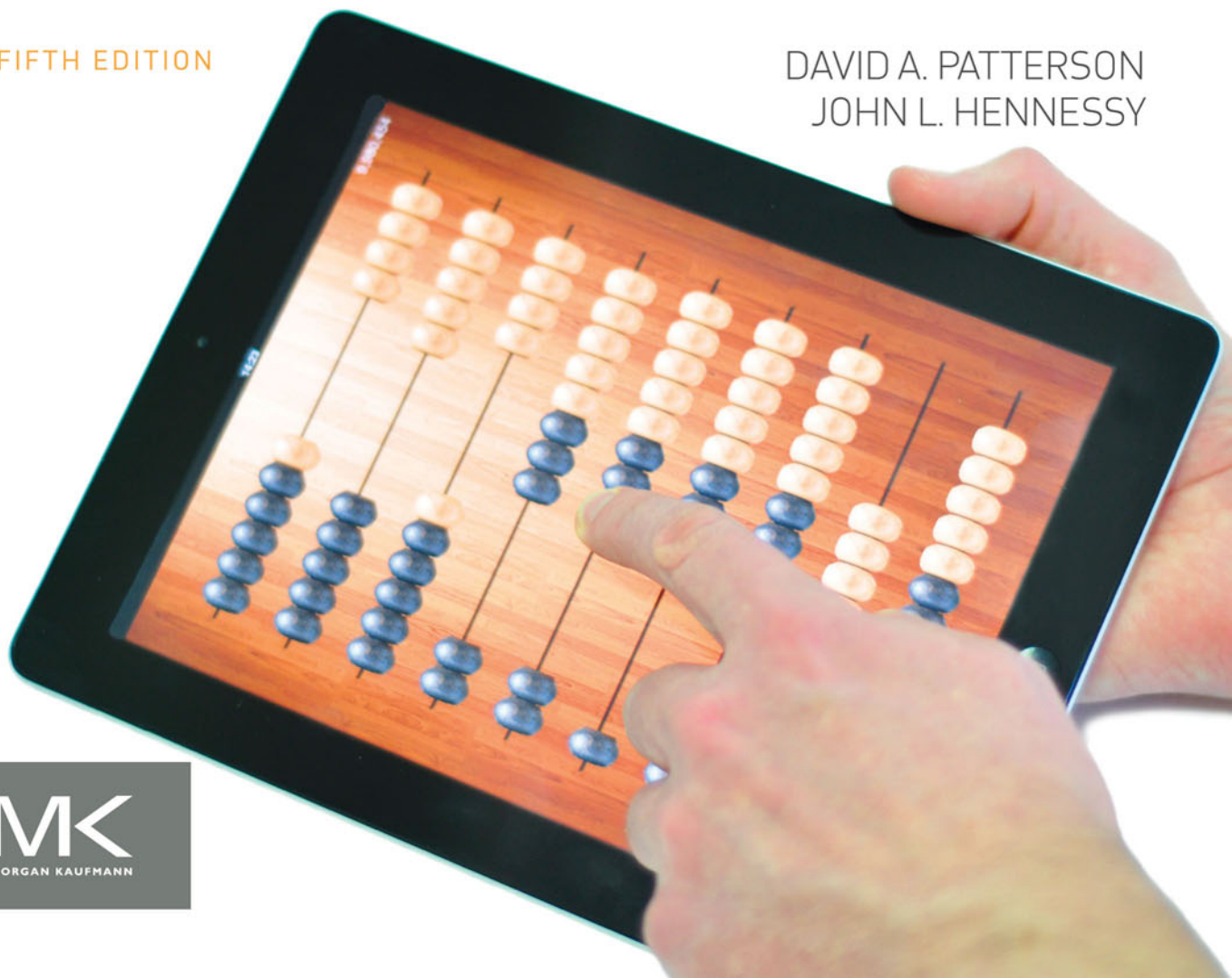


COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE

FIFTH EDITION

DAVID A. PATTERSON
JOHN L. HENNESSY



MK
MORGAN KAUFMANN

In Praise of *Computer Organization and Design: The Hardware/Software Interface*, Fifth Edition

“Textbook selection is often a frustrating act of compromise—pedagogy, content coverage, quality of exposition, level of rigor, cost. *Computer Organization and Design* is the rare book that hits all the right notes across the board, without compromise. It is not only the premier computer organization textbook, it is a shining example of what all computer science textbooks could and should be.”

—Michael Goldweber, *Xavier University*

“I have been using *Computer Organization and Design* for years, from the very first edition. The new Fifth Edition is yet another outstanding improvement on an already classic text. The evolution from desktop computing to mobile computing to Big Data brings new coverage of embedded processors such as the ARM, new material on how software and hardware interact to increase performance, and cloud computing. All this without sacrificing the fundamentals.”

—Ed Harcourt, *St. Lawrence University*

“To Millennials: *Computer Organization and Design* is the computer architecture book you should keep on your (virtual) bookshelf. The book is both old and new, because it develops venerable principles—Moore's Law, abstraction, common case fast, redundancy, memory hierarchies, parallelism, and pipelining—but illustrates them with contemporary designs, e.g., ARM Cortex A8 and Intel Core i7.”

—Mark D. Hill, *University of Wisconsin-Madison*

“The new edition of *Computer Organization and Design* keeps pace with advances in emerging embedded and many-core (GPU) systems, where tablets and smartphones will be quickly becoming our new desktops. This text acknowledges these changes, but continues to provide a rich foundation of the fundamentals in computer organization and design which will be needed for the designers of hardware and software that power this new class of devices and systems.”

—Dave Kaeli, *Northeastern University*

“The Fifth Edition of *Computer Organization and Design* provides more than an introduction to computer architecture. It prepares the reader for the changes necessary to meet the ever-increasing performance needs of mobile systems and big data processing at a time that difficulties in semiconductor scaling are making all systems power constrained. In this new era for computing, hardware and software must be co-designed and system-level architecture is as critical as component-level optimizations.”

—Christos Kozyrakis, *Stanford University*

“Patterson and Hennessy brilliantly address the issues in ever-changing computer hardware architectures, emphasizing on interactions among hardware and software components at various abstraction levels. By interspersing I/O and parallelism concepts with a variety of mechanisms in hardware and software throughout the book, the new edition achieves an excellent holistic presentation of computer architecture for the PostPC era. This book is an essential guide to hardware and software professionals facing energy efficiency and parallelization challenges in Tablet PC to cloud computing.”

