

## 致谢

对于本书的每一个版本，我们都很幸运地得到了许多读者、审稿人和其他贡献者的帮助。这些人的帮助使这本书变得更好。

我们非常感谢 Khaled Benkrid 以及他在 ARM 公司的同事，他们非常认真地检查了 ARM 的相关材料，并提供了有益的反馈。

我们对第 6 章进行了大幅修订，对其中的观点和内容分别进行了审查，针对每一位审稿人的意见都进行了修改。感谢 Christos Kozyrakis (Stanford University)，是他建议我们使用集群系统的网络接口作为实例来说明 I/O 的软硬件接口划分，并对该章的内容组织给出了意见。感谢 Mario Flajsilk (Stanford University)，是他提供了和 NetFPGA NIC 平台相关的技术细节、图表和性能评估数据。感谢以下人员对第 6 章提出了修改建议：David Kaeli (Northeastern University)、Partha Ranganathan (HP Labs)、David Wood (University of Wisconsin)，以及我在 Berkeley 的同事 Siamak Faridani、Shoaib Kamil、Yunsup Lee、Zhangxi Tan 和 Andrew Waterman。

特别感谢 Rimas Avizenis (UC Berkeley)，是他改写了各种版本的矩阵乘法并提供了性能数据。当我还是 UCLA 的研究生时，我和他的父亲一起工作，能与 Rimas 在 UC Berkeley 共事是一件美好的事情。

同时，还要感谢我的长期合作者——Randy Katz (UC Berkeley)，他帮助我提出了“计算机体系结构中的重要思想”的概念，并将其作为我们一起合作开设的本科生课程的教学内容。

我还要感谢 David Kirk 和 John Nickolls 以及他们在 NVIDIA 的同事 (Michael Garland、John Montrym、Doug Voorhies、Lars Nyland、Erik Lindholm、Paulius Micikevicius、Massimiliano Fatica、Stuart Oberman 和 Vasily Volkov)，他们编写了第一个深入阐述 GPU 的附录。我希望能再次表达我对 Jim Larus 的感激之情，最近他刚被任命为 EPFL 的计算机和通信科学学院院长。感谢他愿意在汇编语言编程方面贡献自己的专业知识，并欢迎本书的读者使用他开发及维护的模拟器。

还要感谢 Zachary Kurmas (Grand Valley State University)，是他在原始版本的基础上更新并重新设计了新的章末练习。原始版本由 Perry Alexander (University of Kansas)、Jason Bakos (University of South Carolina)、Javier Bruguera (Universidade de Santiago de Compostela)、Matthew Farrens (University of California, Davis)、David Kaeli (Northeastern University)、Nicole Kaiyan (University of Adelaide)、John Oliver (Cal Poly, San Luis Obispo)、Milos Prvulovic (Georgia Tech)、Jichuan Chang (Google)、Jacob Leverich (Stanford)、Kevin Lim (Hewlett-Packard) 和 Partha Ranganathan (Google) 开发。

特别的感谢送给 Peter Ashenden，他帮我们更新了课程讲义。

感谢下面这些老师，他们回复了出版商的调查问卷，审阅了我们的提议，还参加了各种专题小组。专题小组人员如下：Bruce Barton (Suffolk County Community College)，Jeff Braun (Montana Tech)，Ed Gehringer (North Carolina State)，Michael Goldweber (Xavier University)，Ed Harcourt (St. Lawrence University)，Mark Hill (University of Wisconsin, Madison)，Patrick Homer (University of Arizona)，Norm Jouppi (HP Labs)，Dave Kaeli (Northeastern University)，Christos Kozyrakis (Stanford University)，Jae C. Oh (Syracuse University)，Lu Peng (LSU)，Milos Prvulovic (Georgia Tech)，Partha Ranganathan (HP Labs)，David Wood (University of Wisconsin)，Craig Zilles (University of Illinois at Urbana-

Champaign)。调查问卷和审阅人员如下：Mahmoud Abou-Nasr (Wayne State University), Perry Alexander (The University of Kansas), Behnam Arad (Sacramento State University), Hakan Aydin (George Mason University), Hussein Badr (State University of New York at Stony Brook), Mac Baker (Virginia Military Institute), Ron Barnes (George Mason University), Douglas Blough (Georgia Institute of Technology), Kevin Bolding (Seattle Pacific University), Miodrag Bolic (University of Ottawa), John Bonomo (Westminster College), Jeff Braun (Montana Tech), Tom Briggs (Shippensburg University), Mike Bright (Grove City College), Scott Burgess (Humboldt State University), Fazli Can (Bilkent University), Warren R. Carithers (Rochester Institute of Technology), Bruce Carlton (Mesa Community College), Nicholas Carter (University of Illinois at Urbana-Champaign), Anthony Cocchi (The City University of New York), Don Cooley (Utah State University), Gene Cooperman (Northeastern University), Robert D. Cupper (Allegheny College), Amy Csizmar Dalal (Carleton College), Daniel Dalle (Université de Sherbrooke), Edward W. Davis (North Carolina State University), Nathaniel J. Davis (Air Force Institute of Technology), Molisa Derk (Oklahoma City University), Andrea Di Blas (Stanford University), Derek Eager (University of Saskatchewan), Ata Elahi (Southern Connecticut State University), Ernest Ferguson (Northwest Missouri State University), Rhonda Kay Gaede (The University of Alabama), Etienne M. Gagnon (L'Université du Québec à Montréal), Costa Gerousis (Christopher Newport University), Paul Gillard (Memorial University of Newfoundland), Michael Goldweber (Xavier University), Georgia Grant (College of San Mateo), Paul V. Gratz (Texas A&M University), Merrill Hall (The Master's College), Tyson Hall (Southern Adventist University), Ed Harcourt (St. Lawrence University), Justin E. Harlow (University of South Florida), Paul F. Hemler (Hampden-Sydney College), Jayantha Herath (St. Cloud State University), Martin Herbordt (Boston University), Steve J. Hodges (Cabrillo College), Kenneth Hopkinson (Cornell University), Bill Hsu (San Francisco State University), Dalton Hunkins (St. Bonaventure University), Baback Izadi (State University of New York—New Paltz), Reza Jafari, Robert W. Johnson (Colorado Technical University), Bharat Joshi (University of North Carolina, Charlotte), Nagarajan Kandasamy (Drexel University), Rajiv Kapadia, Ryan Kastner (University of California, Santa Barbara), E.J. Kim (Texas A&M University), Jihong Kim (Seoul National University), Jim Kirk (Union University), Geoffrey S. Knauth (Lycoming College), Manish M. Kochhal (Wayne State), Suzan Koknar-Tezel (Saint Joseph's University), Angkul Kongmunvattana (Columbus State University), April Kontostathis (Ursinus College), Christos Kozyrakis (Stanford University), Danny Krizanc (Wesleyan University), Ashok Kumar, S. Kumar (The University of Texas), Zachary Kurmas (Grand Valley State University), Adrian Lauf (University of Louisville), Robert N. Lea (University of Houston), Alvin Lebeck (Duke University), Baoxin Li (Arizona State University), Li Liao (University of Delaware), Gary Livingston (University of Massachusetts), Michael Lyle, Douglas W. Lynn (Oregon Institute of Technology), Yashwant K Malaiya (Colorado State University), Stephen Mann (University of Waterloo), Bill Mark (University of Texas at Austin), Ananda Mondal (Claflin University), Alvin Moser (Seattle University), Walid Najjar (University of California, Riverside), Vijaykrishnan

