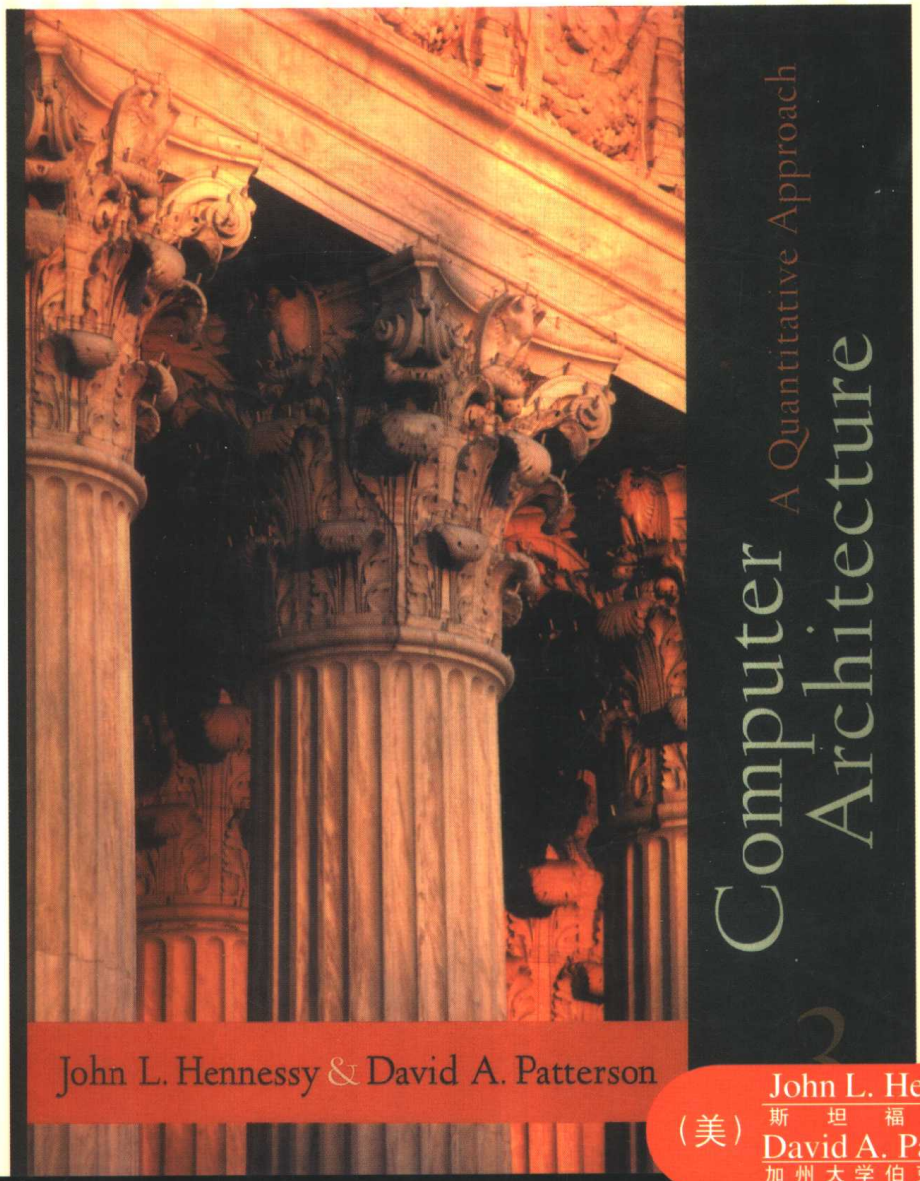
经 典 原 版 书 库

计算机体系结构
量化研究方法

（英文版 · 第3版）



（美）John L. Hennessy 斯坦福大学
David A. Patterson 加州大学伯克利分校 著

机械工业出版社
China Machine Press

经 典 原 版 书 库

计算机体系结构
量化研究方法

(英文版 · 第3版)

Computer Architecture
A Quantitative Approach

(Third Edition)

(美) John L. Hennessy
斯 坦 福 大 学
David A. Patterson
加州大学伯克利分校
著

机械工业出版社
China Machine Press

John L. Hennessy, David A. Patterson: Computer Architecture: A Quantitative Approach, Third Edition.

