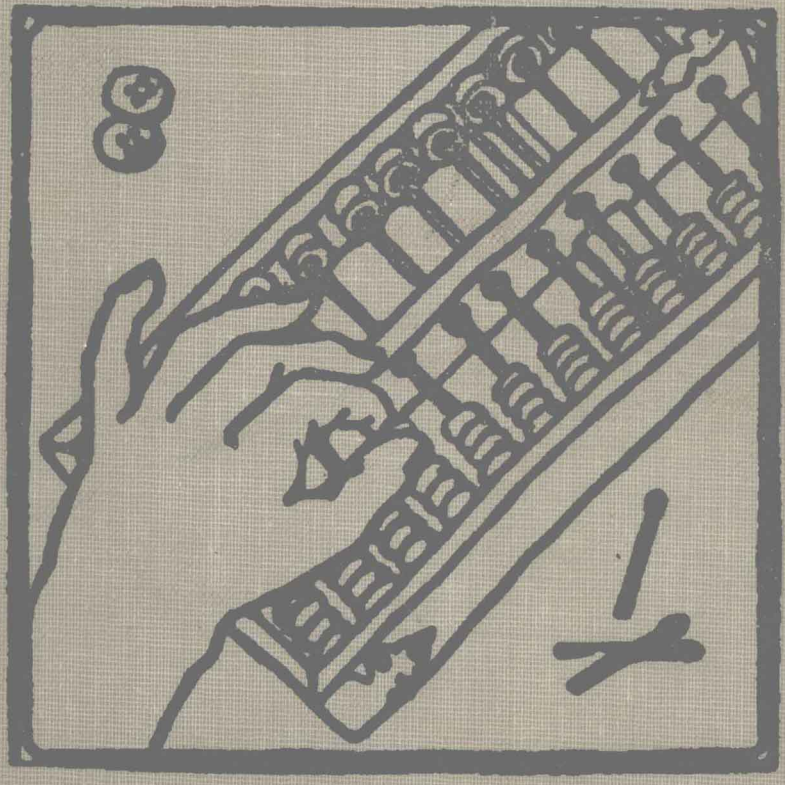
COMPUTER ORGANIZATION & DESIGN
THE HARDWARE/SOFTWARE INTERFACE



DAVID A. PATTERSON · JOHN L. HENNESSY

Computer Organization and Design

THE HARDWARE/SOFTWARE INTERFACE

John L. Hennessy
Stanford University

David A. Patterson
University of California at Berkeley

With a contribution by
James R. Larus
University of Wisconsin

Morgan Kaufmann Publishers, Inc.
San Francisco, California

Senior Editor: Bruce M. Spatz
Production Manager: Yonie Overton
Editorial Coordinator: Douglas Sery
Copyediting: Steve Hiatt and Gary Morris
Text Design: Ross Carron Design
Illustration: Alexander Teshin Associates
Composition/Color Separation/Postscript Programming: Edward W. Sznyter, Babel Press
Cover Design: David Lance Goines
Additional Cover Mechanical Art: Patty King
Chapter Opener Illustrations: Jo Jackson
Indexing: Steve Rath
Proofreading: Gary Morris
Electronic Prepress: The Courier Connection
Printer: Courier Corporation

Morgan Kaufmann Publishers, Inc.
Editorial Office:
340 Pine Street, Sixth Floor
San Francisco, CA 94104

Printed in the United States of America

97 96 95 94 5 4 3 2

Advice, Praise, and Errors: Any correspondence related to this publication or intended for the authors should be addressed to the editorial offices of Morgan Kaufmann Publishers, Inc., Dept. P&H APE. Information regarding error sightings is encouraged. Any error sightings that are accepted for correction in subsequent printings will be rewarded by the authors with a payment of $1.00 (U.S.) per correction at the time of their implementation in a reprint. Electronic mail can be sent to errors@cs.berkeley.edu.

Instructor Support: For information on the SPIM software simulator and other instructor materials available to adoptors, please contact the editorial offices of Morgan Kaufmann Publishers, Inc.