Narayanan (Penn State University), Danial J. Neebel (Loras College), Victor Nelson (Auburn University), John Nestor (Lafayette College), Jae C. Oh (Syracuse University), Joe Oldham (Centre College), Timour Paltashev, James Parkerson (University of Arkansas), Shaunak Pawagi (SUNY at Stony Brook), Steve Pearce, Ted Pedersen (University of Minnesota), Lu Peng (Louisiana State University), Gregory D. Peterson (The University of Tennessee), William Pierce (Hood College), Milos Prvulovic (Georgia Tech), Partha Ranganathan (HP Labs), Dejan Raskovic (University of Alaska, Fairbanks) Brad Richards (University of Puget Sound), Roman Rozanov, Louis Rubinfield (Villanova University), Md Abdus Salam (Southern University), Augustine Samba (Kent State University), Robert Schaefer (Daniel Webster College), Carolyn J. C. Schauble (Colorado State University), Keith Schubert (CSU San Bernardino), William L. Schultz, Kelly Shaw (University of Richmond), Shahram Shirani (McMaster University), Scott Sigman (Drury University), Shai Simonson (Stonehill College), Bruce Smith, David Smith, Jeff W. Smith (University of Georgia, Athens), Mark Smotherman (Clemson University), Philip Snyder (Johns Hopkins University), Alex Sprintson (Texas A&M), Timothy D. Stanley (Brigham Young University), Dean Stevens (Morningside College), Nozar Tabrizi (Kettering University), Yuval Tamir (UCLA), Alexander Taubin (Boston University), Will Thacker (Winthrop University), Mithuna Thottethodi (Purdue University), Manghui Tu (Southern Utah University), Dean Tullsen (UC San Diego), Steve VanderLeest (Calvin College), Christopher Vickery (Queens College of CUNY), Rama Viswanathan (Beloit College), Ken Vollmar (Missouri State University), Guoping Wang (Indiana-Purdue University), Patricia Wenner (Bucknell University), Kent Wilken (University of California, Davis), David Wolfe (Gustavus Adolphus College), David Wood (University of Wisconsin, Madison), Ki Hwan Yum (University of Texas, San Antonio), Mohamed Zahran (City College of New York), Amr Zaky (Santa Clara University), Gerald D. Zarnett (Ryerson University), Nian Zhang (South Dakota School of Mines & Technology), Jiling Zhong (Troy University), Huiyang Zhou (North Carolina State University), Weiyu Zhu (Illinois Wesleyan University)。

特别感谢 Mark Smotherman，他通过多次检查发现了技术和写作方面的问题，这显著提高了本书的质量。

我们还要感谢 Morgan Kaufmann 这个大家庭同意在 Katey Birtcher、Steve Merken 和 Nate McFadden 的带领下再次出版这本书，没有他们我当然不可能完成。还要感谢负责图书制作过程的 Lisa Jones 以及负责封面设计的 Victoria Pearson Esser。本书封面巧妙地第 1 版封面的基础上融入了后 PC 时代的内容。

最后，我欠着 Yunsup Lee 和 Andrew Waterman 莫大的人情，是他们在创业之余，利用业余时间将本书改写成 RISC-V 版本。Eric Love 也是如此，他在完成博士学位的同时编辑了本书中 RISC-V 版本的练习。我们都很兴奋，想看看 RISC-V 在学术界和其他领域到底能做些什么。

以上共提到近 150 人，是他们帮助我们完成了新书，我希望这是迄今为止我们最好的一本书。谢谢各位！

David A. Patterson

### 戴维·A. 帕特森 (David A. Patterson)

自 1977 年从 UCLA 毕业加入加州大学伯克利分校开始执教起，一直在该校讲授计算机体系结构课程，是伯克利计算机科学 Pardee 讲座教授主席。他曾因教学工作获得加州大学杰出教学奖、ACM Karlstrom 奖、IEEE Mulligan 教育奖章以及 IEEE 本科教学奖。因为对 RISC 的贡献，Patterson 获得了 IEEE 技术进步奖和 ACM Eckert-Mauchly 奖，并因为对 RAID 的贡献分享了 IEEE Johnson 信息存储奖。他和 John Hennessy 共同获得了 IEEE John von Neumann 奖章以及 C&C 奖金。与 Hennessy 一样，Patterson 是美国艺术与科学院和计算机历史博物馆院士，ACM 和 IEEE 会士，并入选了美国国家工程院、国家科学院和硅谷工程名人堂。他曾服务于美国总统信息技术咨询委员会，担任过伯克利电子工程与计算机科学 (EECS) 系计算机科学分部主任、计算研究学会主席和 ACM 主席。这些工作使他获得了 ACM、CRA 以及 SIGARCH 的杰出服务奖。

在伯克利，Patterson 领导了 RISC I 的设计与实现工作，这可能是第一台 VLSI 精简指令集计算机，为商用 SPARC 体系结构奠定了基础。他也是廉价冗余磁盘阵列 (RAID) 项目的领导者，RAID 技术引导许多公司开发出了高可靠的存储系统。他还参加了工作站网络 (NOW) 项目，正是因为该项目，才有了被互联网公司广泛使用的集群技术以及后来的云计算。这些项目获得了四个 ACM 最佳论文奖。Patterson 当前的研究项目包括：“算法 - 机器 - 人” (AMP)；可证明最优实现的高效及弹性算法和专用器 (ASPIRE)。AMP 实验室的目标是开发可扩展的机器学习算法、适用于仓储式计算机的编程模型以及能够快速洞悉云中海量数据的众包工具。ASPIRE 实验室的目标是通过深度硬件和软件协同，在移动和机架计算系统中获得可能的最高性能和能效。

