用需要多长时间？

### 1.6.3 CPU 性能及其因素

用户和设计者往往以不同的尺度看待性能。如果我们能掌握这些不同尺度之间的关系，就能确定一个设计的变化对性能的影响。由于我们都关注 CPU 性能，因而性能度量实际上针对的是 CPU 执行时间。下面一个简单的公式把最基本的尺度（时钟周期数和时钟周期时间）和 CPU 时间联系起来：

一个程序的 CPU 执行时间 = 一个程序的 CPU 时钟周期数 × 时钟周期时间  
由于时钟频率和时钟周期时间互为倒数，故

$$\text{一个程序的 CPU 执行时间} = \frac{\text{一个程序的 CPU 时钟周期数}}{\text{时钟频率}}$$

这个公式清楚地表明，硬件设计者减少一个程序的 CPU 时钟周期数，或减少时钟周期时间，就能改进性能。在后面几章中我们将看到，设计者经常要面对这些因素之间的权衡。许多技术在减少时钟周期数的同时也会引起时钟周期时间的增加。

### 01 例题·性能的改进

某程序在一台时钟频率为 2GHz 的计算机 A 上运行需要 10 秒。现在将设计一台计算机 B，希望将运行时间缩短为 6 秒。计算机的设计者采用的方法是提高时钟频率，但这会影响 CPU 其余部分的设计，使计算机 B 运行该程序时需要相当于计算机 A 的 1.2 倍的时钟周期数。那么计

算机设计者应该将时钟频率提高到多少?

### 01 答案

我们首先要知道在 A 上运行该程序需要多少时钟周期数:

$$\text{CPU 时间}_A = \text{CPU 时钟周期数}_A / \text{时钟频率}_A$$

$$10 \text{ 秒} = \text{CPU 时钟周期数}_A / 2 \times 10^9 \text{ 周期数/秒}$$

$$\text{CPU 时钟周期数}_A = 10 \text{ 秒} \times 2 \times 10^9 \text{ 周期数/秒} = 20 \times 10^9 \text{ 周期数}$$

B 的 CPU 时间公式为:

$$\text{CPU 时间}_B = 1.2 \times \text{CPU 时钟周期数}_A / \text{时钟频率}_B$$

$$6 \text{ 秒} = 1.2 \times 20 \times 10^9 \text{ 时钟周期数/时钟频率}_B$$

$$\text{时钟频率}_B = 1.2 \times 20 \times 10^9 \text{ 时钟周期数/6 秒} = 0.2 \times 20 \times 10^9 \text{ 时钟周期数/秒}$$

$$= 4 \times 10^9 \text{ 时钟周期数/秒} = 4 \text{ GHz}$$

因此,要在 6 秒内运行完该程序, B 的时钟频率必须提高为 A 的 2 倍。 □

## 1.6.4 指令的性能

上述性能公式没有涉及程序所需的指令数。然而,由于计算机是通过执行指令来运行程序的,因此执行时间一定依赖于程序中的指令数。一种考虑执行时间的方法是,执行时间等于执行的指令数乘以每条指令的平均时间。所以,一个程序需要的时钟周期数可写为:

$$\text{CPU 时钟周期数} = \text{程序的指令数} \times \text{每条指令的平均时钟周期数}$$

术语 CPI (clock cycle per instruction) 表示执行每条指令所需的时钟周期数的平均值。不同的指令需要的时间可能不同,CPI 是一个程序全部指令所用时钟周期数的平均值。CPI 提供了比较相同指令集的不同实现方式的方法,因为一个程序执行的指令数是一样的。

- ② CPI: 每条指令的时钟周期数,表示执行某个程序或者程序片段时每条指令所需的时钟周期平均数。

### 01 例题·性能公式的使用

假设我们有相同指令集的两种不同实现方式。计算机 A 的时钟周期为 250ps,对某程序的 CPI 为 2.0;计算机 B 的时钟周期为 500ps,对同样程序的 CPI 为 1.2。对于该程序,请问哪台计算机执行的速度更快?快多少?

### 01 答案

我们知道,对于固定的程序,每台计算机执行的总指令数是相同的,我们用 I 来表示。首先,求每台计算机的 CPU 时钟周期数:

$$\text{CPU 时钟周期数}_A = I \times 2.0$$

$$\text{CPU 时钟周期数}_B = I \times 1.2$$

现在,可以计算每台计算机的 CPU 时间:

$$\text{CPU 时间}_A = \text{CPU 时钟周期数}_A \times \text{时钟周期时间} = I \times 2.0 \times 250\text{ps} = 500 \times I\text{ps}$$

同理

$$\text{CPU 时间}_B = I \times 1.2 \times 500\text{ps} = 600 \times I\text{ps}$$

显然,计算机 A 更快。快多少由执行时间之比来计算

$$\text{CPU 性能}_A / \text{CPU 性能}_B = \text{执行时间}_B / \text{执行时间}_A = 600 \times I\text{ps} / 500 \times I\text{ps} = 1.2$$

因此,对于该程序计算机 A 是计算机 B 的 1.2 倍快。 □

### 1.6.5 经典的 CPU 性能公式

现在我们可以用指令数 (instruction count)、CPI 和时钟周期时间来写出基本的性能公式：

$$\text{CPU 时间} = \text{指令数} \times \text{CPI} \times \text{时钟周期时间}$$

或

$$\text{CPU 时间} = \text{指令数} \times \text{CPI} / \text{时钟频率}$$

这些公式特别有用，因为它们把性能分解为三个关键因素。如果知道实现方案或替代方案如何影响这三个参数，我们可用这些公式来比较不同的实现方案或评估某个设计的替代方案。

- ② 指令数：执行某程序所需的总指令数量。

36

#### 01 例题·代码段的比较

一个编译器设计者试图在两个代码序列之间进行选择。硬件设计者给出了如下数据：

	每类指令的 CPI		
	A	B	C
CPI	1	2	3

对于某行高级语言语句的实现，两个代码序列所需的指令数量如下：

代码序列	每类指令的数量		
	A	B	C
1	2	1	2
2	4	1	1

哪个代码序列执行的指令数更多？哪个执行速度更快？每个代码序列的 CPI 是多少？

#### 01 答案

代码序列 1 共执行  $2 + 1 + 2 = 5$  条指令。代码序列 2 共执行  $4 + 1 + 1 = 6$  条指令。所以，代码序列 1 执行的指令数更少。

基于指令数和 CPI，我们可以用 CPU 时钟周期公式计算出每个代码序列的总时钟周期数为：

$$\text{CPU 时钟周期数} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

因此

$$\text{CPU 时钟周期数}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ 周期}$$

$$\text{CPU 时钟周期数}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ 周期}$$

故代码序列 2 更快，尽管它多执行了一条指令。由于代码序列 2 总时钟周期数较少，而指令数较多，因此它一定具有较小的 CPI。CPI 的计算公式为：

$$\text{CPI} = \text{CPU 时钟周期数} / \text{指令数}$$

$$\text{CPI}_1 = \text{CPU 时钟周期数}_1 / \text{指令数}_1 = 10 / 5 = 2$$

$$\text{CPI}_2 = \text{CPU 时钟周期数}_2 / \text{指令数}_2 = 9 / 6 = 1.5$$

□ 37

#### 01 重点

图 1-15 给出了计算机在不同层次上的性能测试指标及其测试单位。通过这些指标的组合可以计算出程序的执行时间（单位为秒）：

$$\text{执行时间} = \text{秒}/\text{程序} = \text{指令数}/\text{程序} \times \text{时钟周期数}/\text{指令} \times \text{秒}/\text{时钟周期}$$