Cataloging-in-Publication Data

Patterson, David A.
Computer organization and design: the hardware/software interface
/ David A. Patterson, John L. Hennessy.
p. cm.
Includes bibliographical references and index.
ISBN 1-55860-281-X
1. Computer organization. 2. Computers--Design and construction.
3. Computer interfaces. I. Hennessy, John L. II. Title
QA76.9.C643P37 1994
004.2'2--dc20 94-17639
CIP

Computer Organization and Design

THE HARDWARE/SOFTWARE INTERFACE

TRADEMARKS

The following trademarks are the property of the following organizations:

TeX is a trademark of American Mathematical Society.

Apple II and Macintosh are trademarks of Apple Computer, Inc.

UNIX and UNIX F77 are trademarks of Novell.

The Cosmic Cube is a trademark of California Institute of Technology.

CP3100 is a trademark of Conner Peripherals.

CDC 6600, CDC 7600, CDC STAR-100, CYBER-180, CYBER-180/990, and CYBER-205 are trademarks of Control Data Corporation.

CRAY-1, CRAY-1S, CRAY-2, CRAY X-MP, CRAY X-MP/416, CRAY Y-MP, CFT-77 V3.0, CFT, CFT2 V1.3a, and T3D are trademarks of Cray Research.

Alpha, CVAX, DEC, DECsystem, DECstation, DECstation 3100, DEC system 10/20, fort, LP11, Massbus, MicroVAX-I, MicroVAX-II, PDP-8, PDP-10, PDP-11, RS-11M/IAS, Unibus, Ultrix, Ultrix 3.0, VAX, VAXstation, VAXstation 2000, VAXstation 3100, VAX-11, VAX 11/780, VAX-11/785, VAX Model 730, Model 750, Model 780, VAX 8600, VAX 8700, VAX 8800, VS FORTRAN V2.4, and VMS are trademarks of Digital Equipment Corporation.

Gnu C Compiler is a trademark of Free Software Foundation.

M2361A, Super Eagle, VP100, and VP200 are trademarks of Fujitsu Corporation.

Apollo DN 300, Apollo DN 10000, HP Precision Architecture, HPPA, HP 850, HP 3000, HP 3000/70, HP 9000, and Precision are trademarks of Hewlett-Packard Company.

432, 960 CA, 4004, 8008, 8080, 8086, 8087, 8088, 80186, 80286, 80386, 80486, Delta, iAPX 432, i860, iPSC, iPSC/2, Intel, Intel Hypercube, Multibus, Multibus II, and Paragon are trademarks of Intel Corporation.

Inmos and Transputer are trademarks of Inmos.

IBM, 360, 360/30, 360/40, 360/50, 360/65, 360/85, 360/91, 370, 370/135, 370/138, 370/145, 370/155, 370/158, 370/165, 370/168, 370-XA, ESA/370, System/360, System/370, 701, 704, 709, 801, 3033, 3080, 3080 series, 3080 VF, 3081, 3090, 3090/100, 3090/200, 3090/400, 3090/600, 3090/600S, 3090 VF, 3330, 3380, 3380D, 3380 Disk Model AK4, 3380J, 3390, 3880-23, 3990, 7030, 7090, 7094, IBM FORTRAN, ISAM, MVS, IBM PC, IBM PC-AT, PL.8, PowerPC, RT-PC, SAGE, Stretch, IBM SVS, Vector Facility, and VM are trademarks of International Business Machines Corporation.

FutureBus is a trademark of the Institute of Electrical and Electronic Engineers.

Goodyear MPP is a trademark of Goodyear Tire and Rubber Co, Inc.

ICL DAP is a trademark of International Computers Limited.

KSR-1 is a trademark of Kendall Square Research.

MASPAR MP-1 is a trademark of MasPar Corporation.

NuBus is a trademark of Massachusetts Institute of Technology.