—Jae C. Oh, *Syracuse University*

This page intentionally left blank

F I F T H E D I T I O N

Computer Organization and Design

T H E H A R D W A R E / S O F T W A R E I N T E R F A C E

David A. Patterson has been teaching computer architecture at the University of California, Berkeley, since joining the faculty in 1977, where he holds the Pardee Chair of Computer Science. His teaching has been honored by the Distinguished Teaching Award from the University of California, the Karlstrom Award from ACM, and the Mulligan Education Medal and Undergraduate Teaching Award from IEEE. Patterson received the IEEE Technical Achievement Award and the ACM Eckert-Mauchly Award for contributions to RISC, and he shared the IEEE Johnson Information Storage Award for contributions to RAID. He also shared the IEEE John von Neumann Medal and the C & C Prize with John Hennessy. Like his co-author, Patterson is a Fellow of the American Academy of Arts and Sciences, the Computer History Museum, ACM, and IEEE, and he was elected to the National Academy of Engineering, the National Academy of Sciences, and the Silicon Valley Engineering Hall of Fame. He served on the Information Technology Advisory Committee to the U.S. President, as chair of the CS division in the Berkeley EECS department, as chair of the Computing Research Association, and as President of ACM. This record led to Distinguished Service Awards from ACM and CRA.

At Berkeley, Patterson led the design and implementation of RISC I, likely the first VLSI reduced instruction set computer, and the foundation of the commercial SPARC architecture. He was a leader of the Redundant Arrays of Inexpensive Disks (RAID) project, which led to dependable 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 and later to cloud computing. These projects earned three dissertation awards from ACM. His current research projects are Algorithm-Machine-People and Algorithms and Specializers for Provably Optimal Implementations with Resilience and Efficiency. The AMP Lab is developing scalable machine learning algorithms, warehouse-scale-computer-friendly programming models, and crowd-sourcing tools to gain valuable insights quickly from big data in the cloud. The ASPIRE Lab uses deep hardware and software co-tuning to achieve the highest possible performance and energy efficiency for mobile and rack computing systems.

John L. Hennessy is the tenth 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, the National Academy of Science, and the American Philosophical Society; and a Fellow of the American Academy of Arts and Sciences. Among his many awards are the 2001 Eckert-Mauchly Award for his contributions to RISC technology, the 2001 Seymour Cray Computer Engineering Award, and the 2000 John von Neumann Award, which he shared with David Patterson. He has also received seven honorary doctorates.

In 1981, he started the MIPS project at Stanford with a handful of graduate students. After completing the project in 1984, he took a leave from the university to cofound MIPS Computer Systems (now MIPS Technologies), which developed one of the first commercial RISC microprocessors. As of 2006, over 2 billion MIPS microprocessors have been shipped in devices ranging from video games and palmtop computers to laser printers and network switches. Hennessy subsequently led the DASH (Director Architecture for Shared Memory) project, which prototyped the first scalable cache coherent multiprocessor; many of the key ideas have been adopted in modern multiprocessors. In addition to his technical activities and university responsibilities, he has continued to work with numerous start-ups both as an early-stage advisor and an investor.

F I F T H E D I T I O N

Computer Organization and Design

THE HARDWARE / SOFTWARE INTERFACE

David A. Patterson

University of California, Berkeley

John L. Hennessy

Stanford University

With contributions by

Perry Alexander

The University of Kansas

Peter J. Ashenden

Ashenden Designs Pty Ltd

Jason D. Bakos

University of South Carolina

Javier Bruguera

Universidade de Santiago de Compostela

Jichuan Chang

Hewlett-Packard

Matthew Farrens

University of California, Davis

David Kaeli

Northeastern University

Nicole Kaiyan

University of Adelaide

David Kirk

NVIDIA

James R. Larus

School of Computer and
Communications Science at EPFL

Jacob Leverich

Hewlett-Packard

Kevin Lim

Hewlett-Packard

John Nickolls

NVIDIA

John Oliver

Cal Poly, San Luis Obispo

Milos Prvulovic

Georgia Tech

Partha Ranganathan

Hewlett-Packard



ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier



Acquiring Editor: Todd Green
Development Editor: Nate McFadden
Project Manager: Lisa Jones
Designer: Russell Purdy

Morgan Kaufmann is an imprint of Elsevier
The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB
225 Wyman Street, Waltham, MA 02451, USA

Copyright © 2014 Elsevier Inc. All rights reserved

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: www.elsevier.com/permissions

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods or professional practices, may become necessary. Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information or methods described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data

Patterson, David A.

Computer organization and design: the hardware/software interface/David A. Patterson, John L. Hennessy. — 5th ed.
p. cm. — (The Morgan Kaufmann series in computer architecture and design)

Rev. ed. of: Computer organization and design/John L. Hennessy, David A. Patterson. 1998.

Summary: "Presents the fundamentals of hardware technologies, assembly language, computer arithmetic, pipelining, memory hierarchies and I/O"— Provided by publisher.

ISBN 978-0-12-407726-3 (pbk.)

1. Computer organization. 2. Computer engineering. 3. Computer interfaces. I. Hennessy, John L. II. Hennessy, John L. Computer organization and design. III. Title.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

ISBN: 978-0-12-407726-3

For information on all MK publications visit our
website at www.mkp.com

Printed and bound in the United States of America

13 14 15 16 10 9 8 7 6 5 4 3 2 1



Working together
to grow libraries in
developing countries

www.elsevier.com • www.bookaid.org

*To Linda,
who has been, is, and always will be the love of my life*

ACKNOWLEDGMENTS

Figures 1.7, 1.8 Courtesy of iFixit (www.ifixit.com).

Figure 1.9 Courtesy of Chipworks (www.chipworks.com).

Figure 1.13 Courtesy of Intel.

Figures 1.10.1, 1.10.2, 4.15.2 Courtesy of the Charles Babbage Institute, University of Minnesota Libraries, Minneapolis.

Figures 1.10.3, 4.15.1, 4.15.3, 5.12.3, 6.14.2 Courtesy of IBM.

Figure 1.10.4 Courtesy of Cray Inc.

Figure 1.10.5 Courtesy of Apple Computer, Inc.

Figure 1.10.6 Courtesy of the Computer History Museum.

Figures 5.17.1, 5.17.2 Courtesy of Museum of Science, Boston.

Figure 5.17.4 Courtesy of MIPS Technologies, Inc.

Figure 6.15.1 Courtesy of NASA Ames Research Center.