### 约翰·L. 亨尼斯 (John L. Hennessy)

斯坦福大学第十任校长，从 1977 年开始任教于该校电子工程与计算机科学系。Hennessy 是 IEEE 和 ACM 会士，美国国家工程院、国家科学院、美国哲学院以及美国艺术与科学院院士。Hennessy 获得的众多奖项包括：2001 年 ACM Eckert-Mauchly 奖（因对 RISC 的贡献），2001 年 Seymour Cray 计算机工程奖，2000 年与 Patterson 共同获得 John von Neumann 奖章。他还获得了七个荣誉博士学位。

1981 年，Hennessy 带领几位研究生在斯坦福大学开始研究 MIPS 项目。1984 年完成该项目后，他暂时离开大学，与他人共同创建了 MIPS Computer Systems 公司（现在的 MIPS Technologies 公司），该公司开发了最早的商用 RISC 微处理器之一。2006 年，已有超过 20 亿个 MIPS 微处理器应用在从视频游戏和掌上计算机到激光打印机和网络交换机的各类设备中。Hennessy 后来领导了共享存储器控制体系结构 (DASH) 项目，该项目设计了第一个可扩展 cache 一致性多处理器原型，其中的很多关键思想都在现代多处理器中得到了应用。除了参与科研活动和履行学校职责，Hennessy 还作为前期顾问和投资者参与了很多初创项目，为相关领域学术成果的商业化做出了杰出贡献。

出版者的话  
赞誉  
译者序  
前言  
作者简介

第 1 章 计算机抽象及相关技术 ..... 1

1.1 引言 ..... 1

1.1.1 传统的计算应用分类及其特点 ... 2

1.1.2 欢迎来到后 PC 时代 ..... 3

1.1.3 你能从本书中学到什么 ..... 4

1.2 计算机体系结构中的 8 个伟大思想 ... 6

1.2.1 面向摩尔定律的设计 ..... 6

1.2.2 使用抽象简化设计 ..... 7

1.2.3 加速经常性事件 ..... 7

1.2.4 通过并行提高性能 ..... 7

1.2.5 通过流水线提高性能 ..... 7

1.2.6 通过预测提高性能 ..... 7

1.2.7 存储层次 ..... 7

1.2.8 通过冗余提高可靠性 ..... 7

1.3 程序表象之下 ..... 8

1.4 箱盖后的硬件 ..... 10

1.4.1 显示器 ..... 11

1.4.2 触摸屏 ..... 12

1.4.3 打开机箱 ..... 13

1.4.4 数据安全 ..... 16

1.4.5 与其他计算机通信 ..... 16

1.5 处理器和存储制造技术 ..... 17

1.6 性能 ..... 20

1.6.1 性能的定义 ..... 21

1.6.2 性能的度量 ..... 23

1.6.3 CPU 性能及其度量因素 ..... 24

1.6.4 指令性能 ..... 25

1.6.5 经典的 CPU 性能公式 ..... 26

1.7 功耗墙 ..... 28

1.8 沧海巨变：从单处理器向多处理器  
转变 ..... 30


1.9 实例：评测 Intel Core i7 ..... 32

1.9.1 SPEC CPU 基准评测程序 ..... 33

1.9.2 SPEC 功耗基准评测程序 ..... 34

1.10 谬误与陷阱 ..... 35

1.11 本章小结 ..... 37

 1.12 历史视角和拓展阅读 ..... 38

1.13 练习 ..... 38

第 2 章 指令：计算机的语言 ..... 43

2.1 引言 ..... 43

2.2 计算机硬件的操作 ..... 45

2.3 计算机硬件的操作数 ..... 47

2.3.1 存储器操作数 ..... 48

2.3.2 常数或立即数操作数 ..... 51

2.4 有符号数与无符号数 ..... 52

2.5 计算机中的指令表示 ..... 57

2.6 逻辑操作 ..... 62

2.7 用于决策的指令 ..... 65

2.7.1 循环 ..... 66

2.7.2 边界检查的简便方法 ..... 67

2.7.3 case/switch 语句 ..... 68

2.8 计算机硬件对过程的支持 ..... 68

2.8.1 使用更多的寄存器 ..... 69

2.8.2 嵌套过程 ..... 71

2.8.3 在栈中为新数据分配空间 ..... 73

2.8.4 在堆中为新数据分配空间 ..... 74

2.9 人机交互 ..... 76

2.10 对大立即数的 RISC-V 编址和  
寻址 ..... 79




2.10.1 大立即数 ..... 79



2.10.2 分支中的寻址 ..... 80

2.10.3 RISC-V 寻址模式总结 ..... 82

2.10.4 机器语言译码 ..... 83



2.11	指令与并行性：同步	85	3.3.5	总结	128
2.12	翻译并启动程序	87	3.4	除法	128
2.12.1	编译器	87	3.4.1	除法算法及其硬件实现	128
2.12.2	汇编器	87	3.4.2	有符号除法	131
2.12.3	链接器	89	3.4.3	快速除法	131
2.12.4	加载器	91	3.4.4	RISC-V 中的除法	132
2.12.5	动态链接库	91	3.4.5	总结	132
2.12.6	启动 Java 程序	93	3.5	浮点运算	133
2.13	以 C 排序程序为例的汇总整理	94	3.5.1	浮点表示	134
2.13.1	swap 过程	94	3.5.2	例外和中断	135
2.13.2	sort 过程	95	3.5.3	IEEE 754 浮点数标准	135
2.14	数组与指针	100	3.5.4	浮点加法	138
2.14.1	用数组实现 clear	100	3.5.5	浮点乘法	141
2.14.2	用指针实现 clear	101	3.5.6	RISC-V 中的浮点指令	144
2.14.3	比较两个版本的 clear	102	3.5.7	精确算术	148
 2.15	高级专题：编译 C 语言和解释 Java 语言	102	3.5.8	总结	150
2.16	实例：MIPS 指令	103	3.6	并行性与计算机算术：子字并行	151
2.17	实例：x86 指令	104	3.7	实例：x86 中的 SIMD 扩展和 高级向量扩展	151
2.17.1	Intel x86 的演变	104	3.8	加速：子字并行和矩阵乘法	153
2.17.2	x86 寄存器和寻址模式	106	3.9	谬误与陷阱	155
2.17.3	x86 整数操作	107	3.10	本章小结	158
2.17.4	x86 指令编码	109	 3.11	历史视角和拓展阅读	159
2.17.5	x86 总结	110	3.12	练习	159
2.18	实例：RISC-V 指令系统的剩余 部分	111	第 4 章	处理器	163
2.19	谬误与陷阱	112	4.1	引言	163
2.20	本章小结	113	4.1.1	一种基本的 RISC-V 实现	164
 2.21	历史视角和扩展阅读	115	4.1.2	实现概述	164
2.22	练习	115	4.2	逻辑设计的一般方法	166
第 3 章	计算机的算术运算	121	4.3	建立数据通路	169
3.1	引言	121	4.4	一个简单的实现方案	175
3.2	加法和减法	121	4.4.1	ALU 控制	175
3.3	乘法	124	4.4.2	设计主控制单元	176
3.3.1	串行版的乘法算法及其硬件 实现	124	4.4.3	数据通路操作	180
3.3.2	带符号乘法	127	4.4.4	控制的结束	182
3.3.3	快速乘法	127	4.4.5	为什么现在不使用单周期 实现	182
3.3.4	RISC-V 中的乘法	127	4.5	流水线概述	183
			4.5.1	面向流水线的指令系统设计	187