MIPS, MIPS 120, MIPS/120A, M/500, M/1000, RC6230, RC6280, R2000, R2000A, R2010, R3000, R3010, and R4000 are trademarks of MIPS Technology, Inc.

Delta Series 8608, System V/88 R32V1, VME bus, 6809, 68000, 68010, 68020, 68030, 68882, 88000, 88000 1.8.4m14, 88100, and 88200 are trademarks of Motorola Corporation.

Ncube and nCube/ten are trademarks of Ncube Corporation.

Parsytec GC is a trademark of Parsytec, Inc.

Wren IV, Imprimis, Sabre, Sabre 97209, and IPI-2 are trademarks of Seagate Corporation.

Sequent, Balance 800, Balance 21000, and Symmetry are trademarks of Sequent Computers.

Silicon Graphics 4D/60, 4D/240, and Silicon Graphics 4D Series are trademarks of Silicon Graphics.

Connection Machine, CM-2, and CM-5 are trademarks of Thinking Machines.

Burroughs 6500, B5000, B5500, D-machine, UNIVAC, UNIVAC I, UNIVAC 1103 are trademarks of UNISYS.

Spice and 4.2 BSD UNIX are trademarks of University of California at Berkeley.

Alto, Ethernet, PARC, Palo Alto Research Center, Smalltalk, and Xerox are trademarks of Xerox Corporation.

TO LINDA AND ANDREA

Foreword

by Maurice V. Wilkes
Cambridge, England

This is an excellent time for a new book on computer design. During the last ten years the subject has undergone a marked renaissance. It has become less dependent on intuition and personal opinion, and more on measurement and rational analysis. The subtitle of the author's earlier book *Computer Architecture: A Quantitative Approach* makes this point.

Patterson and Hennessy played their part in these developments. Indeed, widespread public discussion of the new ideas may be said to have begun with the publication in 1980 of a paper by Patterson and Ditzel entitled "The Case for the Reduced Instruction Set Computer." The acronym RISC derives from this paper. Patterson proceeded to put RISC ideas into practice by developing the Berkeley RISC with the aid of a group of students. This design became the basis of the SPARC workstations. Hennessy applied his energies to the MIPS design project at Stanford and became one of the founders of the MIPS Computer company.

Like many seminal ideas, RISC is deceptively simple. The emphasis is not on the size of the instruction set, but on its nature; RISC instruction sets have made it possible to apply, in a single chip processor, subtle techniques for instruction level concurrency that were previously to be found only in large computers.

There is a lot more to computer design than is comprised in the RISC philosophy, as will be shown by a glance at the diagram entitled "The Five Classic Components of a Computer" which appears, with differing highlighting, at the head of various chapters of this book. But RISC has been a unifying influence. Another major unifying influence has been the need to work within the boundaries of a silicon chip, where there is never enough space and, if one thing goes in, another must go out. Performance depends critically on decisions taken at the chip level. No longer can system design be a subject divorced from computer implementation, a change which may have left some people high and dry, but which is nevertheless wholly to the good. Nor can system design be divorced from consideration of the software interface, a point which the authors bring out both in their subtitle and in their text.

The computer field, particularly on the software side, abounds with examples of new ideas that have found their way into industrial practice by being taught in universities and have thereby become part of the professional tool kit

which graduating students have carried with them into industry. In hardware, it is often the other way round; teaching follows practice. This is another reason for welcoming a book by two engineers who can write of current practice with authority.

As I followed the authors through their unhurried chapters, I was conscious of the all-seeing eye of the evaluator pictured at the chapter heads. Ever watchful, she—I think it is a female eye, but I cannot be sure—seemed to be saying that every argument has another side, and that every insight gains by being put into perspective.

I think students will enjoy learning from this book; at least, I hope so and I give them my good wishes.

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, 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 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.

Traditionally, the competing influences of assembly language, organization, and design have encouraged books that consider each area as a distinct subset. In our view, such distinctions have increasingly lost meaning as computer technology has advanced. To truly understand the breadth of our field, it is important to understand the interdependencies among these topics.