永远记住，唯一能够被完全可靠测量的计算机性能指标是时间。例如，对指令集减少指令数目的改进可能降低时钟周期时间或提高 CPI，从而抵消了改进的效果。类似地，CPI 与执行的指令类型相关，执行指令数最少的代码其执行速度未必是最快的。

性能的分量	测量单位
程序的CPU执行时间	程序执行的执行时间，以秒为单位
指令数目	程序执行的指令数目
指令的平均执行时钟周期(CPI)	每条指令的平均执行的时钟周期数
时钟周期时间	每个时钟周期的长度，以秒为单位

图 1-15 基本的性能指标及其测量单位

如何确定性能公式中这些因素的值呢？我们可以通过运行程序来测量 CPU 的执行时间，并且计算机的说明书中通常介绍了时钟周期时间。难以测量的是指令数和 CPI。当然，如果确定了时钟频率和 CPU 执行时间，我们只需要知道指令数或者 CPI 两者之一，就可以依据性能公式计算出另一个。

可以通过用体系结构仿真器等软件工具预执行程序来测量出指令数，也可以用大多数处理器中的硬件计数器来测量执行的指令数、平均 CPI 和性能损失源等。由于指令数量取决于计算机体系结构，并不依赖于计算机的具体实现，因而我们可以在不知道计算机全部实现细节的情况下对指令数进行测量。但是，CPI 与计算机的各种设计细节密切相关，包括存储系统和处理器结构（我们将在第 4、5 章中看到），以及应用程序中不同类型的指令所占的比例。因此，CPI 对于不同应用程序是不同的，对于相同指令集的不同实现方式也是不同的。

38

上述例子表明，只用一种因素（如指令数）去评价性能是危险的。当比较两台计算机时，必须考虑全部三个因素，它们组合起来才能确定执行时间。如果某个因素相同（如上例中的时钟频率），则必须考虑不同的因素才能确定性能的优劣。因为 CPI 随着指令组合 (instruction mix) 而变化，所以必须比较指令的条数和 CPU，即使时钟频率是相同的。在本章最后的练习题中，有几道是关于计算机和编译程序改进后对时钟频率、CPI 和指令数目影响的评价。在 1.10 节，我们将讨论一种因没有全面考虑各种因素而导致的对性能的误解。

◎ 指令组合：在一个或多个程序中，指令的动态使用频度的评价指标。

**01 理解程序性能** 程序的性能与算法、编程语言、编译程序、体系结构以及实际的硬件有关。下表概括了这些成分是如何影响 CPU 性能公式中的各种因素的。

硬件或软件指标	影响什么	如何影响
算法	指令数，可能的 CPI	算法决定源程序执行指令的数目，从而也决定了 CPU 执行指令的数目。算法也可能通过使用较快或较慢的指令影响 CPI。例如，当算法使用更多的除法运算时，将会导致 CPI 增大
编程语言	指令数，CPI	编程语言显然会影响指令数，因为编程语言中的语句必须翻译为指令，从而决定了指令数。编程语言也可影响 CPI，例如，Java 语言充分支持数据抽象，因此将进行间接调用，需要使用较高的 CPI 指令
编译程序	指令数，CPI	因为编译程序决定了源程序到计算机指令的翻译过程，所以编译程序的效率既影响指令数又影响 CPI。编译器会以复杂的方式影响 CPI
指令集体系结构	指令数，CPI 时钟频率	指令集体系结构影响 CPU 性能的所有 3 个方面，因为它影响完成某功能所需的指令数、每条指令的周期数以及处理器的时钟频率

**01 精解** 也许你期望 CPI 最小值为 1.0。在第 4 章我们将看到，有些处理器在每个时钟周期可对多条指令取指并执行。有些设计者用 IPC (instruction per clock cycle) 来代替 CPI。如一个处理器每时钟周期可执行平均 2 条指令，则它的  $IPC = 2$ ,  $CPI = 0.5$ 。

**01 精解** 虽然时钟周期时间传统上是固定的，但是为了节省能量或暂时提升性能，当今的计算机可以使用不同的时钟频率，因此我们需要对程序使用平均时钟频率。例如，Intel Core i7 处理器在处理器温度升高之前可以暂时将时钟频率提高 10%。Intel 称之为快速模式 (Turbo mode)。

### 01 小测验

某 Java 程序在桌面处理器上运行需时 15 秒。一个新版本的 Java 编译程序发行了，其编译产生的指令数量是旧版本 Java 编译程序的 0.6 倍，不幸的是，CPI 增加为原来的 1.1 倍。请问该程序在新版本的 Java 编译程序中运行速度是多少？从以下三个选项中选出正确答案。

- a.  $15 \times 0.6 / 1.1 = 8.2$  秒
- b.  $15 \times 0.6 \times 1.1 = 9.9$  秒
- c.  $15 \times 1.1 / 0.6 = 27.5$  秒

## 1.7 功耗墙

图 1-16 表示 30 年间 Intel 八代微处理器的时钟频率和功耗的增长趋势。两者增长几乎保持了几十年，但近几年来突然缓和下来。其原因在于两者是密切相关的，而且功耗已经到达了极限，无法再将处理器冷却下来。

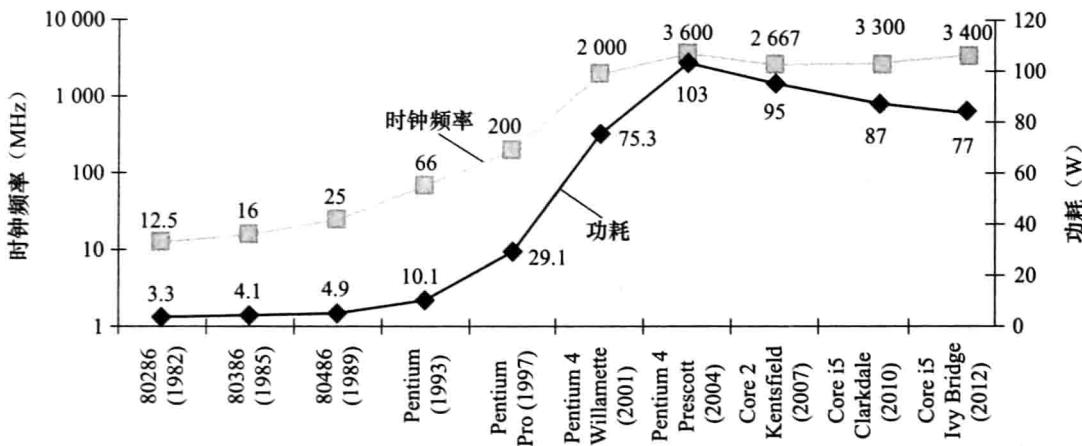


图 1-16 25 年间 Intel x86 八代微处理器的时钟频率和功耗。奔腾 4 处理器时钟频率和功耗提高很大，但是性能提升不大。Prescott 发热问题导致奔腾 4 处理器的生产线被放弃。Core 2 生产线恢复使用低时钟频率的简单流水线和片上多处理器。Core i5 采用同样的流水线