4.5.2	流水线冒险	187	5.2.3	闪存	264
4.5.3	总结	193	5.2.4	磁盘	264
4.6	流水线数据通路和控制	194	5.3	cache 基础	266
4.6.1	流水线的图形化表示	203	5.3.1	cache 访问	268
4.6.2	流水线控制	205	5.3.2	处理 cache 失效	272
4.7	数据冒险：前递与停顿	208	5.3.3	处理写操作	273
4.8	控制冒险	218	5.3.4	cache 实例：Intrinsity FastMATH 处理器	275
4.8.1	假设分支不发生	218	5.3.5	总结	276
4.8.2	缩短分支延迟	219	5.4	cache 的性能评估和改进	277
4.8.3	动态分支预测	221	5.4.1	使用更为灵活的替换策略降低 cache 失效率	279
4.8.4	流水线总结	223	5.4.2	在 cache 中查找数据块	283
4.9	例外	223	5.4.3	选择替换的数据块	284
4.9.1	RISC-V 体系结构中如何处理 例外	224	5.4.4	使用多级 cache 减少失效 代价	285
4.9.2	流水线实现中的例外	225	5.4.5	通过分块进行软件优化	287
4.10	指令间的并行性	228	5.4.6	总结	291
4.10.1	推测的概念	229	5.5	可靠的存储器层次	291
4.10.2	静态多发射	230	5.5.1	失效的定义	291
4.10.3	动态多发射处理器	234	5.5.2	纠正 1 位错、检测 2 位错的 汉明编码	293
4.10.4	高级流水线和能效	237	5.6	虚拟机	296
4.11	实例：ARM Cortex-A53 和 Intel Core i7 流水线结构	238	5.6.1	虚拟机监视器的必备条件	297
4.11.1	ARM Cortex-A53	238	5.6.2	指令系统体系结构（缺乏） 对虚拟机的支持	297
4.11.2	Intel Core i7 920	240	5.6.3	保护和指令系统体系结构	298
4.11.3	Intel Core i7 处理器的 性能	242	5.7	虚拟存储	298
4.12	加速：指令级并行和矩阵乘法	243	5.7.1	页的存放和查找	301
 4.13	高级专题：数字设计概述——使用 硬件设计语言进行流水线建模以及 更多流水线示例	246	5.7.2	缺页失效	303
4.14	谬误与陷阱	246	5.7.3	支持大虚拟地址空间的虚拟 存储	304
4.15	本章小结	247	5.7.4	关于写	305
 4.16	历史视角和拓展阅读	247	5.7.5	加快地址转换：TLB	306
4.17	练习	247	5.7.6	Intrinsity FastMATH TLB	307
5.7.7	集成虚拟存储、TLB 和 cache	309	5.7.8	虚拟存储中的保护	311
5.7.9	处理 TLB 失效和缺页失效	312	5.7.10	总结	314
第 5 章	大而快：层次化存储	258			
5.1	引言	258			
5.2	存储技术	262			
5.2.1	SRAM 存储技术	262			
5.2.2	DRAM 存储技术	262			

5.8	存储层次结构的一般框架 .....	315	6.3.1	x86 中的 SIMD: 多媒体扩展 .....	355
5.8.1	问题一: 块可以被放在何处 .....	315	6.3.2	向量机 .....	355
5.8.2	问题二: 如何找到块 .....	316	6.3.3	向量与标量 .....	356
5.8.3	问题三: 当 cache 发生失效时 替换哪一块 .....	317	6.3.4	向量与多媒体扩展 .....	357
5.8.4	问题四: 写操作如何处理 .....	317	6.4	硬件多线程 .....	359
5.8.5	3C: 一种理解存储层次结构的 直观模型 .....	318	6.5	多核及其他共享内存多处理器 .....	362
5.9	使用有限状态自动机控制简单的 cache .....	320	6.6	GPU 简介 .....	365
5.9.1	一个简单的 cache .....	320	6.6.1	NVIDIA GPU 体系结构简介 .....	366
5.9.2	有限状态自动机 .....	321	6.6.2	NVIDIA GPU 存储结构 .....	367
5.9.3	使用有限状态自动机作为 简单的 cache 控制器 .....	322	6.6.3	对 GPU 的展望 .....	368
5.10	并行和存储层次结构: cache 一致性 .....	324	6.7	集群、仓储级计算机和其他消息 传递多处理器 .....	370
5.10.1	实现一致性的基本方案 .....	325	6.8	多处理器网络拓扑简介 .....	374
5.10.2	监听协议 .....	325	6.9	与外界通信: 集群网络 .....	376
5.11	并行与存储层次结构: 廉价磁盘 冗余阵列 .....	327	6.10	多处理器测试基准和性能模型 .....	377
5.12	高级专题: 实现缓存控制器 .....	327	6.10.1	性能模型 .....	379
5.13	实例: ARM Cortex-A53 和 Intel Core i7 的存储层次结构 .....	327	6.10.2	Roofline 模型 .....	380
5.14	实例: RISC-V 系统的其他部分和 特殊指令 .....	331	6.10.3	两代 Opteron 的比较 .....	381
5.15	加速: cache 分块和矩阵乘法 .....	331	6.11	实例: 评测 Intel Core i7 960 和 NVIDIA Tesla GPU 的 Roofline 模型 .....	384
5.16	谬误与陷阱 .....	333	6.12	加速: 多处理器和矩阵乘法 .....	388
5.17	本章小结 .....	336	6.13	谬误与陷阱 .....	390
5.18	历史视角和拓展阅读 .....	337	6.14	本章小结 .....	391
5.19	练习 .....	337	6.15	历史视角和拓展阅读 .....	393
			6.16	练习 .....	394
第 6 章	并行处理器: 从客户端 到云 .....	348	附录 A	逻辑设计基础 .....	402
6.1	引言 .....	348	术语表	.....	460
6.2	创建并行处理程序的难点 .....	350	网络内容 <sup>⊖</sup>		
6.3	SISD、MIMD、SIMD、SPMD 和向 量机 .....	354	附录 B	图形处理单元	
			附录 C	将控制映射至硬件	
			附录 D	精简指令系统体系结构计算机	
			扩展阅读		