Contents

Preface xv

CHAPTERS

1

Computer Abstractions and Technology 2

- 1.1 Introduction 3
- 1.2 Eight Great Ideas in Computer Architecture 11
- 1.3 Below Your Program 13
- 1.4 Under the Covers 16
- 1.5 Technologies for Building Processors and Memory 24
- 1.6 Performance 28
- 1.7 The Power Wall 40
- 1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors 43
- 1.9 Real Stuff: Benchmarking the Intel Core i7 46
- 1.10 Fallacies and Pitfalls 49
- 1.11 Concluding Remarks 52
-  1.12 Historical Perspective and Further Reading 54
- 1.13 Exercises 54

2

Instructions: Language of the Computer 60

- 2.1 Introduction 62
- 2.2 Operations of the Computer Hardware 63
- 2.3 Operands of the Computer Hardware 66
- 2.4 Signed and Unsigned Numbers 73
- 2.5 Representing Instructions in the Computer 80
- 2.6 Logical Operations 87
- 2.7 Instructions for Making Decisions 90
- 2.8 Supporting Procedures in Computer Hardware 96
- 2.9 Communicating with People 106
- 2.10 MIPS Addressing for 32-Bit Immediates and Addresses 111
- 2.11 Parallelism and Instructions: Synchronization 121
- 2.12 Translating and Starting a Program 123
- 2.13 A C Sort Example to Put It All Together 132
- 2.14 Arrays versus Pointers 141

-  2.15 Advanced Material: Compiling C and Interpreting Java 145
- 2.16 Real Stuff: ARMv7 (32-bit) Instructions 145
- 2.17 Real Stuff: x86 Instructions 149
- 2.18 Real Stuff: ARMv8 (64-bit) Instructions 158
- 2.19 Fallacies and Pitfalls 159
- 2.20 Concluding Remarks 161
-  2.21 Historical Perspective and Further Reading 163
- 2.22 Exercises 164

3 Arithmetic for Computers 176

- 3.1 Introduction 178
- 3.2 Addition and Subtraction 178
- 3.3 Multiplication 183
- 3.4 Division 189
- 3.5 Floating Point 196
- 3.6 Parallelism and Computer Arithmetic: Subword Parallelism 222
- 3.7 Real Stuff: Streaming SIMD Extensions and Advanced Vector Extensions in x86 224
- 3.8 Going Faster: Subword Parallelism and Matrix Multiply 225
- 3.9 Fallacies and Pitfalls 229
- 3.10 Concluding Remarks 232
-  3.11 Historical Perspective and Further Reading 236
- 3.12 Exercises 237

4 The Processor 242

- 4.1 Introduction 244
- 4.2 Logic Design Conventions 248
- 4.3 Building a Datapath 251
- 4.4 A Simple Implementation Scheme 259
- 4.5 An Overview of Pipelining 272
- 4.6 Pipelined Datapath and Control 286
- 4.7 Data Hazards: Forwarding versus Stalling 303
- 4.8 Control Hazards 316
- 4.9 Exceptions 325
- 4.10 Parallelism via Instructions 332
- 4.11 Real Stuff: The ARM Cortex-A8 and Intel Core i7 Pipelines 344
- 4.12 Going Faster: Instruction-Level Parallelism and Matrix Multiply 351
-  4.13 Advanced Topic: An Introduction to Digital Design Using a Hardware Design Language to Describe and Model a Pipeline and More Pipelining Illustrations 354

- 4.14 Fallacies and Pitfalls 355
- 4.15 Concluding Remarks 356
-  4.16 Historical Perspective and Further Reading 357
- 4.17 Exercises 357

5**Large and Fast: Exploiting Memory Hierarchy 372**

- 5.1 Introduction 374
- 5.2 Memory Technologies 378
- 5.3 The Basics of Caches 383
- 5.4 Measuring and Improving Cache Performance 398
- 5.5 Dependable Memory Hierarchy 418
- 5.6 Virtual Machines 424
- 5.7 Virtual Memory 427
- 5.8 A Common Framework for Memory Hierarchy 454
- 5.9 Using a Finite-State Machine to Control a Simple Cache 461
- 5.10 Parallelism and Memory Hierarchies: Cache Coherence 466
-  5.11 Parallelism and Memory Hierarchy: Redundant Arrays of Inexpensive Disks 470
-  5.12 Advanced Material: Implementing Cache Controllers 470
- 5.13 Real Stuff: The ARM Cortex-A8 and Intel Core i7 Memory Hierarchies 471
- 5.14 Going Faster: Cache Blocking and Matrix Multiply 475
- 5.15 Fallacies and Pitfalls 478
- 5.16 Concluding Remarks 482
-  5.17 Historical Perspective and Further Reading 483
- 5.18 Exercises 483

6**Parallel Processors from Client to Cloud 500**

- 6.1 Introduction 502
- 6.2 The Difficulty of Creating Parallel Processing Programs 504
- 6.3 SISD, MIMD, SIMD, SPMD, and Vector 509
- 6.4 Hardware Multithreading 516
- 6.5 Multicore and Other Shared Memory Multiprocessors 519
- 6.6 Introduction to Graphics Processing Units 524
- 6.7 Clusters, Warehouse Scale Computers, and Other Message-Passing Multiprocessors 531
- 6.8 Introduction to Multiprocessor Network Topologies 536
-  6.9 Communicating to the Outside World: Cluster Networking 539
- 6.10 Multiprocessor Benchmarks and Performance Models 540
- 6.11 Real Stuff: Benchmarking Intel Core i7 versus NVIDIA Tesla GPU 550

- 6.12 Going Faster: Multiple Processors and Matrix Multiply 555
- 6.13 Fallacies and Pitfalls 558
- 6.14 Concluding Remarks 560
-  6.15 Historical Perspective and Further Reading 563
- 6.16 Exercises 563

A P P E N D I C E S

A

Assemblers, Linkers, and the SPIM Simulator A-2

- A.1 Introduction A-3
- A.2 Assemblers A-10
- A.3 Linkers A-18
- A.4 Loading A-19
- A.5 Memory Usage A-20
- A.6 Procedure Call Convention A-22
- A.7 Exceptions and Interrupts A-33
- A.8 Input and Output A-38
- A.9 SPIM A-40
- A.10 MIPS R2000 Assembly Language A-45
- A.11 Concluding Remarks A-81
- A.12 Exercises A-82

B

The Basics of Logic Design B-2

- B.1 Introduction B-3
- B.2 Gates, Truth Tables, and Logic Equations B-4
- B.3 Combinational Logic B-9
- B.4 Using a Hardware Description Language B-20
- B.5 Constructing a Basic Arithmetic Logic Unit B-26
- B.6 Faster Addition: Carry Lookahead B-38
- B.7 Clocks B-48
- B.8 Memory Elements: Flip-Flops, Latches, and Registers B-50
- B.9 Memory Elements: SRAMs and DRAMs B-58
- B.10 Finite-State Machines B-67
- B.11 Timing Methodologies B-72
- B.12 Field Programmable Devices B-78
- B.13 Concluding Remarks B-79
- B.14 Exercises B-80