虽然功耗提供了能够冷却的极限，然而在后 PC 时代，能量是真正关键的资源。对于个人移动设备来说，电池寿命比性能更为关键。对于具有 100 000 个服务器的仓储式计算机来说，冷却费用非常高，因此设计者要尽量降低其功耗。就像在评价性能时，使用执行时间比使用 MIPS (见 1.10 节) 之类的比率更加可信一样，在评价功耗时，使用能耗比功耗更加合理，能耗的单位是焦耳/秒。

占统治地位的集成电路技术是 CMOS (互补型金属氧化半导体)，其主要的能耗来源是动态能耗，即在晶体管开关过程中产生的能耗，即晶体管的状态从 0 翻转到 1 或从 1 翻转到 0 消

耗的能量。动态能耗取决于每个晶体管的负载电容和工作电压：

$$\text{能耗} \propto \text{负载电容} \times \text{电压}^2$$

这个等式表示的是一个  $0 \rightarrow 1 \rightarrow 0$  或  $1 \rightarrow 0 \rightarrow 1$  的逻辑转换过程中消耗的能量。一个晶体管消耗的能量为：

$$\text{能耗} \propto 1/2 \times \text{负载电容} \times \text{电压}^2$$

每个晶体管需要的功耗是一个翻转需要的能耗和开关频率的乘积：

$$\text{功耗} \propto 1/2 \times \text{负载电容} \times \text{电压}^2 \times \text{开关频率}$$

开关频率是时钟频率的函数，负载电容是连接到输出上的晶体管数量（称为扇出）和工艺的函数，该函数决定了导线和晶体管的电容。

思考一下图 1-16 的趋势，为什么时钟频率增长为 1 000 倍，而功耗只增长为 30 倍呢？因为能耗和功耗是电压平方的函数，能够通过降低电压来大幅减少，每次工艺更新换代时都会这样做。一般来说，每代的电压降低大约 15%。20 多年来，电压从 5V 降到了 1V。这就是功耗只增长 30 倍的原因所在。

### 01 例题·相对功耗

假设我们需要开发一种新处理器，其负载电容只有旧处理器的 85%。再假设其电压可以调节，与旧处理器相比电压降低了 15%，进而导致频率也降低了 15%，问这对新处理器的动态功耗有何影响？

### 01 答案

$$\frac{P_{\text{新}}}{P_{\text{旧}}} = (\text{电容负载} \times 0.85) \times (\text{电压} \times 0.85)^2 \times \frac{(\text{开关频率} \times 0.85)}{(\text{电容负载} \times \text{电压}^2 \times \text{开关频率})}$$

因此功耗比为

$$0.85^4 = 0.52$$

新处理器的功耗大约为旧处理器的一半。 □

目前的问题是如果电压继续下降会使晶体管泄漏电流过大，就像水龙头不能被完全关闭一样。目前 40% 的功耗是由于泄漏造成的，如果晶体管的泄漏电流再大，情况将会变得无法收拾。

为了解决功耗问题，设计者连接大设备以增加冷却，而且将芯片中的一些在给定时钟周期内暂时不用的部分关闭。尽管有很多更加昂贵的方式来冷却芯片，但继续提高芯片的功耗（比如到 300 瓦）对个人计算机甚至服务器来说成本太高了，对个人移动设备就更不用说了。

由于计算机设计者遇到了功耗墙问题，因此他们需要开辟新的路径，选择不同于已经用了 30 多年的方法继续前进。

**01 精解** 虽然动态能耗是 CMOS 能耗的主要来源，但静态能耗也是存在的，因为即使在晶体管关闭的情况下，还是有泄漏电流存在。在服务器中，典型的电流泄漏占 40% 的能耗。因此，增加晶体管的数目，就会增加漏电功耗，即使这些晶体管总是关闭的。人们采用各种各样的设计和工艺创新来控制电流泄漏，但还是难以进一步降低电压。

**01 精解** 功耗成为集成电路设计的一个挑战有两个原因。首先，电源必须由外部输入并且分布到芯片的各个角落。现代微处理器通常使用几百个管脚作为电源和地！同样，多层次芯片互联仅仅为了解决芯片的电源和地的分布比例问题。其次，功耗作为热量形式散发，因此必须进行散热处理。服务器芯片的功耗可高达 100 瓦以上，因此芯片及外围系统的散热是仓储规模计算机的主要开销（见第 6 章）。

## 1.8 沧海巨变：从单处理器向多处理器转变

迄今为止，很多软件很像独唱者所写的音乐；使用当代的芯片，我们对于编写二重唱、四重唱以及小型合奏的经验很少，但是为大型交响乐或者合唱谱曲则是一个不同的挑战。

——Brian Hayes, 《Computing in a Parallel Universe》, 2007

功耗的极限迫使微处理器的设计产生了巨变。图 1-17 给出了桌面微处理器的程序响应时间的发展。从 2002 年起，其每年的增长速率从 1.5 下降到 1.2。

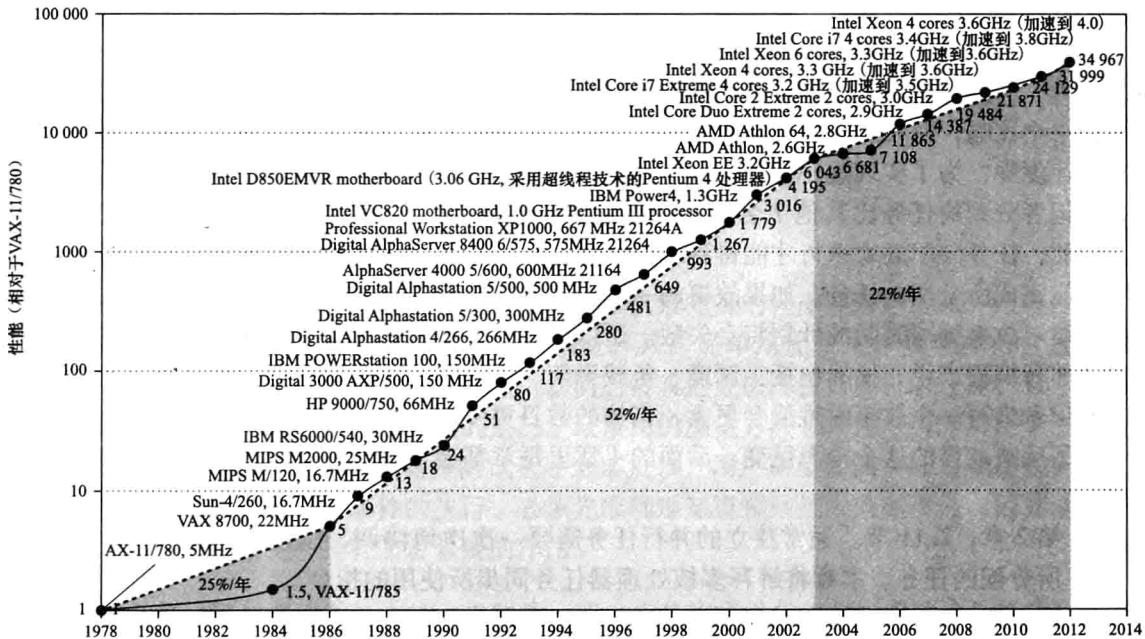


图 1-17 自 20 世纪 80 年代中期以来处理器性能的发展。本图描绘了和 VAX 11/780 相比，采用 SPECint 测试程序得到的性能数据（见 1.10 节）。在 20 世纪 80 年代中期以前，性能的增长主要靠技术驱动，平均每年增长 25%。在这个阶段之后，增长速度达到 52%，这归功于体系结构和组织方式的创新。从 20 世纪 80 年代中期开始，性能每年大约提高 52%，如果按照原先的 25% 的增长率计算，则到 2002 年的性能只有实际的 1/7。从 2002 年开始，受到功耗、指令级并行程度和存储器长延迟时间的限制，单核处理器的性能增长放缓，大约每年 22%