ISBN: 1-55860-596-7.

本书版权登记号：图字：01-2002-3582

图书在版编目（CIP）数据

计算机体系结构：量化研究方法：第3版/（美）亨尼西（Hennessy, J. L.），（美）帕特森（Patterson, D. A.）著. -北京：机械工业出版社，2002.9

（经典原版书库）

书名原文：Computer Architecture: A Quantitative Approach

ISBN 7-111-10921-X

Ⅰ. 计… Ⅱ. ① 亨… ② 帕… Ⅲ. 计算机体系结构-英文 Ⅳ. TP303

中国版本图书馆CIP数据核字（2002）第068889号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：华 章

北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行

2002年9月第1版 · 2003年1月第2次印刷

787mm×1092mm 1/16 · 71印张

印数：3 001-5 000册

定价：99.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：针对本科生的核心课程，剔抉外版菁华而成“国外经典教材”系列；对影印版的教材，则单独开辟出“经典原版书库”；定位在高级教程和专业参考的“计算机科学丛书”还将保持原来的风格，继续出版新的品种。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

“经典原版书库”是响应教育部提出的使用原版国外教材的号召，为国内高校的计算机教学

度身订造的。在广泛地征求并听取丛书的“专家指导委员会”的意见后，我们最终选定了这30多种篇幅内容适度、讲解鞭辟入里的教材，其中的大部分已经被M.I.T.、Stanford、U.C. Berkley、C.M.U.等世界名牌大学采用。丛书不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995265
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

John L. Hennessy is the president of Stanford University, where he has been a member of the faculty since 1977 in the departments of electrical engineering and computer science. Hennessy is a Fellow of the IEEE and ACM, a member of the National Academy of Engineering, and a Fellow of the American Academy of Arts and Sciences. He received the 2001 Eckert-Mauchly Award for his contributions to RISC technology, the 2001 Seymour Cray Computer Engineering Award, and shared the John von Neumann Award in 2000 with David Patterson. Hennessy's original research group at Stanford developed several of the techniques now in commercial use for optimizing compilers. In 1981, he started the MIPS project at Stanford with a handful of graduate students.

After completing the project in 1984, he took a one-year leave from the university to cofound MIPS Computer Systems, which developed one of the first commercial RISC microprocessors. After being acquired by Silicon Graphics in 1991, MIPS Technologies became an independent company in 1998, focusing on microprocessors for the embedded marketplace. As of 2001, over 200 million MIPS microprocessors have been shipped in devices ranging from video games and palmtop computers to laser printers and network switches.

Hennessy's more recent research at Stanford focuses on the area of designing and exploiting multiprocessors. He helped lead the design of the DASH multiprocessor architecture, the first distributed shared-memory multiprocessors supporting cache coherency and the basis for several commercial multiprocessor designs, including the Silicon Graphics Origin multiprocessors.

David A. Patterson has been teaching computer architecture at the University of California, Berkeley, since joining the faculty in 1977, and he holds the Pardee Chair of Computer Science. His teaching has been honored by the ACM and the University of California. In 2000 he won the James H. Mulligan, Jr. Education Medal from IEEE "for inspirational teaching through the development of creative curricula and teaching methodology, for important textbooks, and for effective integration of education and research missions." Patterson has also received the 1995 IEEE Technical Achievement Award for contributions to RISC and shared the 1999 IEEE Reynold B. Johnson Information Storage Award for contributions to RAID. In 2000 he shared the IEEE John von Neumann Medal with John Hennessy "for creating a revolution in computer architecture through their exploration, popularization, and commercialization of architectural innovations." Patterson is a member of the National Academy of Engineering and is a Fellow of both the ACM and the IEEE. In the past, he has been chair of the CS division in the EECS department at Berkeley, the ACM SIG in computer architecture, and the Computing Research Association.