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.

Relationship to CA:AQA

Many readers will be familiar with *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann, 1990). Our motivation in writing that 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 scientific methodologies instead of a descriptive approach.

We've discovered that many people have used that book as a first introduction to the field. We've also learned that many institutions are now using this quantitative approach in more introductory computer organization courses. However, *Computer Architecture* was written at a more advanced level, for readers who already understood the basic principles.

A majority of the readers for *Computer Organization and Design: The Hardware/Software Interface* will not be or plan to become computer architects. However, the performance of future software systems will be dramatically affected 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.

Given these factors, we knew that this book had to be much more than a subset of the material in *Computer Architecture*. We've approached every topic in a new way. Topics shared between the books were written anew for this effort, while many other topics are presented here for the first time. To further ensure the uniqueness of *Computer Organization and Design*, we exchanged the writing responsibilities we assigned to ourselves for *Computer Architecture*. The topics that Hennessy covered in the first book were written by Patterson in this one, and vice versa. Several of our reviewers suggested that we call this book "Computer Organization: A Conceptual Approach" to emphasize the significant differences from our other book. It is our hope that the reader will find new insights in every section, as well as a more tractable introduction to the abstractions and principles at work in a modern computer.

Learning by Evolution

It is tempting for authors to present the latest version of a hardware concept and spend considerable time explaining how these often sophisticated ideas work. We decided instead to present each idea from its first principles, emphasizing the simplest version of an idea, how it works, and how it came to be. We believe that presenting the fundamental concepts first offers greater insight into why machines look the way they do today, as well as how they might evolve as technology changes.

To facilitate this approach, we have based the book upon the MIPS processor. It offers an easy to understand instruction set and can be implemented in a simple, straightforward manner. This allows readers to grasp an entire machine organization and to follow exactly how the machine implements its instructions. Throughout the text, we present the concepts before the details, building from simpler versions of ideas to more complex ones. Examples of this approach can be found in almost every chapter. Chapter 3 builds up to MIPS assembly language starting with one simple instruction type. The con-

cepts and algorithms used in modern computer arithmetic are built up starting from the familiar grammar school algorithms in Chapter 4. Chapters 5 and 6 start from the simplest possible implementation of a MIPS subset and build to a fully pipelined version. Chapter 7 illustrates the abstractions and concepts in memory hierarchies by starting with the simplest possible cache and introducing virtual memory and TLBs as an extension of the concepts.

This evolutionary process is used extensively in Chapters 5 and 6, where the complete datapath and control for a processor are presented. Since learning is a visual process, we have included sequences of figures that contain progressively more detail or show a sequence of events within the machine. We have also used a second color to help readers follow the figures and sequences of figures.

Learning from this Book

Our objective of demonstrating first principles through the interrelationship of hardware and software is enhanced by several features found in each chapter. The Hardware/Software Interface sections are used to highlight these relationships. We've also included Big Picture sections for each chapter to remind readers of the major insights. We hope that these elements reinforce our goal of making this book equally valuable as a foundation for further study in both hardware and software courses.

To illustrate the relationship between high-level language and machine language and to describe the hardware algorithms, we have chosen C. It is widely used in compiler and operating system courses, it is widely used by computer professionals, and several facilities in the language make it suitable for describing hardware algorithms. For those who are familiar with Pascal rather than C, Appendix D provides a quick introduction to C for Pascal programmers and should be sufficient to understand the code sequences in the text.

We have tried to manage the pace of the presentation for readers of varying experience. Ideas that are not essential to a newcomer, but which may be of interest to the more advanced reader, are presented as Elaborations. When appropriate, advanced concepts have been saved for the exercise sets and enhanced with additional discussion as In More Depth sections. In addition, we found that the extent of background that students have in logic design varies widely. Thus, Appendix B provides all the necessary background for those readers not versed in the basics of logic design, as well as some slightly more sophisticated material for the more advanced student. Within a course, this material can be used in supplementary lectures or incorporated into the mainstream of the course, depending on the background of the students and the goals of the instructor.