在 2006 年，所有桌面和服务器公司都在单片微处理器中加入了多个处理器，以求更大的吞吐率，而不再继续追求降低单个程序运行在单个处理器上的响应时间。为了减少 processor 和 microprocessor（微处理器）这两个词语之间的混淆，一些公司将 processor 作为“cores”的代称，这样 microprocessor 就是多核处理器了。因此，一个“四核”微处理器是一个包含了 4 个 processor 或者 4 个 core 的芯片。

在过去，程序员可以依赖于硬件、体系结构和编译程序的创新，无需修改一行代码，程序的性能每 18 个月翻一番。而今天，程序员要想显著改进响应时间，必须重写他们的程序以充分利用多处理器的优势。而且，随着核的数目不断加倍，程序员也必须不断改进他们的代码，以便在新微处理器上获得显著的性能提升。

为了强调软件和硬件系统的协同工作，我们在本书用“硬件/软件接口”的概念来进行描述，并对这一接口概括一些重要的观点，下面是本书中的第一个。

**01 硬件/软件接口** 并行性对计算性能一直十分重要，但它往往是隐蔽的。第 4 章将说明流水线，

它是一种漂亮的技术，通过指令重叠执行使程序运行得更快。这是指令级并行性的一个例子。在抽取了硬件的并行本质之后，程序员或编译程序可认为在硬件中指令是串行执行的。

迫使程序员意识到硬件的并行性，并显式地按并行方式重写其程序，曾经是计算机体系结构的“第三抱怨”，以致很多采用此种方式进行革新的公司都失败了（见6.15节）。从历史发展的角度来观察，整个IT行业已经把它放到了未来的发展方向上，程序员最终将成功地跃进到显式并行编程。

43

为什么程序员编写显式并行程序如此困难呢？第一个原因是并行编程以提高性能为目的，必然增加编程的难度。不仅程序必须要正确，能够解决重要问题，而且运行速度要快，还需要为用户或其他程序提供接口以便使用，否则编写一个串行程序就足够了。

第二个原因是为发挥并行硬件的速度，程序员必须将应用划分为每个核上有大致相同数量的任务，并同时完成。还要尽可能减小调度的开销，以不至于浪费并行性能。

44

作一个比喻，现在有一个写新闻故事的任务，如果由8名记者共同来完成，能否提高8倍的写作速度呢？为了实现这一目标，这个新闻故事需要进行划分，让每个记者都有事可做。假如某名记者分到的任务比其他7名记者加起来的任务还要多，那用8名记者的好处就不存在了。因此，任务分配必须平衡才能得到理想的加速。另一个存在的危险是记者要花费时间互相交流才能完成所分配的任务。如果故事的一部分，例如结论，在所有其他部分完成之前不能编写，则缩短故事编写时间的计划将会失败。所以，必须尽量减少通信和同步的开销。对于上述比喻和并行编程来说，挑战包括：调度、负载平衡、通信以及同步等开销。你也许会想到，当更多的记者来写一个故事时挑战会更大，当核的数目更多时，并行编程的挑战将更大。

为了反映业界的这个沧海巨变，后面的4章里每章都会至少有一节介绍有关并行性革命的内容：

- 第2章，2.11节。通常独立的并行任务需要一次次地协调，以便通报它们何时完成了所分配的任务。本章将解释多核处理器任务同步所使用的指令。
- 第3章，3.6节。并行性的最简单方式是将计算设备单元并行工作，例如两个向量相乘。**摩尔定律**提供了位宽更大且能同时处理多个操作数的算术单元，字并行就是利用这种资源的并行性的。
- 第4章，4.10节。尽管明确地知道并行编程的困难，但在20世纪90年代，人们依然付出了巨大的努力和投资用于从流水线开始研究硬件和编译程序的并行性。本章描述了这些技术，包括取指与多指令同时执行和通过预测的方式推测决策结果、指令执行等。
- 第5章，5.10节。降低通信开销的一个方法是让所有处理器使用同一个地址空间，任何处理器可以读写任何数据。今天的计算机都采用cache技术，即在处理器附近更快的存储器中，保持数据的一个临时复本。可以想象，如果多个处理器访问cache中的共享数据不一致的话，并行编程将尤为困难。本章将介绍保持所有cache数据一致性的机制。
- 第5章，5.11节。本节介绍如何使用许多磁盘共同构成一个能够提供更高吞吐率的系统，这就是廉价冗余磁盘阵列（RAID）的灵感。RAID流行的真正原因是它能够通过采用适当数量的冗余磁盘提供更高的可靠性。本节将介绍在不同RAID级别的性能、成本和可靠性。

45

除了这些章节之外，还有一整章介绍并行编程。第6章详细叙述了并行编程的挑战性，提出了两种方法来解决共享编址通信和显式消息传输，介绍了一种易于编程的并行性模型，讨论了使用基准测试程序对并行处理器进行评测的困难，为多核微处理器引入了一个新的简单性能模型，最后描述和评价了4种使用该种模型的多核微处理器。

如上所述，第3~6章使用矩阵向量相乘作为采用并行性提高性能的例子。

附录 C 介绍了一种在桌面计算机中越来越普及的图形处理器（Graphics Processing Unit, GPU）。它是为加速图像处理而发明的。得益于高度的并行性，GPU 表现出了优越的性能，并已发展为完善的编程平台。

附录 C 介绍了 NVIDIA GPU 及其并行编程环境。

## 1.9 实例：Intel Core i7 基准

我想，就像书一样，“计算机”是一个全世界广泛应用的概念。但我没有想到它会发展得如此迅速，因为我完全没有预料到我们在一块芯片上可以得到像我们最终得到的如此多的部件。晶体管的进步完全出乎我们的预料。它比我们预想的发展要快。

——J. Presper Eckert, ENIAC 的创建者之一, 1991

本书的每一章都有“实例”一节，它将本书中的概念与我们日常使用的计算机联系起来，这些小节涵盖了现代计算机中使用的技术。下面是本书中的第一个“实例”小节，我们将以 Intel Core i7 为例，说明如何制造集成电路，以及如何测量性能和功耗。

### 1.9.1 SPEC CPU 基准测试程序

用户日复一日使用的程序是用于评价新型计算机最完美的程序。所运行的一组程序集构成了工作负载（workload）。要评价两台计算机系统，只需简单地比较工作负载在两台计算机上的执行时间。然而大多数用户并不这样做，他们通过其他方法测量计算机的性能，希望这些方法能够反映计算机执行用户工作负载的情况。最常用的测量方法是使用一组专门用于测量性能的基准测试程序（benchmark）。这些测试程序形成负载，用户期望预测实际负载的性能。我们在前面提到，要加速大概率事件的执行，必须先准确地知道哪些是概率事件，因此基准测试程序在计算机系统结构中具有非常重要的作用。

- ① 工作负载：运行在计算机上的一组程序，可以直接使用用户的一组实际应用程序，也可以从实际程序中构建。一个典型的工作负载必须指明程序和相应的频率。
- ② 基准测试程序：用于比较计算机性能的程序。