⊖ 网络内容请访问原书配套网站 [booksite.elsevier.com/9780128122754](http://booksite.elsevier.com/9780128122754) 下载。——编辑注



# 计算机抽象及相关技术

## 1.1 引言

欢迎阅读本书！我们很高兴有机会来分享令人兴奋的计算机系统世界。这绝不是一个枯燥乏味、进步缓慢、新思想在忽视中凋零的领域。相反！计算机是难以置信而充满活力的信息技术产业的产物，其各类相关产品约占美国国民生产总值的 10%。美国经济在某些方面已经与信息技术密不可分，而这一领域正按照摩尔定律所预示的那样快速发展。这个不同寻常的行业在以惊人的速度拥抱创新。在过去的 30 年中，已经出现了许多新型计算机，这些新型计算机的引入导致了计算产业的革命，而它们很快又被其他人所建造的更好的计算机所取代。

人类文明因为那些我们不假思索即可完成的要务数量的增加而进步。

*Alfred North Whitehead,*  
数学导论, 1911

自从 20 世纪 40 年代末电子计算机诞生以来，这场创新竞争已经带来了史无前例的进步。例如，如果运输行业能够和计算机行业保持同样的发展速度，那么如今我们花一分钱就可以在一秒钟内从纽约赶到伦敦。稍微思考一下这样的进步会如何改变社会——居住在南太平洋的塔希提岛，而工作在旧金山，晚上去莫斯科参加波修瓦芭蕾舞团的演出——你就能理解这种变革的意义了。

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。由此产生的人类智力的成倍增长自然而深刻地影响了人们的日常生活，甚至改变了人们寻求新知识的方式。现在有了一个新的科学探索方式，即计算科学家加入了理论与实验科学家的行列，共同探索天文学、生物学、化学和物理学等领域的前沿问题。

计算机革命仍在继续向前推进。每当计算成本降低为原来的十分之一，计算机的发展机遇就会成倍增长。原本出于经济因素而不可行的应用突然变得切实可行。例如在不久之前，下述各项应用还曾经只是“计算机科学幻想”。

- 车载计算机：直到 20 世纪 80 年代初微处理器的价格和性能得到显著改善之前，用计算机来控制汽车几乎是天方夜谭。如今，计算机可以通过控制汽车发动机降低污染、提高燃油效率，还能通过盲点警示、车道偏离警示、移动目标检测和气囊保护实现碰撞时对乘客的保护，从而增加汽车的安全性。
- 手机：谁曾预想到计算机系统的发展会导致全球半数以上的人口拥有移动电话，并让人们几乎可以与世界上任何地方的人进行交流？
- 人类基因组项目：以前用以匹配和分析人类 DNA 序列的计算机设备成本高达数亿美元。15 ~ 25 年前，这个项目的计算机成本要高出 10~100 倍，那时，几乎没有人会考虑加入这个项目。目前，该项目的计算机成本仍然在持续下降，人们将很快就能够获得自己的基因组，从而量身定制医疗服务。
- 万维网：在本书第 1 版出版时，万维网还不存在，而现在万维网已经改变了我们的社会。对于很多人来说，网络已经取代了图书馆和报纸。

- 搜索引擎：随着万维网内容的规模 and 价值的增长，如何找到相关信息变得越来越重要。如今，很多人的生活都很大程度地依赖搜索引擎，如果没有搜索引擎，生活可能会变得举步维艰。

显然，计算机技术的进步几乎影响着社会的方方面面。硬件的进步使得程序员能够创造出奇妙而有用的软件，进而证明了为什么计算机无所不在。现在的科学幻想往往预示着未来最具影响力的应用，例如，虚拟现实眼镜、无现金社会以及自动驾驶汽车都即将到来。

### 1.1.1 传统的计算应用分类及其特点

从智能家电到手机再到最大的超级计算机，虽然在计算机中使用了一套通用的硬件技术（见 1.4 节和 1.5 节），但不同的应用具有不同的设计要求，并以不同的方式使用核心硬件技术。宽泛地说，计算机主要被用于如下三种不同的应用场景中。

个人计算机（Personal Computer, PC）可能是最广为人知的计算机形式，本书的读者几乎都在广泛使用。个人计算机强调以低成本向单个用户交付良好的性能，通常运行第三方软件。这类计算方式推动了许多计算技术的发展，尽管它仅有 35 年的历史！

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。

服务器是过去曾是庞然大物的计算机的现代形式，通常只能通过网络访问。服务器适用于执行巨大的工作负载，其中可能包括单个复杂应用——通常是科学或工程应用程序；也可以执行许多小型作业，例如在构建大型 Web 服务器时发生的任务。这些应用程序通常来自其他来源（如数据库或模拟系统）的软件，但往往会针对特定需求进行修改或定制。服务器采用与桌面计算机相同的基础技术构建，但能够提供更强大的计算、存储和输入/输出能力。通常情况下，服务器更强调可靠性，因为相比单用户个人计算机而言，服务器发生故障的代价更高。

服务器：用于为多个用户并行运行大型程序的计算机，通常只能通过网络访问。

服务器的成本和功能范围跨度极其广泛。低端的服务器可能比一台不带屏幕和键盘的桌面计算机稍贵一些，大约需要一千美元。此类低端服务器通常用于文件存储、小型商业应用或简单的 Web 服务。高端的服务器则是超级计算机（supercomputer），当前的超级计算机一般由成千上万颗处理器和数太字节（terabyte）的内存组成，且成本高达几千万甚至数亿美元。超级计算机通常用于高端科学和工程计算，例如天气预报、石油勘探、蛋白质结构测定和其他大规模问题。尽管这样的超级计算机代表了最高的计算能力，但它们只占据了服务器中相对较小的一部分，在整个计算机市场中所占总销售收入的比例也很小。