We have also found that readers enjoy learning the history of the field, so the Historical Perspective sections include many photographs of important machines and little known stories about the ideas behind them. We hope that the perspective offered by these anecdotes and photographs will add a new dimension for our readers.

Course Syllabi and this Book

One particularly difficult issue facing instructors is the balance of assembly language programming with computer organization. We have written this book so that readers will learn more about organization and design, while still providing a complete introduction to assembly language. By using a RISC architecture, students can learn the basics of an instruction set and assembly language programming in less time than is typically reserved in the curriculum for CISC based assembly courses. Many instructors have also found that using a simulator, rather than running in native mode on a real machine, provides the experience of assembly language programming in substantially less time (and with less pain for the student).

Instructors may contact the publisher regarding the SPIM simulator of the MIPS processor. The XSPIM simulator developed by James R. Larus is retrievable via ftp (see page xxiii). Adaptations are also available in Windows and Macintosh formats. Although not identical, they offer the same general functionality. We feel this will enhance student opportunities for learning about computer organization (see Appendix A). Finally, stepwise derivation of assembly from a high-level language takes less study time than learning it from the ground up. Chapter 3 and Appendix A may be used together or separately, depending upon the reader's background. Chapter 3 provides the basics and can be supplemented with additional detail from Appendix A for a complete introduction to modern assembly language programming, including assemblers, linkers, and loaders. In the end, we hope this approach offers a more efficient treatment of assembly for most readers, while being sufficiently broad to support detailed lecture or laboratory coverage if an instructor wants more emphasis on assembly language programming.

For those courses intended to expose students to the important principles of computer organization, the chapters from 4 to 9 explain the key ideas. Chapter 4 explains the idea of number representation for both integers and floating-point numbers and shows how arithmetic algorithms work. Chapters 5 and 6 introduce key ideas in control and pipelining and can be covered at several levels. Chapter 7 introduces the principles of memory hierarchies, unifying the ideas of caching and virtual memory. Chapter 8 shows how I/O systems are organized and controlled, explaining the cooperative relationship between the hardware and the operating system. Finally, Chapter 9 uses examples to introduce the key principles used in multiprocessors.

For readers who want a greater emphasis on computer design, Chapters 4 through 6, together with Appendices B and C, provide that opportunity. For example, Chapter 4 explains a number of techniques used by computer designers to speed up addition and multiplication. Chapters 5 and 6 derive complete implementations of a MIPS subset using the arithmetic elements from Chapter 4 and a number of common datapath elements (such as register files and memories) that are explained in detail in Appendix B. Chapter 5 starts with a very simple implementation; a complete datapath and control unit are constructed for this organization. The implementation is then modified to derive a faster version where each instruction can take differing numbers of clock

cycles. The control for this multicycle implementation is designed using two different methods in Chapter 5. Appendix C shows in detail how the control specifications are implemented using structured logic blocks. Chapter 6 builds on the single-clock cycle implementation created in Chapter 5 to show how pipelined machines are designed. The design is extended to show how hazards can be handled and how control for interrupts works. The student interested in computer design, is not only exposed to three different designs for the same instruction set, but can also see how these designs compare in terms of advantages and disadvantages.

Chapter Organization and Overview

Using these plans as the core, we developed the other chapters to introduce and support that core.

Many students remarked that they appreciated learning about the continuing rapid change in speed and capacity of hardware, as well as some of the history of computer development. This material is the focus of Chapter l. It provides a perspective on how software or hardware will need to scale during the coming decades. Chapter 1 also introduces topics to be covered in later chapters.

Chapter 2 shows that time is the only safe measure of computer performance. It also relates common measurements used by hardware and software designers to the reliable measurement of time. The material in this chapter motivates the techniques discussed in Chapters 5, 6, and 7 and provides a framework for evaluating them.