SPEC (system performance evaluation cooperative) 是由许多计算机销售商共同出资赞助并支持的合作组织，目的是为现代计算机系统建立基准测试程序集。1989 年，SPEC 建立了重点面向处理器性能的基准程序集（现在称为 SPEC89）。历经 5 代发展，目前最新的是 SPEC CPU 2006，它包括 12 个整数基准程序集（CINT 2006）和 17 个浮点基准程序集（CFP 2006）。CINT 2006 包括 C 编译程序、量子计算机仿真、下象棋程序等，CFP 2006 包括有限元模型结构化网格法、分子动力学质点法、流体动力学稀疏线性代数法等。

图 1-18 列举了 SPEC 整数基准程序及其在 Intel Core i7 上的执行时间、指令数、CPI 和时钟周期时间等组成的 SPEC 分值。注意，CPI 的最大值和最小值相差达到 5 倍。

为了简化测试结果，SPEC 决定使用单一的数字来归纳所有 12 种整数基准程序。具体方法是将被测计算机的执行时间标准化，即将被测计算机的执行时间除以一个参考处理器的执行时间，结果称为 SPECratio。SPECratio 值越大，表示性能越快（因为 SPECratio 是执行时间的倒数）。CINT2006 或 CFP2006 的综合测试结果是取 SPECratio 的几何平均值。

**01 精解** 在使用 SPECratio 比较两台计算机时采用的是几何平均值，这样可以使得无论采用哪台计算机进行标准化都可得到同样的相对值。如果采用的是算术平均值，结果会随选用的参考计算机而变。

描述	名称	指令数目 ( $\times 10^9$ )	CPI	时钟周期时间 ( $\times 10^{-9}$ 秒)	执行时间 (秒)	参考时间 (秒)	Spec分值
字符串处理解释程序	perl	2 252	0.60	0.376	508	9 770	19.2
块排序压缩	bzip2	2 390	0.70	0.376	629	9 650	15.4
GNU C编译器	gcc	794	1.20	0.376	358	8 050	22.5
组合优化	mcf	221	2.66	0.376	221	9 120	41.2
go游戏(人工智能)	go	1 274	1.10	0.376	527	10 490	19.9
围棋游戏(人工智能)	hmmer	2 616	0.60	0.376	590	9 330	15.8
基因序列搜索	sjeng	1 948	0.80	0.376	586	12 100	20.7
量子计算机仿真	libquantum	659	0.44	0.376	109	20 720	190.0
视频压缩	h264avc	3 793	0.50	0.376	713	22 130	31.0
离散事件仿真库	omnetpp	367	2.10	0.376	290	6 250	21.5
游戏/寻找路径	astar	1 250	1.00	0.376	470	7 020	14.9
XML解析	xalancbmk	1 045	0.70	0.376	275	6 900	25.1
几何平均	-	-	-	-	-	-	25.7

图 1-18 SPECINTC2006 基准程序在 2.66GHz 的 Intel Core i7 920 上的运行结果。按照经典的 CPU 性能公式一节(原书第 36 页)的等式, 执行时间是本表的三个因素的乘积: 以亿为单位的指令数、每条指令的时钟数(CPI)以及纳秒级的时钟周期时间。SPECratio 仅仅是参考时间, 由 SPEC 提供, 被所测量的执行时间相除。SPECINTC2006 所引用的单个数目是 SPECratio 的几何平均数

几何平均值的公式是

$$\sqrt[n]{\prod_{i=1}^n \text{执行时间比}_i}$$

其中, 执行时间比<sub>i</sub>是总共 n 个工作负载中第 i 个程序的执行时间按参照计算机进行标准化的结果, 并且

$$\prod_{i=1}^n a_i \text{ 表示 } a_1 \times a_2 \times \cdots \times a_n$$

### 1.9.2 SPEC 功耗基准测试程序

由于能耗和功耗日益重要, SPEC 增加了一组用于评估功耗的基准测试程序, 它可以报告一段时间内服务器在不同负载水平下(以 10% 的比例递增)的功耗。图 1-19 给出了在基于 Intel Nehalem 处理器的服务器上的测试结果, 与前面类似。

目标负载 %	性能 (ssj_ops)	平均功耗 (瓦特)
100%	865 618	258
90%	786 688	242
80%	698 051	224
70%	607 826	204
60%	521 391	185
50%	436 757	170
40%	345 919	157
30%	262 071	146
20%	176 061	135
10%	86 784	121
0%	0	80
合计	4 787 166	1 922
$\sum \text{ssj\_ops} / \sum \text{power} =$		2 490

图 1-19 SPECpower\_ssj2008 在服务器上的运行结果。服务器的具体配置为双插槽 2.6GHz Intel Xeon X5650 处理器, 16GB DRAM, 100GB 固态硬盘

SPECpower 最早来自于面向 Java 商业应用的 SPEC 基准程序 (SPECJBB2005)，它主要测试处理器、cache、主存以及 Java 虚拟机、编译器、无用单元收集器、操作系统片段。性能采用吞吐率来测量，单位是每秒完成的操作次数。还是为了简化结果，SPEC 采用单个的数字来进行归纳，称为“overall ssj\_ops per watt”，其计算公式是：

$$\text{overall ssj\_ops per watt} = \left( \sum_{i=0}^{10} \text{ssj\_ops}_i \right) / \left( \sum_{i=0}^{10} \text{power}_i \right)$$

式中， $\text{ssj\_ops}_i$  为工作负载在每 10% 增量处的性能， $\text{power}_i$  是对应的功耗。

## 1.10 谬误与陷阱

科学一定开始于神话和对神话的批判。

—— Sir Karl Popper, 《The Philosophy of Science》, 1957

本书中每一章都会有“谬误与陷阱”一节，其目的是说明我们在实际中经常遇到的误解，我们称之为谬误。当讨论谬误时，我们会举出一个反例。我们也讨论陷阱，即那些容易犯的错误。通常陷阱是指一般原理只在有限的上下文中才是真的。本节旨在帮助你在设计或使用计算机时避免犯同样的错误。价格/性能谬误和陷阱使许多计算机架构师掉入圈套，包括我们下面开始介绍的本书的第一个陷阱，虽然它曾迷惑了许多设计者，却揭示了计算机设计中的一个重要关系。

**陷阱：**在改进计算机的某个方面时期望总性能的提高与改进大小成正比。

**加速大概率事件**的伟大思想导致的令人泄气的结果困扰着软件和硬件设计人员。它提醒我们一个事件需要的时间影响着改进的机会。

用一个简单的例子就可以很好地说明。假设一个程序在一台计算机上运行需要 100 秒，其中 80 秒的时间用于乘法操作。如果要把该程序的运行速度提高到 5 倍，乘法操作的速度应该改进多少？

改进以后的程序执行时间可用下面的 Amdahl 定律计算：

改进后的执行时间 = 受改进影响的执行时间 / 改进量 + 不受影响的执行时间  
代入本例的数据进行计算：

$$\text{改进后的执行时间} = 80/n + (100 - 80)$$

由于要求快至 5 倍，新的执行时间应该是 20：

$$20 = 80/n + 20$$

$$0 = 80/n$$

② **Amdahl 定律：**阐述了“对于特定改进的性能提升可能由所使用的改进特征的数量所限制”的规则。它是“收益递减定律”的量化版本。

可见，如果乘法运算占总负载的 80%，则无论怎样改进乘法，也无法达到性能提高至 5 倍的结果。特定改进的性能提升由所使用的改进特征的数量所限制。这个概念也产生了日常生活中我们称为“收益递减”的定律。