超级计算机：具有最高的性能和成本的一类计算机，一般被配置为服务器且通常耗费数千万美元甚至数亿美元。

嵌入式计算机是计算机中最大的一个类别，其应用场景和性能范围也最为广泛。嵌入式计算机包括汽车、电视机中的微处理器或计算机，以及控制飞机或货船的处理器网络。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。因此，尽管嵌入式计算机的数量庞大，但仍然有大多数用户从来没有意识到他们正在使

太字节：原始的定义为  $1\,099\,511\,627\,776\,(2^{40})$  字节，而通信和辅助存储系统研发人员用其表示  $1\,000\,000\,000\,000\,(10^{12})$  字节。为了减少混淆，本书使用太比字节（tebibyte, TiB）这个术语来表示  $2^{40}$  字节，而使用太字节（terabyte, TB）表示  $10^{12}$  字节。图 1-1 展现了十进制和二进制数值和名称的全部范围。

嵌入式计算机：用于运行某预定应用程序或软件集合的计算机，一般内嵌于其他设备中。

用计算机。

十进制			二进制			增长百分比
术语	缩写	数值	术语	缩写	数值	
千字节	KB	10 <sup>3</sup>	千比字节	KiB	2 <sup>10</sup>	2%
兆字节	MB	10 <sup>6</sup>	兆比字节	MiB	2 <sup>20</sup>	5%
吉字节	GB	10 <sup>9</sup>	吉比字节	GiB	2 <sup>30</sup>	7%
太字节	TB	10 <sup>12</sup>	太比字节	TiB	2 <sup>40</sup>	10%
拍字节	PB	10 <sup>15</sup>	拍比字节	PiB	2 <sup>50</sup>	13%
艾字节	EB	10 <sup>18</sup>	艾比字节	EiB	2 <sup>60</sup>	15%
泽字节	ZB	10 <sup>21</sup>	泽比字节	ZiB	2 <sup>70</sup>	18%
尧字节	YB	10 <sup>24</sup>	尧比字节	YiB	2 <sup>80</sup>	21%

图 1-1 通过为所有常见规格术语添加二进制记法解决了 2<sup>x</sup> 与 10<sup>y</sup> 字节的不明确性。在最后一列中，我们标记了二进制术语比相应的十进制术语大出多少的具体比例，可以看到该数值从上往下逐渐增大。这些前缀可以用于位和字节，所以吉位（Gb）为 10<sup>9</sup> 位，而吉比位（Gib）为 2<sup>30</sup> 位

嵌入式应用常常具有特定的应用程序要求，这需要将最低性能与严格的成本及功耗限制结合在一起考虑。以音乐播放器为例，处理器只需要尽快执行有限的功能，除此之外，成本和功耗最小化是最重要的目标。尽管成本很低，但是嵌入式计算机通常对故障的容忍度较低，因为故障可能会令人不安（例如，新买的电视无法正常工作），甚至是破坏性的（例如，飞机或货船上的计算机系统崩溃）。在面向消费者的嵌入式应用（如数字家电）中，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，采用了服务器领域的冗余技术。尽管本书的重点是通用计算机，但大多数概念直接或稍作修改即可适用于嵌入式计算机。

**详细阐述** “详细阐述”是正文中的一些简短段落，提供读者可能感兴趣的特定主题的更多详细信息。因为后续材料并不依赖于详细阐述内容，所以对此不感兴趣的读者可以直接跳过。

许多嵌入式处理器都是使用处理器核进行设计，处理器核是使用 Verilog 或 VHDL（参见第 4 章）等硬件描述语言编写的处理器版本。它使得设计人员可将其他专用硬件与处理器核集成在一块单芯片上进行制造。

### 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代际变迁，也改变了整个信息技术产业。从本书的上一版以来，我们经历了这种变化，这种变化就好像 30 年前个人电脑所带来的影响一样重大。替代个人电脑的是个人移动设备（Personal Mobile Device, PMD）。个人移动设备利用电池供电，通过无线方式连接到互联网，价格通常只有几百美元。此外，和个人电脑一样，用户可以下载软件（“应用程序”）在其上运行。与个人电脑不同的是，个人移动设备不再拥有键盘和鼠标，更可能依靠触摸屏或语音作为输入。当前的个人移动设备是智能手机或平板电脑，但未来的个人移动设备可能包括电子眼镜。图 1-2 给出了平板电脑及智能手机与个人计算机及传统手机之数量随时间快速增长情况的对比。