At Berkeley, Patterson led the design and implementation of RISC I, likely the first VLSI reduced instruction set computer. This research became the foundation of the SPARC architecture, currently used by Sun Microsystems, Fujitsu, and others. He was a leader of the redundant arrays of inexpensive disks (RAID) project, which led to high-performance storage systems from many companies. He was also involved in the network of workstations (NOW) project, which led to cluster technology used by Internet companies. These projects earned three dissertation awards from the ACM. His current research project is called recovery oriented computing (ROC), which is developing techniques for building dependable, maintainable, and scalable Internet services.

Foreword

by Bill Joy, Chief Scientist and Corporate Executive Officer
Sun Microsystems, Inc.

I am very lucky to have studied computer architecture under Prof. David Patterson at U.C. Berkeley more than 20 years ago. I enjoyed the courses I took from him, in the early days of RISC architecture. Since leaving Berkeley to help found Sun Microsystems, I have used the ideas from his courses and many more that are described in this important book.

The good news today is that this book covers incredibly important and contemporary material. The further good news is that much exciting and challenging work remains to be done, and that working from *Computer Architecture: A Quantitative Approach* is a great way to start.

The most successful architectural projects that I have been involved in have always started from simple ideas, with advantages explainable using simple numerical models derived from hunches and rules of thumb. The continuing rapid advances in computing technology and new applications ensure that we will need new similarly simple models to understand what is possible in the future, and that new classes of applications will stress systems in different and interesting ways. The quantitative approach introduced in Chapter 1 is essential to understanding these issues. In particular, we expect to see, in the near future, much more emphasis on minimizing power to meet the demands of a given application, across all sizes of systems; much remains to be learned in this area.

I have worked with many different instruction sets in my career. I first programmed a PDP-8, whose instruction set was so simple that a friend easily learned to disassemble programs just by glancing at the hole punches in paper tape! I wrote a lot of code in PDP-11 assembler, including an interpreter for the Pascal programming language and for the VAX (which was used as an example in the first edition of this book); the success of the VAX led to the widespread use of UNIX on the early Internet.

The PDP-11 and VAX were very conventional complex instruction set (CISC) computer architectures, with relatively compact instruction sets that proved nearly impossible to pipeline. For a number of years in public talks I used the performance of the VAX 11/780 as the baseline; its speed was extremely well known because faster implementations of the architecture were so long delayed. VAX performance stalled out just as the x86 and 680x0 CISC architectures were

appearing in microprocessors; the strong economic advantages of microprocessors led to their overwhelming dominance. Then the simpler reduced instruction set (RISC) computer architectures—pioneered by John Cocke at IBM; promoted and named by Patterson and Hennessy; and commercialized in POWER PC, MIPS, and SPARC—were implemented as microprocessors and permitted high-performance pipeline implementations through the use of their simple register-oriented instruction sets. A downside of RISC was the larger code size of programs and resulting greater instruction fetch bandwidth, a cost that could be seen to be acceptable using the techniques of Chapter 1 and by believing in the future CMOS technology trends promoted in the now-classic views of Carver Mead. The kind of clear-thinking approach to the present problems and to the shape of future computing advances that led to RISC architecture is the focus of this book.

Chapter 2 (and various appendices) presents interesting examples of contemporary and important historical instruction set architecture. RISC architecture—the focus of so much work in the last twenty years—is by no means the final word here. I worked on the design of the SPARC architecture and several implementations for a decade, but more recently have worked on two different styles of processor: picoJava, which implemented most of the Java Virtual Machine instructions—a compact, high-level, bytecoded instruction set—and MAJC, a very simple and multithreaded VLIW for Java and media-intensive applications. These two architectures addressed different and new market needs: for low-power chips to run embedded devices where space and power are at a premium, and for high performance for a given amount of power and cost where parallel applications are possible. While neither has achieved widespread commercial success, I expect that the future will see many opportunities for different ISAs, and an in-depth knowledge of history here often gives great guidance—the relationships between key factors, such as the program size, execution speed, and power consumption, returning to previous balances that led to great designs in the past.