当我们知道一些函数所消耗的时间及其潜在的加速时，我们就可以使用 Amdahl 定律预测性能的提升。将 Amdahl 定律与 CPU 性能公式结合，是一种很方便的性能评价工具。读者可以在本章练习中进一步体会。

Amdahl 定律还应用于并行处理器数量的实际限制中，我们将在第 6 章中的“谬误与陷阱”中介绍。

**谬误：**利用率低的计算机功耗低。

服务器的工作负载是变化的，所以在低利用率的情况下功率很重要。例如，Google 仓储式计算机中 CPU 利用率大多数时间在 10% ~ 50% 之间，只有不到 1% 的时间达到 100%。即使花费 5 年时间来研究如何很好地运行 SPECpower 基准测试程式，在 2012 年，根据最好的结果配置的计算机中，只有 10% 的工作负载能够消耗 1/3 的峰值功耗。在实际工作中的系统由于没有针对 SPECpower 进行配置，因此其结果将会更加糟糕。

由于服务器的工作负载差异大且消耗了峰值功耗的很大比例，Luiz Barroso 和 Urs Holzle 提出需要对硬件重新进行设计，已达到“按能量比例计算”。这就是说，在未来的服务器中，10% 的工作负载使用 10% 的峰值功耗，这将减少数据中心的电费和二氧化碳的排放。

**谬误：**面向性能的设计和面向能量效率的设计具有不相关的目标。

由于能耗是功耗和时间的乘积，在通常情况下，对于软硬件的优化而言，即使在优化的部分起作用时能耗可能高了一些，但是这些优化缩短了系统运行时间，因此整体上还是节约了能量。一个重要的原因是当一个程序运行时，计算机的其他部分仍在消耗能量，因此，即使优化的部分多消耗了能量，运行时间的减少也可以减少整个系统的能耗。

**陷阱：**用性能公式的一个子集去度量性能。

50

我们早就指出了一种谬误：简单地只用时钟频率、指令数和 CPI 之一去预测性能。另一种常犯的错误是只用三种因素之二去比较性能。虽然这样做在有些条件下可能正确，但这种方法容易误用。实际上，几乎所有取代用时间去度量性能的方法都会导致歪曲的结果或错误的解释。

有一种用 **MIPS** (million instructions per second，每秒百万条指令) 取代时间以度量性能的方法。对于一个给定的程序，MIPS 表示为：

$$\text{MIPS} = \text{指令数} / (\text{执行时间} \times 10^6)$$

MIPS 是指令执行的速率，它规定了性能与执行时间成反比，越快的计算机具有越高的 MIPS 值。从表面看，MIPS 既容易理解，又符合人的直觉。

⦿ **MIPS：**基于百万条指令的程序执行速度的一种测量。指令条数除以执行时间与  $10^6$  之积就得到了 MIPS。

其实，用 MIPS 作为度量性能的指标存在三个问题。首先，MIPS 规定了指令执行的速率，但没有考虑指令的能力。我们没有办法用 MIPS 比较不同指令集的计算机，因为指令数肯定是不同的。其次，在同一计算机上，不同的程序会有不同的 MIPS，因而一台计算机不会只有一个 MIPS 值。例如，将执行时间用 MIPS、CPI、时钟频率代入之后可得：

$$\text{MIPS} = \text{指令数} / (\text{指令数} \times \text{CPI} / \text{时钟频率} \times 10^6) = \text{时钟频率} / (\text{CPI} \times 10^6)$$

回顾一下，图 1-18 显示了 SPEC2006 在 Intel Core i7 上的 CPI 最大值和最小值是相差 5 倍的，MIPS 也是如此。最后一点，也是最重要的一点，如果一个新程序执行的指令数更多，但每条指令的执行速度更快，则 MIPS 的变化是与性能无关的。

## 01 小测验

某程序在两台计算机上的性能测量结果为：

测量内容	计算机 A	计算机 B
指令数	100 亿次	80 亿次
时钟频率	4GHz	4GHz
CPI	1.0	1.1

- a. 哪台计算机的 MIPS 值更高？
- b. 哪台计算机更快？

51

## 1.11 本章小结

那里……ENIAC 配备有 18 000 个真空管，重量达 30 吨，未来的计算机具有 1000 个真空管，可能仅仅有 1.5 吨重。

——《Popular Mechanics》，1949. 3

虽然很难准确预测未来计算机的成本与性能将发展到怎样的水平，但可以确定的是一定会比现在的计算机更好。计算机性能水平的提高是永无止境的，计算机设计者和程序员必须理解更广泛的问题。

硬件和软件设计者都采用分层的方法构建计算机系统，每个下层都对其上层隐藏本层的细节。抽象原理是理解当今计算机系统的基础，但这并不意味着设计者只要懂得抽象原理就足够了。也许最重要的抽象层次是硬件和底层软件之间的接口，称为指令集体系结构。将指令集体系结构作为一个常量可以使不同的实现方法（价格和性能可能不同）能够运行同一软件。这种方法产生的一个副效应是，要预先排除可能需要接口发生变化的那些革新结构。

有一个可靠的测定性能的方法，即用实际程序的执行时间作为尺度。该执行时间与我们能够通过下面公式测量到的其他重要指标相关：

$$\frac{\text{秒数}}{\text{程序}} = \frac{\text{指令数}}{\text{程序}} \times \frac{\text{时钟周期数}}{\text{指令数}} \times \frac{\text{秒数}}{\text{时钟周期数}}$$

本书中我们将多次使用这一公式及其组成因子。必须明确的是，任何一个独立的因子都不能确定性能，只有三个因子的乘积（即执行时间）才是可靠的性能度量标准。

**01 重点** 执行时间是唯一有效且不可推翻的性能度量方法。人们曾经提出许多其他度量方法，但均以失败告终。有些从一开始就没有反映执行时间，因而是无效的；还有一些只能在有限条件下有效，超出了限制条件则失效，或是没有清晰地说明有效性的限制条件。

现代处理器的关键硬件技术是硅。与理解集成电路技术同样重要的是理解我们所期望的摩尔定律中描述的技术进步速率。在硅技术加快硬件进步的同时，计算机组织的新思想也改进了产品的性价比。其中有两个重要的新思想：第一，在程序中开发并行性，目前的典型方法是借助多处理器；第二，开发存储器层次结构的访问局部性，目前的典型方法是通过 cache。

能量效率已经取代芯片面积，成为微处理器设计中最重要的资源。保存功耗并且改进性能的需求已经迫使硬件工业向多核微处理器跃进，从而迫使软件工业向并行硬件编程跃进。并行化现在是提高性能的必要途径。

计算机设计总是以价格和性能来度量的，也包括其他一些重要的因素，如能耗、可靠性、成本和可扩展性等。尽管本章的重点在于价格、性能和能耗，但是最佳的设计应该在特定的应用领域中取得所有因素之间适当的平衡。

## 本书导读

在抽象的底部是计算机的 5 个经典部件：数据通路、控制器、存储器、输入和输出（见图 1-5）。这 5 个部件也是本书后面几章的框架：

- 数据通路：第 3、4、6 章和附录 C
- 控制器：第 4、6 章和附录 C
- 存储器：第 5 章
- 输入：第 5 章和第 6 章
- 输出：第 5 章和第 6 章