个人移动设备：连接到互联网的小型无线设备；它们依靠电池供电，并通过下载 App 的方式来安装软件。常见的例子有智能手机和平板电脑。

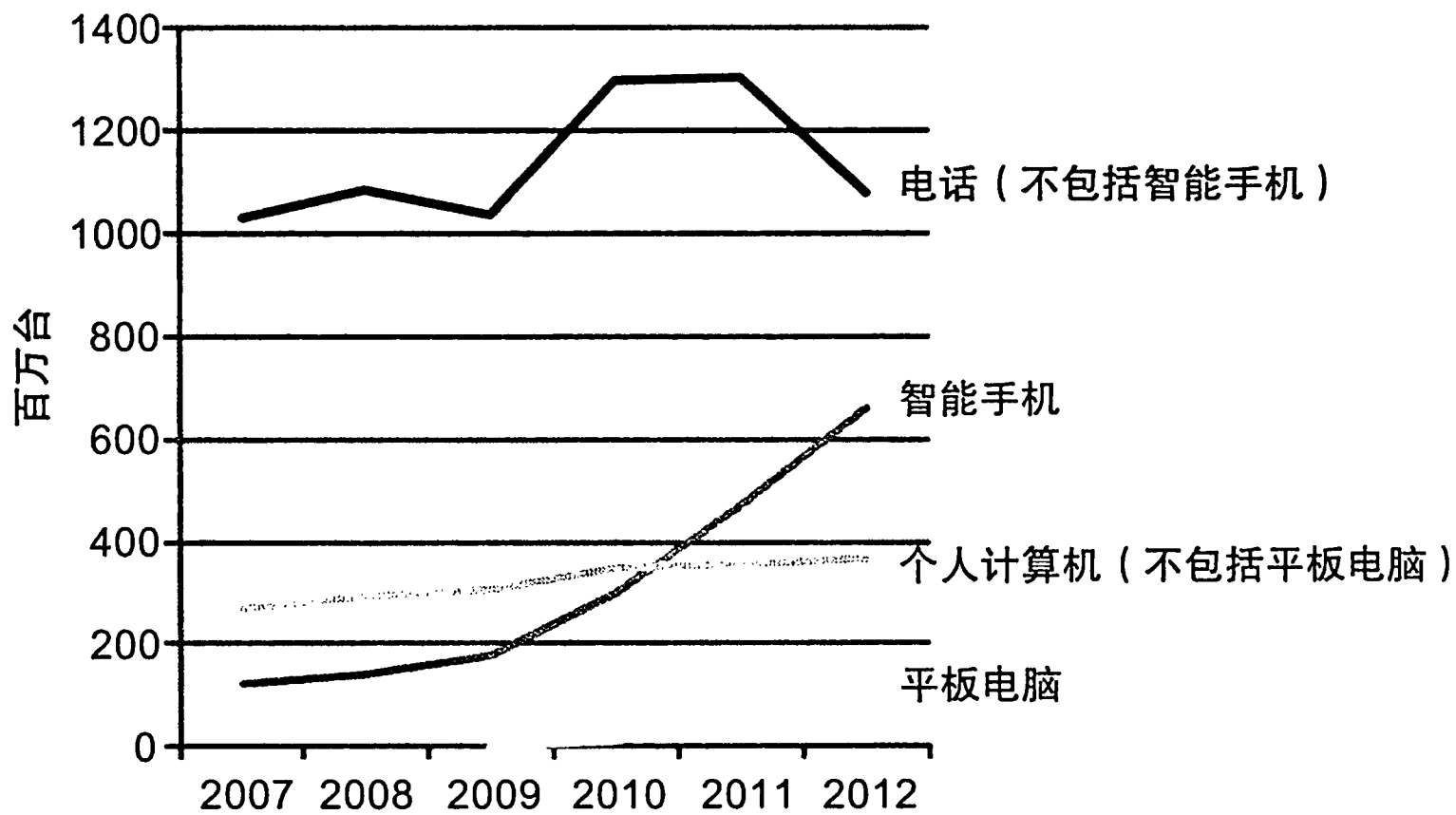


图 1-2 代表着后 PC 时代的平板电脑及智能手机每年的产量与个人计算机及传统手机的对比。智能手机反映了手机行业近期的增长情况，并于 2011 年超过了个人计算机产量。平板电脑是增长最快的计算机类别，在 2011 年至 2012 年间几乎成倍增长。个人计算机和传统电话的产量保持不变甚至下降

云计算接替了传统服务器，它依赖于现在称为仓储级计算机 (Warehouse Scale Computer, WSC) 的巨型数据中心。像亚马逊和谷歌这样的公司构建了包含 100 000 台服务器的仓储级计算机，一些公司租用其中的一部分为个人移动设备提供软件服务而无须建立自己的仓储级计算机。事实上，正如个人移动设备和仓储级计算机正在改变硬件行业一样，通过云计算实现的软件即服务 (SaaS) 正在彻底改变软件行业。当今的软件开发人员经常会将应用的一部分运行在个人移动设备上，另一部分则部署在云上。

**云计算：**通过互联网提供服务的大规模服务器集群，一些服务提供商动态地将不同数量的服务器作为像水、电一样的公用资源进行租用。

**软件即服务：**通过互联网以服务的方式提供软件和数据，通常是通过一个小型客户端程序（例如运行在本地客户端设备上的浏览器）连接网络以运行程序或获取数据，而不是必须完全在本地设备上安装和运行所有二进制代码。具体的例子包括 Web 搜索和社交网络。

### 1.1.3 你能从本书中学到什么

成功的程序员总是关注程序的性能，因为让用户快速获得结果对于软件的成功而言至关重要。20 世纪六七十年代，限制计算机性能的主要因素是计算机的内存容量。因此，当时的程序员经常遵循一个简单的信条：尽量减少程序占用的内存空间以加速程序运行。近十年以来，计算机设计和存储器技术有了显著进步，除了嵌入式计算系统以外，大多数应用系统中内存容量对计算机性能的影响已大大降低了。

现在，关心性能的程序员应该非常明确，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存储的层次性。我们将在第 3 章至第 6 章中展示如何将 C 程序的性能提高 200 倍，从而阐释这一理解的重要性。此外，正如我们在 1.7 节中所阐述的，当今的程序员需要考虑在个人移动设备或云上程序运行的能效，这就要求他们了解代码之下的诸多细节。因此，程序员为了创造有竞争力的软件，必须加强对计算机组成的认知。

我们很荣幸有机会解释这个革命性的计算机器中的内容，并阐明程序之下的软件以及机箱覆盖下的硬件是如何工作的。在你读完这本书的时候，我们相信你能够回答以下问题：

- 用 C 或 Java 等高级语言编写的程序如何被翻译成机器语言，以及硬件如何执行最终



的程序？这些概念是理解软硬件如何影响程序性能的基础。

- 软件和硬件之间的接口是什么？软件如何指导硬件执行所需的功能？这些概念对于理解如何编写软件是至关重要的。
- 什么因素决定了程序的性能，以及程序员如何改进程序性能？我们将从本书知道，这取决于原始程序、将该程序转换成计算机语言的软件以及硬件执行该程序的有效性。
- 硬件设计人员可以使用哪些技术来提高性能？本书将介绍现代计算机设计的基本概念。感兴趣的读者可以在我们的进阶教材《计算机体系结构：量化研究方法》中找到更多关于此主题的内容。
- 硬件设计人员可以使用哪些技术来改善能效？程序员可以做些什么来改变能效？
- 串行处理近来发展到并行处理的原因和结果是什么？本书给出了这一发展变化的动机，描述了当前支持并行的硬件机制，并评述了新一代“多核”微处理器（见第 6 章）。
- 自 1951 年第一台商用计算机以来，计算机架构师提出的哪些伟大思想奠定了现代计算技术的基础？

多核微处理器：在单个集成电路中包含多个处理器（“核”）的微处理器。

如果不了解这些问题的答案，那么要在现代计算机上改进程序性能，或者要评估不同计算机解决特定问题的优劣，都将是一个复杂的反复试验过程而非一个深入分析的科学过程。

第 1 章为本书的其余章节奠定基础。它介绍了基本概念和定义，对软件和硬件的主要组成部分进行了剖析，展示了如何评估性能和功耗，介绍了集成电路（推动计算机革命的技术），并在最后解释了技术向多核转变的原因。