Chapters 3 and 4 describe instruction-level parallelism (ILP): the ability to execute more than one instruction at a time. This has been aided greatly, in the last 20 years, by techniques such as RISC and VLIW (very long instruction word) computing. But as later chapters here point out, both RISC and especially VLIW as practiced in the Intel itanium architecture are very power intensive. In our attempts to extract more instruction-level parallelism, we are running up against the fact that the complexity of a design that attempts to execute N instructions simultaneously grows like N^2: the number of transistors and number of watts to produce each result increases dramatically as we attempt to execute many instructions of arbitrary programs simultaneously. There is thus a clear countertrend emerging: using simpler pipelines with more realistic levels of ILP while exploiting other kinds of parallelism by running both multiple threads of execution per processor and, often, multiple processors on a single chip. The challenge for designers of high-performance systems of the future is to understand when simultaneous execution is possible, but then to use these techniques judiciously in combination with other, less granular techniques that are less power intensive and complex.

In graduate school I would often joke that cache memories were the only great idea in computer science. But truly, where you put things affects profoundly the design of computer systems. Chapter 5 describes the classical design of cache and main memory hierarchies and virtual memory. And now, new, higher-level programming languages like Java support much more reliable software because they insist on the use of garbage collection and array bounds checking, so that security breaches from "buffer overflow" and insidious bugs from false sharing of memory do not creep into large programs. It is only languages, such as Java, that insist on the use of automatic storage management that can implement true software components. But garbage collectors are notoriously hard on memory hierarchies, and the design of systems and language implementations to work well for such areas is an active area of research, where much good work has been done but much exciting work remains.

Java also strongly supports thread-level parallelism—a key to simple, power-efficient, and high-performance system implementations that avoids the N^2 problem discussed earlier but brings challenges of its own. A good foundational understanding of these issues can be had in Chapter 6. Traditionally, each processor was a separate chip, and keeping the various processors synchronized was expensive, both because of its impact on the memory hierarchy and because the synchronization operations themselves were very expensive. The Java language is also trying to address these issues: we tried, in the Java Language Specification, which I coauthored, to write a description of the memory model implied by the language. While this description turned out to have (fixable) technical problems, it is increasingly clear that we need to think about the memory hierarchy in the design of languages that are intended to work well on the newer system platforms. We view the Java specification as a first step in much good work to be done in the future.

As Chapter 7 describes, storage has evolved from being connected to individual computers to being a separate network resource. This is reminiscent of computer graphics, where graphics processing that was previously done in a host processor often became a separate function as the importance of graphics increased. All this is likely to change radically in the coming years—massively parallel host processors are likely to be able to do graphics better than dedicated outboard graphics units, and new breakthroughs in storage technologies, such as memories made from molecular electronics and other atomic-level nanotechnologies, should greatly reduce both the cost of storage and the access time. The resulting dramatic decreases in storage cost and access time will strongly encourage the use of multiple copies of data stored on individual computing nodes, rather than shared over a network. The "wheel of reincarnation," familiar from graphics, will appear in storage.

It is also critical that storage systems, and indeed all systems, become much more robust in the face of failures, not only of hardware, but also of software flaws and human error. This is an enormous challenge in the years ahead.

Chapter 8 provides a great foundational description of computer interconnects and networks. My model of these comes from Andy Bechtolsheim, another of the

cofounders of Sun, who famously said, "Ethernet always wins." More modestly stated: given the need for a new networking interconnect, and despite its shortcomings, adapted versions of the Ethernet protocols seem to have met with overwhelming success in the marketplace. Why? Factors such as the simplicity and familiarity of the protocols are obvious, but quite possibly the most likely reason is that the people who are adapting Ethernet can get on with the job at hand rather than arguing about details that, in the end, aren't dispositive. This lesson can be generalized to apply to all the areas of computer architecture discussed in this book.