Index I-1

ONLINE CONTENT

**Graphics and Computing GPUs C-2**

- C.1 Introduction C-3
- C.2 GPU System Architectures C-7
- C.3 Programming GPUs C-12
- C.4 Multithreaded Multiprocessor Architecture C-25
- C.5 Parallel Memory System C-36
- C.6 Floating Point Arithmetic C-41
- C.7 Real Stuff: The NVIDIA GeForce 8800 C-46
- C.8 Real Stuff: Mapping Applications to GPUs C-55
- C.9 Fallacies and Pitfalls C-72
- C.10 Concluding Remarks C-76
- C.11 Historical Perspective and Further Reading C-77

**Mapping Control to Hardware D-2**

- D.1 Introduction D-3
- D.2 Implementing Combinational Control Units D-4
- D.3 Implementing Finite-State Machine Control D-8
- D.4 Implementing the Next-State Function with a Sequencer D-22
- D.5 Translating a Microprogram to Hardware D-28
- D.6 Concluding Remarks D-32
- D.7 Exercises D-33

**A Survey of RISC Architectures for Desktop, Server, and Embedded Computers E-2**

- E.1 Introduction E-3
- E.2 Addressing Modes and Instruction Formats E-5
- E.3 Instructions: The MIPS Core Subset E-9
- E.4 Instructions: Multimedia Extensions of the Desktop/Server RISCs E-16
- E.5 Instructions: Digital Signal-Processing Extensions of the Embedded RISCs E-19
- E.6 Instructions: Common Extensions to MIPS Core E-20
- E.7 Instructions Unique to MIPS-64 E-25
- E.8 Instructions Unique to Alpha E-27
- E.9 Instructions Unique to SPARC v9 E-29
- E.10 Instructions Unique to PowerPC E-32
- E.11 Instructions Unique to PA-RISC 2.0 E-34
- E.12 Instructions Unique to ARM E-36
- E.13 Instructions Unique to Thumb E-38
- E.14 Instructions Unique to SuperH E-39

E.15 Instructions Unique to M32R E-40

E.16 Instructions Unique to MIPS-16 E-40

E.17 Concluding Remarks E-43

 Glossary G-1

 Further Reading FR-1

Preface

The most beautiful thing we can experience is the mysterious. It is the source of all true art and science.

Albert Einstein, *What I Believe*, 1930

About This Book

We believe that learning in computer science and engineering should reflect the current state of the field, as well as introduce the principles that are shaping computing. We also feel that readers in every specialty of computing need to appreciate the organizational paradigms that determine the capabilities, performance, energy, and, ultimately, the success of computer systems.

Modern computer technology requires professionals of every computing specialty to understand both hardware and software. The interaction between hardware and software at a variety of levels also offers a framework for understanding the fundamentals of computing. Whether your primary interest is hardware or software, computer science or electrical engineering, the central ideas in computer organization and design are the same. Thus, our emphasis in this book is to show the relationship between hardware and software and to focus on the concepts that are the basis for current computers.

The recent switch from uniprocessor to multicore microprocessors confirmed the soundness of this perspective, given since the first edition. While programmers could ignore the advice and rely on computer architects, compiler writers, and silicon engineers to make their programs run faster or be more energy-efficient without change, that era is over. For programs to run faster, they must become parallel. While the goal of many researchers is to make it possible for programmers to be unaware of the underlying parallel nature of the hardware they are programming, it will take many years to realize this vision. Our view is that for at least the next decade, most programmers are going to have to understand the hardware/software interface if they want programs to run efficiently on parallel computers.

The audience for this book includes those with little experience in assembly language or logic design who need to understand basic computer organization as well as readers with backgrounds in assembly language and/or logic design who want to learn how to design a computer or understand how a system works and why it performs as it does.

About the Other Book

Some readers may be familiar with *Computer Architecture: A Quantitative Approach*, popularly known as Hennessy and Patterson. (This book in turn is often called Patterson and Hennessy.) Our motivation in writing the earlier book was to describe the principles of computer architecture using solid engineering fundamentals and quantitative cost/performance tradeoffs. We used an approach that combined examples and measurements, based on commercial systems, to create realistic design experiences. Our goal was to demonstrate that computer architecture could be learned using quantitative methodologies instead of a descriptive approach. It was intended for the serious computing professional who wanted a detailed understanding of computers.

A majority of the readers for this book do not plan to become computer architects. The performance and energy efficiency of future software systems will be dramatically affected, however, by how well software designers understand the basic hardware techniques at work in a system. Thus, compiler writers, operating system designers, database programmers, and most other software engineers need a firm grounding in the principles presented in this book. Similarly, hardware designers must understand clearly the effects of their work on software applications.

Thus, we knew that this book had to be much more than a subset of the material in *Computer Architecture*, and the material was extensively revised to match the different audience. We were so happy with the result that the subsequent editions of *Computer Architecture* were revised to remove most of the introductory material; hence, there is much less overlap today than with the first editions of both books.

Changes for the Fifth Edition

We had six major goals for the fifth edition of *Computer Organization and Design*: demonstrate the importance of understanding hardware with a running example; highlight major themes across the topics using margin icons that are introduced early; update examples to reflect changeover from PC era to PostPC era; spread the material on I/O throughout the book rather than isolating it into a single chapter; update the technical content to reflect changes in the industry since the publication of the fourth edition in 2009; and put appendices and optional sections online instead of including a CD to lower costs and to make this edition viable as an electronic book.

Before discussing the goals in detail, let's look at the table on the next page. It shows the hardware and software paths through the material. Chapters 1, 4, 5, and 6 are found on both paths, no matter what the experience or the focus. Chapter 1 discusses the importance of energy and how it motivates the switch from single core to multicore microprocessors and introduces the eight great ideas in computer architecture. Chapter 2 is likely to be review material for the hardware-oriented, but it is essential reading for the software-oriented, especially for those readers interested in learning more about compilers and object-oriented programming languages. Chapter 3 is for readers interested in constructing a datapath or in