如上所述，第4章介绍处理器如何开发隐式并行性，第6章介绍并行革命的核心——显式并行多核微处理器，附录C介绍高度并行的图像处理器芯片。第5章介绍如何开发层次存储结构的访问局部性。第2章介绍指令集（编译器和计算机之间的接口），并强调了编译器和编程语言在利用指令集特性方面的作用。附录A提供了第2章指令集的参考数据。第3章介绍计算机如何处理算术运算数据。附录B介绍逻辑设计。

53

## 1.12 历史观点和拓展阅读

活跃的科学领域就像一个巨大的蚂蚁窝；人们消失在互相对立的观点中，以光速传递着信息，将信息从一个地方传到另一个地方。

——Lewis Thomas, 《Lives of a cell》中的“自然科学”，1974

本书的每一章都有“历史观点和拓展阅读”一节，可在本书配套网站上找到。我们可以通过一系列的计算机来追踪某一思想的发展历程，或者叙述一些历史上重要的项目贡献，还提供参考资料以便进一步探究。

本章的“历史观点”提供了几个关键思想的历史背景，其目的是向你介绍对技术进步做出贡献的重要历史人物以及他们的事迹。通过理解过去，你可以更好地理解那些推动未来计算技术进步的力量。配套网站中每个历史观点之后都会提示进一步阅读，这部分具体内容见配套网站中的“进一步阅读”部分。在配套网站可下载1.12节的剩余部分。

## 1.13 练习题

完成练习所需的相对时间比率标示在题号之后的方括号中。平均来说，做标记[10]的练习用的时间是做标记[5]的练习的2倍。做题前应先阅读的章节则标示在尖括号中。例如，<1.4>表示你应该在读过1.4节后才能完成本题。

- 1.1 [2]<1.1>列举和描述除智能手机之外的4种类型的计算机。
- 1.2 [5]<1.2>计算机系统结构中的8个伟大思想与其他领域的思想相同。将计算机系统结构中的8个伟大思想“面向摩尔定律的设计”、“使用抽象简化设计”、“加速大概率事件”、“采用并行提高性能”、“采用流水线提高性能”、“采用预测提高性能”、“存储器层次”、“通过冗余提高可靠性”与其他领域的下列思想进行匹配：
  - a. 汽车制造中的组装生产线
  - b. 吊桥缆索
  - c. 采用风向信息的飞机和船舶导航系统
  - d. 高楼中的高速电梯
  - e. 图书馆的预定台
  - f. 通过增大CMOS晶体管的栅极面积来减小翻转时间
  - g. 增加电磁飞机弹射器（不同于流体驱动模型，它采用电驱动），允许有新型反应堆技术才生成更多的能量
  - h. 制造自动驾驶汽车，其控制系统是安装在汽车上的传感器系统，例如车道偏离检测系统和智能导航控制系统
- 1.3 [2]<1.3>讲述高级语言（例如C）编写的程序转化为能够直接在计算机处理器上执行的表示的步骤。
- 1.4 [2]<1.4>一个彩色显示器中的每个像素由三种基色（红，绿，蓝）构成，每种基色用8位表示，分辨率为 $1280 \times 1024$ 像素。
  - a. 为了保存一帧图像最少需要多大的缓存（以字节计算）？
  - b. 在100Mbit/s的网络上传输一帧图像最少需要多长时间？

54

- 1.5** [4] <1.6> 有 3 种不同的处理器 P1、P2 和 P3 执行同样的指令集, P1 的时钟频率为 3GHz, CPI 为 1.5; P2 的时钟频率为 2.5GHz, CPI 为 1.0; P3 的时钟频率为 4GHz, CPI 为 2.2。
- 以每秒钟执行的指令数目为标准, 哪个处理器性能最高?
  - 如果每个处理器执行一个程序都花费 10 秒钟时间, 求它们的时钟周期数和指令数。
  - 我们试图把执行时间减少 30%, 但这会引起 CPI 增加 20%。问: 时钟频率应该是多少才能达到时间减少 30% 的目的?
- 1.6** [20] <1.6> 同一个指令集体体系结构有两种不同的实现方式。根据 CPI 的不同将指令分成 4 类 (A、B、C 和 D), P1 的时钟频率为 2.5GHz, CPI 分别为 1、2、3 和 3; P2 时钟频率为 3GHz, CPI 分别为 2、2、2 和 2。
- 给定一个程序, 有  $1.0 \times 10^6$  条动态指令, 按如下比例分为 4 类: A, 10%; B, 20%; C, 50%; D, 20%。
- 每种实现方式总的 CPI 是多少?
  - 计算两种情况下的时钟周期。
- 1.7** [15] <1.6> 编译程序对一个应用在给定的处理器上的性能有极深的影响。假定一个程序, 如果采用编译程序 A, 则动态指令数为  $1.0 \times 10^9$ , 执行时间为 1.1s; 如果采用编译程序 B, 则动态指令数为  $1.2 \times 10^9$ , 执行时间为 1.5s。
- 在给定处理器时钟周期为 1ns 时, 找出每个程序的平均 CPI。
  - 假定编译程序是在两个不同的处理器上运行的。如果这两个处理器的执行时间相同, 求运行编译程序 A 的处理器时钟相对于运行编译程序 B 的处理器的时钟快多少?
  - 假设开发了一种新的编译程序, 只用  $6.0 \times 10^8$  条指令, 平均 CPI 为 1.1。求这种新的编译程序相对于原先编译程序 A 和 B 的加速比。
- 1.8** 2004 年发布的 Pentium 4 Prescott 处理器时钟频率为 3.6GHz, 工作电压为 1.25V。假定平均情况下静态功耗为 10W, 动态功耗为 90W。
- 2012 年发布的 Core i5 Ivy Bridge 时钟频率为 3.4GHz, 工作电压为 0.9V。假定平均情况下静态功耗为 30W, 动态功耗为 40W。
- 1.8.1** [5] <1.7> 分别求出每个处理器的平均电容负载。
- 1.8.2** [5] <1.7> 对于每种工艺, 求出静态功耗占总功耗的比例和静态功耗相对于动态功耗的比率。
- 1.8.3** [15] <1.7> 如果要将整体功耗降低 10%, 求出在保持漏电流不变的情况下电压要降低多少?  
注意: 功耗定义为电压与电流的乘积。
- 1.9** 在一个处理器中, 假定算术指令、load/store 指令和分支指令的 CPI 分别是 1、12 和 5。另外假定一个程序在单个处理器核上运行时需要执行  $2.56 \times 10^9$  条算术指令、 $1.28 \times 10^9$  条 load/store 指令和 2.56E8 条分支指令, 并假定处理器的时钟频率为 2GHz。  
现假定程序并行运行在多核上, 分配到每个处理器核上运行的算术指令和 load/store 指令数目为单核情况下相应指令数目除以  $0.7 \times p$  ( $p$  是处理器的数量), 而每个处理器的分支指令的数量保持不变。
- 1.9.1** [5] <1.7> 求出当该程序分别运行在 1、2、4 和 8 个处理器核上的执行时间, 并求出其他情况下相对于单核处理器的加速比。
- 1.9.2** [10] <1.6, 1.8> 如果算术指令的 CPI 加倍, 对分别运行在 1、2、4 和 8 个处理器核上的执行时间有何影响?
- 1.9.3** [10] <1.6, 1.8> 如果要使单核处理器的性能与四核处理器相当, 单处理器中 load/store 指令的 CPI 应该降低多少? 假定四核处理器的 CPI 保持不变。
- 1.10** 假定一个直径 15cm 的晶圆的成本是 12, 包含 84 块芯片, 其缺陷参数为  $0.020 \text{ 瑕疵}/\text{cm}^2$ 。而一个直径 20cm 的晶圆的成本是 15, 包含 100 块芯片, 其缺陷参数为  $0.031 \text{ 瑕疵}/\text{cm}^2$
- 1.10.1** [10] <1.5> 分别求出每种芯片的成品率。
- 1.10.2** [5] <1.5> 分别求出每种芯片的价格。
- 1.10.3** [5] <1.5> 如每晶圆的芯片数增加 10%, 每单位面积的瑕疵数增加 15%, 求芯片面积和成品率。