One of the things I remember Dave Patterson saying many years ago is that for each new project you only get so many "cleverness beans." That is, you can be very clever in a few areas of your design, but if you try to be clever in all of them, the design will probably fail to achieve its goals—or even fail to work or to be finished at all. The overriding lesson that I have learned in 20 plus years of working on these kinds of designs is that you must choose what is important and focus on that; true wisdom is to know what to leave out. A deep knowledge of what has gone before is key to this ability.

And you must also choose your assumptions carefully. Many years ago I attended a conference in Hawaii (yes, it was a boondoggle, but read on) where Maurice Wilkes, the legendary computer architect, gave a speech. What he said, paraphrased in my memory, is that good research often consists of assuming something that seems untrue or unlikely today will become true and investigating the consequences of that assumption. And if the unlikely assumption indeed then becomes true in the world, you will have done timely and sometimes, then, even great research! So, for example, the research group at Xerox PARC assumed that everyone would have access to a personal computer with a graphics display connected to others by an internetwork and the ability to print inexpensively using Xerography. How true all this became, and how seminally important their work was!

In our time, and in the field of computer architecture, I think there are a number of assumptions that will become true. Some are not controversial, such as that Moore's Law is likely to continue for another decade or so and that the complexity of large chip designs is reaching practical limits, often beyond the point of positive returns for additional complexity. More controversially, perhaps, molecular electronics is likely to greatly reduce the cost of storage and probably logic elements as well, optical interconnects will greatly increase the bandwidth and reduce the error rates of interconnects, software will continue to be unreliable because it is so difficult, and security will continue to be important because its absence is so debilitating.

Taking advantage of the strong positive trends detailed in this book and using them to mitigate the negative ones will challenge the next generation of computer architects, to design a range of systems of many shapes and sizes.

Computer architecture design problems are becoming more varied and interesting. Now is an exciting time to be starting out or reacquainting yourself with the latest in this field, and this book is the best place to start. See you in the chips!

Preface

Why We Wrote This Book

Through three editions of this book, our goal has been to describe the basic principles underlying what will be tomorrow's technological developments. Our excitement about the opportunities in computer architecture has not abated, and we echo what we said about the field in the first edition: "It is not a dreary science of paper machines that will never work. No! It's a discipline of keen intellectual interest, requiring the balance of marketplace forces to cost-performance-power, leading to glorious failures and some notable successes."

Our primary objective in writing our first book was to change the way people learn and think about computer architecture. We feel this goal is still valid and important. The field is changing daily and must be studied with real examples and measurements on real computers, rather than simply as a collection of definitions and designs that will never need to be realized. We offer an enthusiastic welcome to anyone who came along with us in the past, as well as to those who are joining us now. Either way, we can promise the same quantitative approach to, and analysis of, real systems.

As with earlier versions, we have strived to produce a new edition that will continue to be as relevant for professional engineers and architects as it is for those involved in advanced computer architecture and design courses. As much as its predecessors, this edition aims to demystify computer architecture through an emphasis on cost-performance-power trade-offs and good engineering design. We believe that the field has continued to mature and move toward the rigorous quantitative foundation of long-established scientific and engineering disciplines. Our greatest satisfaction derives from the fact that the principles described in our first edition in 1990 and the second edition in 1996 could be applied successfully to help predict the landscape of computing technology that exists today. We hope that this third edition will allow readers to apply the fundamentals for similar results as we look forward to the coming decades.

This Edition

The third edition of *Computer Architecture: A Quantitative Approach* should have been easy to write. After all, our quantitative approach hasn't changed, and we sought to continue our focus on the basic principles of computer design through two editions. The examples had to be updated, of course, just as we did for the second edition. The dramatic and ongoing advances in the field as well as the creation of new markets for computers and new approaches for those markets, however, led us to rewrite almost the entire book.