在本章及后面的章节中，读者可能会看到很多新的术语，或者可能听到过但不确定确切含义的术语。请不要惊慌或担心！在描述现代计算机时，确实会使用许多专用术语来作为帮助，它使我们能够精确地描述计算机的功能或性能。此外，计算机设计师（包括本书作者）喜欢使用首字母缩略词，一旦知道这些字母代表的是什麼，就很容易理解了。为了帮助读者记住和理解这些术语，在术语第一次出现的时候，本书将给出明确定义。经过与术语的短间接接触，你将会熟练掌握它们，而且你的朋友也将对你正确使用 BIOS、CPU、DIMM、DRAM、PCIe、SATA 等许多缩略词印象深刻。

首字母缩略词：通过提取一串单词的首字母构成的单词。例如：RAM 是随机存取存储单元（Random Access Memory）的首字母缩略词，CPU 是中央处理单元（Central Processing Unit）的首字母缩略词。

为了加强对程序运行时如何应用软件和硬件以影响性能的理解，我们安排了贯穿整书的专门模块“理解程序性能”，概括对程序性能的重要理解。下面就是第一个。

**| 理解程序性能** 程序的性能取决于以下各因素的组合：程序中所用算法的有效性，用来创建程序和将其翻译为机器指令的软件系统，计算机执行这些机器指令（可能包括输入 / 输出操作）时的有效性。下表总结了软件和硬件是如何影响程序性能的。

软件或硬件组成部分	该部分如何影响性能	该主题出现的位置
算法	决定了源码级语句的数量和执行 I/O 操作的数量	其他书籍
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

如上所述，为了说明本书中思想的作用，我们在一系列章节中对一个完成矩阵和向量相



乘的 C 程序的性能进行优化。每个步骤都可以帮助我们理解现代微处理器中的底层硬件是如何将性能提高 200 倍的！

- 在第 3 章的数据级并行部分中，我们使用 C 语言固有的子字并行将性能提高了 3.8 倍。
- 在第 4 章的指令级并行部分中，我们使用循环展开来开发多指令发射和乱序执行硬件，将性能再提高 2.3 倍。
- 在第 5 章的存储层次优化部分中，我们使用高速缓存分块技术将大型矩阵性能再次提高 2 到 2.5 倍。
- 在第 6 章的线程级并行部分，我们在 OpenMP 中使用循环并行来开发多核硬件，将性能再次提高 4 到 14 倍。

#### Self-Check

**自我检测** “自我检测”部分旨在帮助读者评估自己是否理解一章中介绍的主要概念并理解这些概念的含义。其中一些问题的答案比较简单，另外一些问题则适用于小组讨论。在本章最后可以找到一些具体问题的答案。“自我检测”只出现在章节末，如果你确定自己完全理解了这部分内容，则可以跳过该部分。

1. 每年嵌入式处理器的销售数量远远超过 PC 处理器甚至后 PC 处理器的数量。根据自己的经验，你支持还是反对这种看法？尝试列举你家中的嵌入式处理器。它与你家中传统计算机的数量相比如何？
2. 如前所述，软件和硬件都会影响程序的性能。请思考以下哪个例子属于性能瓶颈。
  - 所选算法
  - 编程语言或编译器
  - 操作系统
  - 处理器
  - I/O 系统和设备

## 1.2 计算机体系结构中的 8 个伟大思想

现在我们来介绍计算机架构师在过去 60 年中提出的 8 个伟大思想。这些思想非常强大，以至于在应用这些思想产生首台计算机之后的很长时间内，新一代的架构师仍然在设计中通过模仿向先驱致敬。这些伟大的思想将贯穿在本章和后续章节的主题中，明确使用了近 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所集成的晶体管资源每 18 至 24 个月翻一番。摩尔定律是 Intel 公司的创始人之一戈登·摩尔在 1965 年对集成电路容量做出的预测。由于计算机设计需要花费数年时间，因此在项目结束时，每个芯片的可用晶体管资源相对设计开始时可以轻易实现双倍或四倍增长。像双向飞碟运动员一样，计算机架构师必须预测其设计完成时的工艺水平，而不是设计开始时的工艺水平。

## 1.2.2 使用抽象简化设计

计算机架构师和程序员都必须发明新技术来提高自己的工作效率，否则根据摩尔定律，设计时间也会随着资源的增长而显著延长。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——隐藏低层细节以提供给高层一个更简单的模型。

## 1.2.3 加速经常性事件

加速经常性事件（make the common case fast）远比优化罕见情形能够更好地提升性能。具有讽刺意味的是，经常性事件往往比罕见情形更简单，因此通常更容易提升。这个常识性建议意味着设计者需要知道经常性事件是什么，这只有通过仔细的实验和测量才可能得出（见 1.6 节）。

## 1.2.4 通过并行提高性能

自计算诞生以来，计算机架构师就通过并行计算操作来获得更高性能。在本书中我们将会看到很多并行的例子。

## 1.2.5 通过流水线提高性能

并行性的一种特殊场景在计算机体系结构中非常普遍，因此它有着专有名称：流水线（pipelining）。例如在许多西部片中有坏人纵火，而在消防车出现之前可能会有一个“消防队列”来灭火——小镇居民们排成一个长链来运水灭火，这可以使水桶在链上快速移动而无须人员往返奔跑。

## 1.2.6 通过预测提高性能

遵照谚语“请求宽恕胜于寻求许可”，下一个伟大思想就是预测。在某些情况下，假设从预测错误中恢复的代价并不高，且预测相对准确，则平均来说进行预测并开始工作可能会比等到明确结果后再执行更快。

## 1.2.7 存储层次

由于存储器的速度通常会影响到性能，存储器的容量限制了可被解决的问题的规模，且当今的内存成本常常是计算机成本的主要部分，因此程序员希望存储器速度更快、容量更大、价格更便宜。架构师发现可以通过存储层次（hierarchy of memory）来处理这些冲突的需求。在存储层次中，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底层。正如我们将在第 5 章中看到的那样，高速缓存给了程序员这样的错觉：主存与存储层次顶层几乎一样快，且与存储层次底层拥有几乎一样大的容量和便宜的价格。

## 1.2.8 通过冗余提高可靠性

计算机不仅要速度快，更需要工作可靠。由于任何物理设备都可能发生故障，因此我们通过引入冗余组件来使系统可靠，该组件在系统发生故障时可以替代失效组件并帮助检测故障。