Chapter or Appendix	Sections	Software focus	Hardware focus
1. Computer Abstractions and Technology	1.1 to 1.11		
	 1.12 (History)		
2. Instructions: Language of the Computer	2.1 to 2.14		
	 2.15 (Compilers & Java)		
	2.16 to 2.20		
	 2.21 (History)		
E. RISC Instruction-Set Architectures	 E.1 to E.17		
3. Arithmetic for Computers	3.1 to 3.5		
	3.6 to 3.8 (Subword Parallelism)		
	3.9 to 3.10 (Fallacies)		
	 3.11 (History)		
B. The Basics of Logic Design	B.1 to B.13		
4. The Processor	4.1 (Overview)		
	4.2 (Logic Conventions)		
	4.3 to 4.4 (Simple Implementation)		
	4.5 (Pipelining Overview)		
	4.6 (Pipelined Datapath)		
	4.7 to 4.9 (Hazards, Exceptions)		
	4.10 to 4.12 (Parallel, Real Stuff)		
	 4.13 (Verilog Pipeline Control)		
	4.14 to 4.15 (Fallacies)		
	 4.16 (History)		
D. Mapping Control to Hardware	 D.1 to D.6		
5. Large and Fast: Exploiting Memory Hierarchy	5.1 to 5.10		
	 5.11 (Redundant Arrays of Inexpensive Disks)		
	 5.12 (Verilog Cache Controller)		
	5.13 to 5.16		
	 5.17 (History)		
6. Parallel Process from Client to Cloud	6.1 to 6.8		
	 6.9 (Networks)		
	6.10 to 6.14		
	 6.15 (History)		
A. Assemblers, Linkers, and the SPIM Simulator	A.1 to A.11		
C. Graphics Processor Units	 C.1 to C.13		

Read carefully



Read if have time



Reference



Review or read



Read for culture



learning more about floating-point arithmetic. Some will skip parts of Chapter 3, either because they don't need them or because they offer a review. However, we introduce the running example of matrix multiply in this chapter, showing how subword parallelism offers a fourfold improvement, so don't skip sections 3.6 to 3.8. Chapter 4 explains pipelined processors. Sections 4.1, 4.5, and 4.10 give overviews and Section 4.12 gives the next performance boost for matrix multiply for those with a software focus. Those with a hardware focus, however, will find that this chapter presents core material; they may also, depending on their background, want to read Appendix C on logic design first. The last chapter on multicores, multiprocessors, and clusters, is mostly new content and should be read by everyone. It was significantly reorganized in this edition to make the flow of ideas more natural and to include much more depth on GPUs, warehouse scale computers, and the hardware-software interface of network interface cards that are key to clusters.

The first of the six goals for this fifth edition was to demonstrate the importance of understanding modern hardware to get good performance and energy efficiency with a concrete example. As mentioned above, we start with subword parallelism in Chapter 3 to improve matrix multiply by a factor of 4. We double performance in Chapter 4 by unrolling the loop to demonstrate the value of instruction level parallelism. Chapter 5 doubles performance again by optimizing for caches using blocking. Finally, Chapter 6 demonstrates a speedup of 14 from 16 processors by using thread-level parallelism. All four optimizations in total add just 24 lines of C code to our initial matrix multiply example.

The second goal was to help readers separate the forest from the trees by identifying eight great ideas of computer architecture early and then pointing out all the places they occur throughout the rest of the book. We use (hopefully) easy to remember margin icons and highlight the corresponding word in the text to remind readers of these eight themes. There are nearly 100 citations in the book. No chapter has less than seven examples of great ideas, and no idea is cited less than five times. Performance via parallelism, pipelining, and prediction are the three most popular great ideas, followed closely by Moore's Law. The processor chapter (4) is the one with the most examples, which is not a surprise since it probably received the most attention from computer architects. The one great idea found in every chapter is performance via parallelism, which is a pleasant observation given the recent emphasis in parallelism in the field and in editions of this book.

The third goal was to recognize the generation change in computing from the PC era to the PostPC era by this edition with our examples and material. Thus, Chapter 1 dives into the guts of a tablet computer rather than a PC, and Chapter 6 describes the computing infrastructure of the cloud. We also feature the ARM, which is the instruction set of choice in the personal mobile devices of the PostPC era, as well as the x86 instruction set that dominated the PC Era and (so far) dominates cloud computing.

The fourth goal was to spread the I/O material throughout the book rather than have it in its own chapter, much as we spread parallelism throughout all the chapters in the fourth edition. Hence, I/O material in this edition can be found in

Sections 1.4, 4.9, 5.2, 5.5, 5.11, and 6.9. The thought is that readers (and instructors) are more likely to cover I/O if it's not segregated to its own chapter.

This is a fast-moving field, and, as is always the case for our new editions, an important goal is to update the technical content. The running example is the ARM Cortex A8 and the Intel Core i7, reflecting our PostPC Era. Other highlights include an overview the new 64-bit instruction set of ARMv8, a tutorial on GPUs that explains their unique terminology, more depth on the warehouse scale computers that make up the cloud, and a deep dive into 10 Gigabyte Ethernet cards.

To keep the main book short and compatible with electronic books, we placed the optional material as online appendices instead of on a companion CD as in prior editions.

Finally, we updated all the exercises in the book.

While some elements changed, we have preserved useful book elements from prior editions. To make the book work better as a reference, we still place definitions of new terms in the margins at their first occurrence. The book element called “Understanding Program Performance” sections helps readers understand the performance of their programs and how to improve it, just as the “Hardware/Software Interface” book element helped readers understand the tradeoffs at this interface. “The Big Picture” section remains so that the reader sees the forest despite all the trees. “Check Yourself” sections help readers to confirm their comprehension of the material on the first time through with answers provided at the end of each chapter. This edition still includes the green MIPS reference card, which was inspired by the “Green Card” of the IBM System/360. This card has been updated and should be a handy reference when writing MIPS assembly language programs.

Changes for the Fifth Edition

We have collected a great deal of material to help instructors teach courses using this book. Solutions to exercises, figures from the book, lecture slides, and other materials are available to adopters from the publisher. Check the publisher's Web site for more information:

textbooks.elsevier.com/9780124077263

Concluding Remarks

If you read the following acknowledgments section, you will see that we went to great lengths to correct mistakes. Since a book goes through many printings, we have the opportunity to make even more corrections. If you uncover any remaining, resilient bugs, please contact the publisher by electronic mail at cod5bugs@mkp.com or by low-tech mail using the address found on the copyright page.

This edition is the second break in the long-standing collaboration between Hennessy and Patterson, which started in 1989. The demands of running one of the world's great universities meant that President Hennessy could no longer make the substantial commitment to create a new edition. The remaining author felt

once again like a tightrope walker without a safety net. Hence, the people in the acknowledgments and Berkeley colleagues played an even larger role in shaping the contents of this book. Nevertheless, this time around there is only one author to blame for the new material in what you are about to read.

Acknowledgments for the Fifth Edition

With every edition of this book, we are very fortunate to receive help from many readers, reviewers, and contributors. Each of these people has helped to make this book better.