The pace of innovation in computer architecture continued unabated in the six years since the second edition. As when we wrote the second edition, we found that numerous new concepts needed to be introduced, and other material designated as more basic. Although this is officially the third edition of *Computer Architecture: A Quantitative Approach*, it is really our fifth book in a series that began with the first edition, continued with *Computer Organization and Design: The Hardware/Software Interface* (COD:HSI), and then the second edition of both books. Over time ideas that were once found here have moved to COD:HSI or to background tutorials in the appendices. This migration, combined with our goal to present concepts in the context of the most recent computers, meant there was remarkably little from the second edition that could be preserved intact, and practically nothing is left from the first edition.

Perhaps the biggest surprise for us was the realization that the computer architecture field had split into three related but different market segments, each with their own needs and somewhat different architectures to address them. The cost-performance theme of our first and second editions is currently best exemplified by desktop computers. The two new paths are embedded computers and server computers. This major shift in the field is reflected in this edition by two major changes. First, throughout the text we broaden the topics considered as well as the metrics of success. Second, a new section, called "Another View," supplements the more traditional examples in "Putting It All Together" with examples that include video games, digital cameras, and cell phones.

Embedded computers have much lower cost targets than do desktop computers. They are often employed in environments where they run a single application. Also, embedded computers often rely on batteries and cannot use active cooling mechanisms, and energy/power efficiency is thus critical. To illustrate the design trade-offs and approaches in embedded processors we made several additions: the EEMBC benchmarks are used to evaluate performance, media processor and DSP instruction set principles and measurements are examined, the most popular embedded instruction set architectures are surveyed in the appendices, and performance-power trade-offs are explored in several chapters. Power-sensitive examples include the Transmeta and low-power MIPS processors, and embedded systems examples include the PlayStation-2 video game, Sanyo digital camera, and Nokia cell phone.

Server computers place more emphasis on reliability, scalability, and on throughput rather than latency to measure performance. Thus, these systems typi-

cally include multiple processors and disks. This edition explains the concept of dependability and includes rarely found statistics on the frequency of component failures. In addition to the SPEC2000 benchmarks for processors, we examine the TPC database benchmarks and the SPEC benchmarks for file servers. Examples of server processors include the Intel IA-64 and the Sun UltraSPARC III, and examples of server systems include the Sun Fire 6800, the Sun Wildfire, EMC Symmetrix, EMC Celerra, the Google cluster, and an IBM cluster for transaction processing.

This edition continues the tradition of using real-world examples to demonstrate the ideas, and the "Putting It All Together" sections are essentially 100% new. The "Putting It All Together" sections of this book include the MIPS64 instruction set architecture, the Intel Pentium III and 4 pipeline organization, the Intel IA-64 architecture and microarchitecture, the Alpha 21264 memory hierarchy, the Sun Wildfire multiprocessor, the EMC Symmetrix storage array, the EMC Celerra file server, and the Google search engine. The "Another View" sections pick real-world examples from the embedded and server communities. This list has the Trimedia TMS media processor, a PowerPC multithreaded processor, the memory hierarchy of Emotion Engine in the Sony Playstation-2, Sun Fire 6800/UltraSPARC III memory hierarchy, EmpowerTel MXP embedded multiprocessor, Sanyo digital camera, and Nokia cell phone.

In response to numerous comments, considerable effort was focused on revising and enhancing the exercises. In particular, all the exercises were reviewed to try to reduce ambiguities and eliminate unproductive exercises, and many new exercises were developed. As many readers requested, Appendix B provides answers to selected exercises.

We also added some new features that should help readers. We replaced the synthetic 32-bit DLX architecture with the popular 64-bit MIPS architecture, as it just made more sense to use existing software rather than recreate and maintain compilers ourselves. We also added a large set of appendices that contains descriptions of a dozen instruction set architectures plus tutorials on basic pipelining, vector processors, and floating-point arithmetic.