55

56

- 1.10.4** [5] <1.5> 假设随着电子器件制造技术的进步，成品率从 0.92 上升到 0.95。给定芯片面积为 200mm<sup>2</sup>，求每一种技术下单位面积的瑕疵数。
- 1.11** SPEC CPU 2006 的 bzip2 基准程序在 AMD Barcelona 处理器上执行的总指令数为  $2.38 \times 10^{12}$ ，执行时间为 750s，参考时间为 9 650s。
- 1.11.1** [5] <1.6, 1.9> 如果时钟周期时间为 0.333ns，求 CPI 值。
- 1.11.2** [5] <1.9> 求 SPEC 的分值。
- 1.11.3** [5] <1.6, 1.9> 如果基准程序的指令数增加 10%，CPI 不变，求 CPU 时间增加多少？
- 1.11.4** [5] <1.6, 1.9> 如果基准程序的指令数增加 10%，CPI 增加 5%，求 CPU 时间增加多少？
- 1.11.5** [5] <1.6, 1.9> 根据上题中指令数和 CPI 的变化，求 SPEC 分值的变化。
- 1.11.6** [10] <1.6> 假设开发了一款新的 AMD Barcelona 处理器，其工作频率为 4GHz，在其指令集中增加了一些新的指令，从而使程序中指令数目减少了 15%，程序的执行时间减少到了 700s，新的 CPI 分值为 13.7，求新的 CPI。
- 1.11.7** [10] <1.6> 当时钟频率由 3GHz 上升到 4GHz 时，上一小题算出的 CPI 比 1.11.1 的高。请确定 CPI 的升高是否与频率升高相同？如果不同，为什么？
- 57** **1.11.8** [5] <1.6> CPU 时间减少了多少？
- 1.11.9** [10] <1.6> 对第二个基准程序 libquantum，假定执行时间为 960ns，CPI 为 1.61，时钟频率为 3GHz。在时钟频率为 4GHz 时，在不影响 CPI 的前提下执行时间降低 10%，求指令数。
- 1.11.10** [10] <1.6> 在指令数和 CPI 保持不变的前提下，如果要将 CPU 时间进一步减少 10%，求时钟频率。
- 1.11.11** [10] <1.6> 在指令数保持不变的前提下，如果要将 CPI 降低 15%，CPU 时间减少 20%，求时钟频率。
- 1.12** 1.10 节引证了一个用性能公式的一个子集去计算性能的陷阱。为了说明它，考虑下面两种处理器。P1 的时钟频率为 4GHz，平均 CPI 为 0.9，需要执行  $5.0 \times 10^9$  条指令；P2 的时钟频率为 3GHz，平均 CPI 为 0.75，需要执行  $1.0 \times 10^9$  条指令。
- 1.12.1** [5] <1.6, 1.10> 一个常见的错误是，认为时钟频率最高的计算机具有最高的性能。这种说法正确吗？请用 P1 和 P2 来验证这一说法。
- 1.12.2** [10] <1.6, 1.10> 另一个错误是，认为执行指令最多的处理器需要更多的 CPU 时间。考虑 P1 执行  $1.0 \times 10^9$  条指令序列所需的时间，P1 和 P2 的 CPI 不变，计算一下 P2 用同样的时间可以执行多少条指令？
- 1.12.3** [10] <1.6, 1.10> 一个常见的错误是用 MIPS（每秒百万条指令）来比较两台不同的处理器的性能，并认为 MIPS 最大的处理器具有最高的性能。这种说法正确吗？请用 P1 和 P2 验证这一说法。
- 1.12.4** [10] <1.10> 另一个常见的性能标志是 MFLOPS（每秒百万条浮点指令），其定义为  

$$\text{MFLOPS} = \text{浮点操作的数目} / (\text{执行时间} \times (1 \times 10^6))$$
 它与 MIPS 有同样的问题。假定 P1 和 P2 上执行的指令有 40% 的浮点指令，求出程序的 MFLOPS。
- 1.13** 1.10 节引证另一个易犯的错误是通过只改进计算机的一个方面来改进计算机的总体性能。假如一台计算机上运行一个程序需要 250s，其中 70s 用于执行浮点指令，85s 用于执行 L/S 指令，40s 用于执行分支指令。
- 58** **1.13.1** [5] <1.10> 如果浮点操作的时间减少 20%，总时间将减少多少？
- 1.13.2** [5] <1.10> 如果总时间减少 20%，整数操作时间将减少多少？
- 1.13.3** [5] <1.10> 如果只减少分支指令时间，总时间能否减少 20%？
- 1.14** 假定一个程序需要执行  $50 \times 10^6$  条浮点指令、 $110 \times 10^6$  条整数指令、 $80 \times 10^6$  条 L/S 指令和  $16 \times 10^6$  条分支指令。每种类型指令的 CPI 分别是 1、1、4 和 2。假定处理器的时钟频率为 2GHz。
- 1.14.1** [10] <1.10> 如果我们要将程序运行速度提高至 2 倍，浮点指令的 CPI 需如何改进？
- 1.14.2** [10] <1.10> 如果我们要将程序运行速度提高至 2 倍，L/S 指令的 CPI 需如何改进？

1.14.3 [5] <1.10> 如果整数和浮点指令的 CPI 减少 40%，L/S 和分支指令的 CPI 减少 30%，程序的执行时间能改进多少？

1.15 [5] <1.8> 多处理器系统中的执行时间可分成例程计算时间加处理器之间的通信时间。

假定一个程序在单处理器上执行时需要  $t = 100\text{s}$ 。当它在  $p$  个处理器上运行时，每个处理器需要  $t/p\text{s}$  的计算时间，另外还需要  $4\text{s}$  的开销，且开销与处理器数量无关。在处理器数目分别为 2、4、8、16、32、64 和 128 时，计算每个处理器的执行时间。在每种情况下，列出相对于单处理器的加速比和实际加速比与理想加速比的比值（理想加速比是指没有开销情况下的加速比）。

### 01 小测验答案

1.1 问题讨论：可以有多种答案。

1.3 DRAM 存储器：易失性，访问时间短（大约  $50 \sim 70\text{ns}$ ），每 GB 的价格（\$5 ~ \$10）。磁盘存储器：非易失性，访问时间比 DRAM 慢  $100\,000 \sim 400\,000$  倍，每 GB 的价格比 DRAM 便宜 100 倍。Flash 存储器：非易失性，访问时间比 DRAM 慢  $100 \sim 1\,000$  倍，每 GB 的价格比 DRAM 便宜 7 ~ 10 倍。

1.5 1、3、4 是正确答案，答案 5 一般可认为正确，因为产量高时能促使额外投资去减小芯片面积，例如减小 10%，这是一种经济决策，但并不总是正确。

1.6 1. 两者都改进，2. 延迟，3. 都不改进；7s。

1.6 b

1.10 a. 计算机 A 有较高的 MIPS 值；b. 计算机 B 更快。