Chapter 6 was so extensively revised that we did a separate review for ideas and contents, and I made changes based on the feedback from every reviewer. I'd like to thank **Christos Kozyrakis** of Stanford University for suggesting using the network interface for clusters to demonstrate the hardware-software interface of I/O and for suggestions on organizing the rest of the chapter; **Mario Flagsilk** of Stanford University for providing details, diagrams, and performance measurements of the NetFPGA NIC; and the following for suggestions on how to improve the chapter: **David Kaeli** of Northeastern University, **Partha Ranganathan** of HP Labs, **David Wood** of the University of Wisconsin, and my Berkeley colleagues **Siamak Faridani**, **Shoaib Kamil**, **Yunsup Lee**, **Zhangxi Tan**, and **Andrew Waterman**.

Special thanks goes to **Rimas Avizenis** of UC Berkeley, who developed the various versions of matrix multiply and supplied the performance numbers as well. As I worked with his father while I was a graduate student at UCLA, it was a nice symmetry to work with Rimas at UCB.

I also wish to thank my longtime collaborator **Randy Katz** of UC Berkeley, who helped develop the concept of great ideas in computer architecture as part of the extensive revision of an undergraduate class that we did together.

I'd like to thank **David Kirk**, **John Nickolls**, and their colleagues at NVIDIA (Michael Garland, John Montrym, Doug Voorhies, Lars Nyland, Erik Lindholm, Paulius Micikevicius, Massimiliano Fatica, Stuart Oberman, and Vasily Volkov) for writing the first in-depth appendix on GPUs. I'd like to express again my appreciation to **Jim Larus**, recently named Dean of the School of Computer and Communications Science at EPFL, for his willingness in contributing his expertise on assembly language programming, as well as for welcoming readers of this book with regard to using the simulator he developed and maintains.

I am also very grateful to **Jason Bakos** of the University of South Carolina, who updated and created new exercises for this edition, working from originals prepared for the fourth edition by **Perry Alexander** (The University of Kansas); **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); and **Jichuan Chang**, **Jacob Leverich**, **Kevin Lim**, and **Partha Ranganathan** (all from Hewlett-Packard).

Additional thanks goes to **Jason Bakos** for developing the new lecture slides.

I am grateful to the many instructors who have answered the publisher's surveys, reviewed our proposals, and attended focus groups to analyze and respond to our plans for this edition. They include the following individuals: Focus Groups in 2012: 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), Zachary Kurmas (Grand Valley State 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). Surveys and Reviews: Mahmoud Abou-Nasr (Wayne State University), Perry Alexander (The University of Kansas), 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), 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), Robert D. Cupper (Allegheny College), Edward W. Davis (North Carolina State University), Nathaniel J. Davis (Air Force Institute of Technology), Molisa Derk (Oklahoma City University), Derek Eager (University of Saskatchewan), Ernest Ferguson (Northwest Missouri State University), Rhonda Kay Gaede (The University of Alabama), Etienne M. Gagnon (UQAM), Costa Gerousis (Christopher Newport University), Paul Gillard (Memorial University of Newfoundland), Michael Goldweber (Xavier University), Georgia Grant (College of San Mateo), 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), Martin Herbordt (Boston University), Steve J. Hodges (Cabrillo College), Kenneth Hopkinson (Cornell 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), Robert N. Lea (University of Houston), 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), Bill Mark (University of Texas at Austin), Ananda Mondal (Claflin University), Alvin Moser (Seattle University), Walid Najjar (University of California, Riverside), Danial J. Neebel (Loras College), 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), Milos Prvulovic (Georgia Tech), Partha Ranganathan (HP Labs), Dejan Raskovic (University of Alaska, Fairbanks) Brad Richards (University of Puget Sound), Roman Rozanov, Louis Rubinfeld (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), 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), 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), Gerald D. Zarnett (Ryerson University), Nian Zhang (South Dakota School of Mines & Technology), Jiling Zhong (Troy University), Huiyang Zhou (The University of Central Florida), Weiyu Zhu (Illinois Wesleyan University).

A special thanks also goes to **Mark Smotherman** for making multiple passes to find technical and writing glitches that significantly improved the quality of this edition.

We wish to thank the extended Morgan Kaufmann family for agreeing to publish this book again under the able leadership of **Todd Green** and **Nate McFadden**: I certainly couldn't have completed the book without them. We also want to extend thanks to **Lisa Jones**, who managed the book production process, and **Russell Purdy**, who did the cover design. The new cover cleverly connects the PostPC Era content of this edition to the cover of the first edition.

The contributions of the nearly 150 people we mentioned here have helped make this fifth edition what I hope will be our best book yet. Enjoy!

David A. Patterson

This page intentionally left blank

1

*Civilization advances
by extending the
number of important
operations which we
can perform without
thinking about them.*

Alfred North Whitehead,
An Introduction to Mathematics, 1911

Computer Abstractions and Technology

- 1.1 Introduction** 3
- 1.2 Eight Great Ideas in Computer
Architecture** 11
- 1.3 Below Your Program** 13
- 1.4 Under the Covers** 16
- 1.5 Technologies for Building Processors and
Memory** 24

1.6	Performance	28
1.7	The Power Wall	40
1.8	The Sea Change: The Switch from Uniprocessors to Multiprocessors	43
1.9	Real Stuff: Benchmarking the Intel Core i7	46
1.10	Fallacies and Pitfalls	49
1.11	Concluding Remarks	52
	1.12 Historical Perspective and Further Reading	54
1.13	Exercises	54

1.1

Introduction

Welcome to this book! We're delighted to have this opportunity to convey the excitement of the world of computer systems. This is not a dry and dreary field, where progress is glacial and where new ideas atrophy from neglect. No! Computers are the product of the incredibly vibrant information technology industry, all aspects of which are responsible for almost 10% of the gross national product of the United States, and whose economy has become dependent in part on the rapid improvements in information technology promised by Moore's Law. This unusual industry embraces innovation at a breath-taking rate. In the last 30 years, there have been a number of new computers whose introduction appeared to revolutionize the computing industry; these revolutions were cut short only because someone else built an even better computer.

This race to innovate has led to unprecedented progress since the inception of electronic computing in the late 1940s. Had the transportation industry kept pace with the computer industry, for example, today we could travel from New York to London in a second for a penny. Take just a moment to contemplate how such an improvement would change society—living in Tahiti while working in San Francisco, going to Moscow for an evening at the Bolshoi Ballet—and you can appreciate the implications of such a change.

Computers have led to a third revolution for civilization, with the information revolution taking its place alongside the agricultural and the industrial revolutions. The resulting multiplication of humankind's intellectual strength and reach naturally has affected our everyday lives profoundly and changed the ways in which the search for new knowledge is carried out. There is now a new vein of scientific investigation, with computational scientists joining theoretical and experimental scientists in the exploration of new frontiers in astronomy, biology, chemistry, and physics, among others.