Topic Selection and Organization

As before, we have taken a conservative approach to topic selection, for there are many more interesting ideas in the field than can reasonably be covered in a treatment of basic principles. We have steered away from a comprehensive survey of every architecture a reader might encounter. Instead, our presentation focuses on core concepts likely to be found in any new machine. The key criterion remains that of selecting ideas that have been examined and utilized successfully enough to permit their discussion in quantitative terms.

Our first dilemma in determining the new topic selections was that topics requiring only a few pages in the prior editions have since exploded in their importance. Second, topics that we excluded previously have matured to a point where they can be discussed based on our quantitative criteria and their success in

the marketplace. To allow for this new material, we reduced the extent of introductory material, assuming the knowledge of the concepts in our introductory text *Computer Organization and Design: The Hardware/Software Interface*. Appendix A on pipelining was added as a valuable tutorial for readers not familiar with the basics of pipelining. (Readers interested strictly in a more basic introduction to computer architecture should read *Computer Organization and Design: The Hardware/Software Interface*.)

Our intent has always been to focus on material that is not available in equivalent form from other sources, so we continue to emphasize advanced content wherever possible. Indeed, there are several systems here whose descriptions cannot be found in the literature.

An Overview of the Content

Chapter 1 covers the basic quantitative principles of computer design and performance measurement. It also addresses the role of technology and the factors affecting the cost of computer systems. It concludes by examining performance and price-performance measurements of processors designed for the desktop, server, and embedded markets, as well as considering the power efficiency of embedded processors.

Chapter 2 covers instruction set design principles and examples. In addition to giving quantitative data on instruction set usage based on the SPEC2000 benchmarks, it describes the MIPS64 architecture used throughout the book. New to this edition are principles of digital signal processor architectures, including common features and measurements. It describes the structure of modern compilers and how that affects the utility of instruction sets for traditional computers, DSPs, and media extensions. It also gives the Trimedia TM5200 as a contrasting example of a media processor, offering instruction mixes for both it and MIPS. Appendices C to G extend this chapter by describing a dozen other popular instruction sets.

Chapters 3 and 4 cover the exploitation of instruction-level parallelism in high-performance processors, including superscalar execution, branch prediction, speculation, dynamic scheduling, and the relevant compiler technology. These topics have grown so much that, even with the creation of a 100-page appendix based on Chapter 3 of the second edition, we still needed two chapters to cover the advanced material. Chapter 3 of this edition focuses on hardware-based approaches to exploiting instruction-level parallelism, while Chapter 4 focuses on more static approaches that rely on more sophisticated compiler technology. The Intel Pentium series is used as the major example in Chapter 3, while Chapter 4 examines the IA-64 architecture and its first implementation in Itanium.

Chapter 5 starts with an introductory review of cache principles. It then reorganizes the optimizations in memory hierarchy design to what are the major challenges today. In addition to real-world examples from traditional computers such as the Alpha 21264, AMD Athlon, and Intel Pentium III and 4, it describes the memory hierarchy of the Emotion Engine in the Sony Playstation-2 video game

and the Sun Fire 6800 server with its UltraSPARC III processor. This edition describes the techniques of the bandwidth-optimized DRAM chips such as RAMBUS, and comments on their cost-performance. It also includes cache performance of multimedia and server applications in addition to the SPEC2000 benchmarks for the desktop.

Chapter 6 discusses multiprocessor systems, focusing on shared-memory architectures. The chapter begins by examining the properties of different application domains with thread-level parallelism. It then explores symmetric and distributed memory architectures, examining both organizational principles and performance. Topics in synchronization, memory consistency models, and multithreading (including simultaneous multithreading) complete the foundational chapters. Sun's Wildfire design, which uses a distributed memory architecture to extend the reach of a symmetric approach, is discussed and analyzed.

Chapter 7, "Storage Systems," saw a surprising amount of revision. There is an expansion of reliability and availability, a tutorial on RAID, availability benchmarks, and rarely found failure statistics of real systems. It continues to provide an introduction to queuing theory and I/O performance benchmarks. It extends the description of traditional buses with embedded and server buses. The five design examples in later sections evolve an I/O system through increasingly realistic performance assumptions, plus an evaluation of the mean time to failure. EMC supplies the examples that put it all together, which is the first time these systems have been documented publicly. The anatomy of a digital camera offers an embedded perspective on storage systems, and the historical perspective includes a ringside view of the development and popularity of RAID.

A goal of Chapter 8 is to provide an introduction to networks from the computer architecture point of view. Since this field is vast and quickly moving, the emphasis here is on an introduction to the terminology and principles. It starts with providing a common framework for the design principles in local area networks, storage area networks, and wide area networks, concluding with a description of the technology of the Internet. The second part of Chapter 8 is an in-depth exploration of clusters and the pros and cons of the use of clusters in both scientific computing and database applications. There is a detailed evaluation of the cost-performance of clusters, including the cost of machine room space and network bandwidth. The first description of the cluster used to provide the popular Google search engine puts this chapter together.

This brings us to Appendices A through I. Appendix A is a tutorial on basic pipelining concepts. Readers relatively new to pipelining should read this appendix before Chapters 3 and 4. As mentioned earlier, Appendix B contains solutions to selected exercises. Given the ubiquity of the Web today, the remaining appendices are online, which allows us to add relevant information without increasing the weight or cost of the book. Appendix C updates the second edition RISC appendix, describing 64-bit versions of Alpha, MIPS, PowerPC, and SPARC and their multimedia extensions. Also included in this appendix are popular embedded instruction sets: ARM, Thumb, SuperH, MIPS16, and Mitsubishi M32R. Appendix D describes the 80x86 architecture. Since we have no page budget for

the online appendices, we include two architectures of more historical interest: the VAX (Appendix E) and IBM 360/370 (Appendix F). Appendix G includes an updated description of vector processors. Finally, Appendix H describes computer arithmetic, and Appendix I describes implementing coherence protocols.

In summary, about 70% of the pages are new to this edition. The third edition is also about 10% longer than the first if we don't include the online appendices, and about 30% longer if we do.

Navigating the text

There is no single best order in which to approach these chapters. We wrote the text so that it can be covered in several ways, the only real restriction being that some chapters should be read in sequence, namely, Chapters 2, 3, and 4 (pipelining) and Chapters 7 and 8 (storage systems, interconnection networks, and clusters). Readers should start with Chapter 1 and should read Chapter 5 (memory hierarchy design) before Chapter 6 (multiprocessors). Appendices C, D, E, F, and H should be read after Chapter 2. If Appendix A is going to be read, it should be read before Chapters 3 and 4. Appendix G is an interesting contrast to the ideas in Chapters 3 and 4.

Despite the many ways to read this book, we expect two primary paths:

1. *Inside out:* The philosophy of this choice is that processor design is still the cornerstone of computer architecture, and the nonprocessor topics are covered as time permits. Start with Chapter 1, then inside the processor (Chapters 2, 3, 4), then memory hierarchy (5), followed by multiprocessors (6), storage (7), and finish with networks and clusters (8).
2. *Outside in:* The philosophy of this path is that the most interesting challenges in computer architecture today are outside the processor, and that processor internals are covered as time permits. Start again with Chapter 1, then memory hierarchy (5), followed by multiprocessors (6), storage (7), networks and clusters (8), and conclude with instruction sets and pipelining (Chapters 2, 3, 4).

Chapter Structure and Exercises

The material we have selected has been stretched upon a consistent framework that is followed in each chapter. We start by explaining the ideas of a chapter. These ideas are followed by a "Crosscutting Issues" section, a feature that shows how the ideas covered in one chapter interact with those given in other chapters. This is followed by a "Putting It All Together" section that ties these ideas together by showing how they are used in a real machine. This is followed by one or two sections titled "Another View," a new feature for the third edition that gives a real-world example from the embedded or server space.

Next in the sequence is "Fallacies and Pitfalls," which lets readers learn from the mistakes of others. We show examples of common misunderstandings and