The computer revolution continues. Each time the cost of computing improves by another factor of 10, the opportunities for computers multiply. Applications that were economically infeasible suddenly become practical. In the recent past, the following applications were "computer science fiction."

- *Computers in automobiles:* Until microprocessors improved dramatically in price and performance in the early 1980s, computer control of cars was ludicrous. Today, computers reduce pollution, improve fuel efficiency via engine controls, and increase safety through blind spot warnings, lane departure warnings, moving object detection, and air bag inflation to protect occupants in a crash.
- *Cell phones:* Who would have dreamed that advances in computer systems would lead to more than half of the planet having mobile phones, allowing person-to-person communication to almost anyone anywhere in the world?
- *Human genome project:* The cost of computer equipment to map and analyze human DNA sequences was hundreds of millions of dollars. It's unlikely that anyone would have considered this project had the computer costs been 10 to 100 times higher, as they would have been 15 to 25 years earlier. Moreover, costs continue to drop; you will soon be able to acquire your own genome, allowing medical care to be tailored to you.
- *World Wide Web:* Not in existence at the time of the first edition of this book, the web has transformed our society. For many, the web has replaced libraries and newspapers.
- *Search engines:* As the content of the web grew in size and in value, finding relevant information became increasingly important. Today, many people rely on search engines for such a large part of their lives that it would be a hardship to go without them.

Clearly, advances in this technology now affect almost every aspect of our society. Hardware advances have allowed programmers to create wonderfully useful software, which explains why computers are omnipresent. Today's science fiction suggests tomorrow's killer applications: already on their way are glasses that augment reality, the cashless society, and cars that can drive themselves.

Classes of Computing Applications and Their Characteristics

Although a common set of hardware technologies (see Sections 1.4 and 1.5) is used in computers ranging from smart home appliances to cell phones to the largest supercomputers, these different applications have different design requirements and employ the core hardware technologies in different ways. Broadly speaking, computers are used in three different classes of applications.

Personal computers (PCs) are possibly the best known form of computing, which readers of this book have likely used extensively. Personal computers emphasize delivery of good performance to single users at low cost and usually execute third-party software. This class of computing drove the evolution of many computing technologies, which is only about 35 years old!

Servers are the modern form of what were once much larger computers, and are usually accessed only via a network. Servers are oriented to carrying large workloads, which may consist of either single complex applications—usually a scientific or engineering application—or handling many small jobs, such as would occur in building a large web server. These applications are usually based on software from another source (such as a database or simulation system), but are often modified or customized for a particular function. Servers are built from the same basic technology as desktop computers, but provide for greater computing, storage, and input/output capacity. In general, servers also place a greater emphasis on dependability, since a crash is usually more costly than it would be on a single-user PC.

Servers span the widest range in cost and capability. At the low end, a server may be little more than a desktop computer without a screen or keyboard and cost a thousand dollars. These low-end servers are typically used for file storage, small business applications, or simple web serving (see Section 6.10). At the other extreme are **supercomputers**, which at the present consist of tens of thousands of processors and many **terabytes** of memory, and cost tens to hundreds of millions of dollars. Supercomputers are usually used for high-end scientific and engineering calculations, such as weather forecasting, oil exploration, protein structure determination, and other large-scale problems. Although such supercomputers represent the peak of computing capability, they represent a relatively small fraction of the servers and a relatively small fraction of the overall computer market in terms of total revenue.

Embedded computers are the largest class of computers and span the widest range of applications and performance. Embedded computers include the microprocessors found in your car, the computers in a television set, and the networks of processors that control a modern airplane or cargo ship. Embedded computing systems are designed to run one application or one set of related applications that are normally integrated with the hardware and delivered as a single system; thus, despite the large number of embedded computers, most users never really see that they are using a computer!

personal computer (PC) A computer designed for use by an individual, usually incorporating a graphics display, a keyboard, and a mouse.

server A computer used for running larger programs for multiple users, often simultaneously, and typically accessed only via a network.

supercomputer A class of computers with the highest performance and cost; they are configured as servers and typically cost tens to hundreds of millions of dollars.

terabyte (TB) Originally 1,099,511,627,776 (2^{40}) bytes, although communications and secondary storage systems developers started using the term to mean 1,000,000,000,000 (10^{12}) bytes. To reduce confusion, we now use the term **tebibyte (TiB)** for 2^{40} bytes, defining **terabyte (TB)** to mean 10^{12} bytes. **Figure 1.1** shows the full range of decimal and binary values and names.

embedded computer A computer inside another device used for running one predetermined application or collection of software.

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
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%

FIGURE 1.1 The 2^x vs. 10^y bytes ambiguity was resolved by adding a binary notation for all the common size terms. In the last column we note how much larger the binary term is than its corresponding decimal term, which is compounded as we head down the chart. These prefixes work for bits as well as bytes, so *gigabit* (Gb) is 10^9 bits while *gibibits* (Gib) is 2^{30} bits.

Embedded applications often have unique application requirements that combine a minimum performance with stringent limitations on cost or power. For example, consider a music player: the processor need only be as fast as necessary to handle its limited function, and beyond that, minimizing cost and power are the most important objectives. Despite their low cost, embedded computers often have lower tolerance for failure, since the results can vary from upsetting (when your new television crashes) to devastating (such as might occur when the computer in a plane or cargo ship crashes). In consumer-oriented embedded applications, such as a digital home appliance, dependability is achieved primarily through simplicity—the emphasis is on doing one function as perfectly as possible. In large embedded systems, techniques of redundancy from the server world are often employed. Although this book focuses on general-purpose computers, most concepts apply directly, or with slight modifications, to embedded computers.

Elaboration: Elaborations are short sections used throughout the text to provide more detail on a particular subject that may be of interest. Disinterested readers may skip over an elaboration, since the subsequent material will never depend on the contents of the elaboration.

Many embedded processors are designed using *processor cores*, a version of a processor written in a hardware description language, such as Verilog or VHDL (see Chapter 4). The core allows a designer to integrate other application-specific hardware with the processor core for fabrication on a single chip.

Welcome to the PostPC Era

The continuing march of technology brings about generational changes in computer hardware that shake up the entire information technology industry. Since the last edition of the book we have undergone such a change, as significant in the past as the switch starting 30 years ago to personal computers. Replacing the

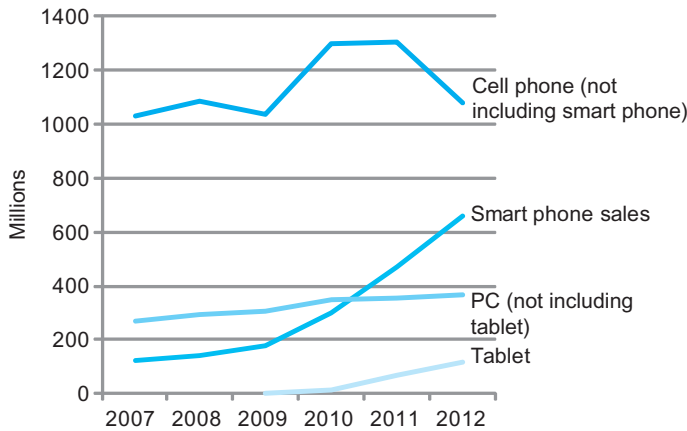


FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the PostPC era, versus personal computers and traditional cell phones. Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011. Tablets are the fastest growing category, nearly doubling between 2011 and 2012. Recent PCs and traditional cell phone categories are relatively flat or declining.

PC is the **personal mobile device (PMD)**. PMDs are battery operated with wireless connectivity to the Internet and typically cost hundreds of dollars, and, like PCs, users can download software (“apps”) to run on them. Unlike PCs, they no longer have a keyboard and mouse, and are more likely to rely on a touch-sensitive screen or even speech input. Today’s PMD is a smart phone or a tablet computer, but tomorrow it may include electronic glasses. Figure 1.2 shows the rapid growth time of tablets and smart phones versus that of PCs and traditional cell phones.

Taking over from the traditional server is **Cloud Computing**, which relies upon giant datacenters that are now known as *Warehouse Scale Computers* (WSCs). Companies like Amazon and Google build these WSCs containing 100,000 servers and then let companies rent portions of them so that they can provide software services to PMDs without having to build WSCs of their own. Indeed, **Software as a Service (SaaS)** deployed via the cloud is revolutionizing the software industry just as PMDs and WSCs are revolutionizing the hardware industry. Today’s software developers will often have a portion of their application that runs on the PMD and a portion that runs in the Cloud.

What You Can Learn in This Book

Successful programmers have always been concerned about the performance of their programs, because getting results to the user quickly is critical in creating successful software. In the 1960s and 1970s, a primary constraint on computer performance was the size of the computer’s memory. Thus, programmers often followed a simple credo: minimize memory space to make programs fast. In the

Personal mobile devices (PMDs)

are small wireless devices to connect to the Internet; they rely on batteries for power, and software is installed by downloading apps. Conventional examples are smart phones and tablets.

Cloud Computing refers to large collections of servers that provide services over the Internet; some providers rent dynamically varying numbers of servers as a utility.

Software as a Service (SaaS)

delivers software and data as a service over the Internet, usually via a thin program such as a browser that runs on local client devices, instead of binary code that must be installed, and runs wholly on that device. Examples include web search and social networking.

last decade, advances in computer design and memory technology have greatly reduced the importance of small memory size in most applications other than those in embedded computing systems.

Programmers interested in performance now need to understand the issues that have replaced the simple memory model of the 1960s: the parallel nature of processors and the hierarchical nature of memories. Moreover, as we explain in Section 1.7, today's programmers need to worry about energy efficiency of their programs running either on the PMD or in the Cloud, which also requires understanding what is below your code. Programmers who seek to build competitive versions of software will therefore need to increase their knowledge of computer organization.

We are honored to have the opportunity to explain what's inside this revolutionary machine, unraveling the software below your program and the hardware under the covers of your computer. By the time you complete this book, we believe you will be able to answer the following questions:

- How are programs written in a high-level language, such as C or Java, translated into the language of the hardware, and how does the hardware execute the resulting program? Comprehending these concepts forms the basis of understanding the aspects of both the hardware and software that affect program performance.
- What is the interface between the software and the hardware, and how does software instruct the hardware to perform needed functions? These concepts are vital to understanding how to write many kinds of software.
- What determines the performance of a program, and how can a programmer improve the performance? As we will see, this depends on the original program, the software translation of that program into the computer's language, and the effectiveness of the hardware in executing the program.
- What techniques can be used by hardware designers to improve performance? This book will introduce the basic concepts of modern computer design. The interested reader will find much more material on this topic in our advanced book, *Computer Architecture: A Quantitative Approach*.
- What techniques can be used by hardware designers to improve energy efficiency? What can the programmer do to help or hinder energy efficiency?
- What are the reasons for and the consequences of the recent switch from sequential processing to parallel processing? This book gives the motivation, describes the current hardware mechanisms to support parallelism, and surveys the new generation of “**multicore**” **microprocessors** (see Chapter 6).
- Since the first commercial computer in 1951, what great ideas did computer architects come up with that lay the foundation of modern computing?

multicore microprocessor

A microprocessor containing multiple processors (“cores”) in a single integrated circuit.

Without understanding the answers to these questions, improving the performance of your program on a modern computer or evaluating what features might make one computer better than another for a particular application will be a complex process of trial and error, rather than a scientific procedure driven by insight and analysis.

This first chapter lays the foundation for the rest of the book. It introduces the basic ideas and definitions, places the major components of software and hardware in perspective, shows how to evaluate performance and energy, introduces integrated circuits (the technology that fuels the computer revolution), and explains the shift to multicores.

In this chapter and later ones, you will likely see many new words, or words that you may have heard but are not sure what they mean. Don't panic! Yes, there is a lot of special terminology used in describing modern computers, but the terminology actually helps, since it enables us to describe precisely a function or capability. In addition, computer designers (including your authors) *love* using **acronyms**, which are *easy* to understand once you know what the letters stand for! To help you remember and locate terms, we have included a **highlighted** definition of every term in the margins the first time it appears in the text. After a short time of working with the terminology, you will be fluent, and your friends will be impressed as you correctly use acronyms such as BIOS, CPU, DIMM, DRAM, PCIe, SATA, and many others.

To reinforce how the software and hardware systems used to run a program will affect performance, we use a special section, *Understanding Program Performance*, throughout the book to summarize important insights into program performance. The first one appears below.

acronym A word constructed by taking the initial letters of a string of words. For example: **RAM** is an acronym for Random Access Memory, and **CPU** is an acronym for Central Processing Unit.

The performance of a program depends on a combination of the effectiveness of the algorithms used in the program, the software systems used to create and translate the program into machine instructions, and the effectiveness of the computer in executing those instructions, which may include input/output (I/O) operations. This table summarizes how the hardware and software affect performance.

Understanding Program Performance

Hardware or software component	How this component affects performance	Where is this topic covered?
Algorithm	Determines both the number of source-level statements and the number of I/O operations executed	Other books!
Programming language, compiler, and architecture	Determines the number of computer instructions for each source-level statement	Chapters 2 and 3
Processor and memory system	Determines how fast instructions can be executed	Chapters 4, 5, and 6
I/O system (hardware and operating system)	Determines how fast I/O operations may be executed	Chapters 4, 5, and 6