Chapter 3 builds on the knowledge of a programming language to derive an assembly language, offering several rules of thumb that guide the designer of the assembly language. We chose the instruction set of a real computer, in this case MIPS, so that real compilers could be used by students to see the code that would be generated. We hide the delayed branch and load until Chapter 6 for pedagogical reasons. Fortunately, the MIPS assembler schedules both delayed branches and loads so the assembly language programmer can ignore these complexities without danger. Readers interested in seeing a very different approach to instruction set design should read Appendix E, which gives a short introduction to the VAX architecture using the same major programming example as in Chapter 3.

Although there is no consensus on what should be covered or what should be skipped in learning about computer arithmetic, we couldn't write Chapter 4 without reaching some conclusions of our own! We understand that the topics and depth of coverage vary greatly from one course to another, sometimes within the same department, depending upon the taste and background of the individual instructor. For example, some instructors feel it's essential that everyone learn multibit Booth algorithms, while others will skip signed multiplication. Our solution is to introduce all the central ideas in the chapter and to provide additional background for more advanced topics in the exercises. This allows one instructor to cover more advanced topics and assign exercises based on them, while another instructor may skip the material.

Chapters 5 and 6 show a realistic example of a processor in detail. Most readers appreciate having a real example to study, and a complete example provides the insight needed to see how all the pieces of a processor fit together for a pipelined and nonpipelined machine. To facilitate skipping some details on hardware implementation of control, we have included much of this material in Appendix C.

Just as Chapters 2 through 6 provide important background for readers with an interest in compilers, Chapters 7 and 8 provide vital background to anyone pursuing further work in operating systems or databases. Chapter 7 describes the principles of memory hierarchies, focusing on the commonality between virtual memory and caching. Chapter 7 emphasizes the role of the operating system and its interaction with the memory system.

Topics as diverse as operating systems, databases, graphics, and networking require an understanding of I/O systems organization as well as the major technical characteristics of devices that influence this organization. Chapter 8 focuses on the topic of how I/O systems are organized starting with bus organizations, working up to communication between the processor and I/O device, and finally to the management role of the operating system. While we emphasize the interfacing issues, especially between hardware and software, several other important topics are introduced. Many of these topics are useful not only in computer organization but as background in other areas. For example, the handshaking protocol, used to interface asynchronous I/O devices, has applications in any distributed system.

For some readers, this book may be their only overview of computer systems, so we have included a survey of parallel processing. Rather than the traditional catalog of characteristics for many parallel machines, we have tried to describe the underlying principles that will drive the designs of parallel processors for the next decade. This section includes a small running example to show different versions of the same program for different parallel architectures.

Because the book is intended as an introduction for readers with a variety of interests, we tried to keep the presentation flexible. The appendices on assembly language and logic design are one of the principle vehicles to allow such flexibility, as these are easily skipped by more advanced readers. The presence of the appendices has made it possible to use this book in a course that mixes EE and CS majors with fairly different backgrounds in logic design and software.

Assembly language programming is best learned by doing and in many cases will be done with the use of the simulator available with this book. Because of this, we invited Jim Larus, the creator of the SPIM simulator, to join us as a contributor of Appendix A. Appendix A describes the SPIM simulator and provides further details of the MIPS assembly language. In addition, it describes assemblers and linkers, which handle the translation of assembly language programs to executable machine language.

The logic design appendix is intended as a supplement to the material on computer organization rather than a comprehensive introduction to logic design. While many EE students in a computer organization course will have al-

ready had a course on logic design or digital electronics, we have found that CS majors in many institutions have not had much exposure to this area. The first few sections of Appendix B provide the necessary background. We include some material, such as the organization of memories and finite state machine control of a processor, in the mainstream material, since it is crucial to understanding computer organization.

Selection of Material

If you had no prior background and wanted to read from cover-to-cover, the following order makes sense: Chapters 1 and 2, Appendix D (if needed), Chapter 3, Appendices A and E, Chapter 4, Appendix B, Chapter 5, Appendix C, Chapters 6, 7, 8, and 9. Clearly, most readers skip material. We have worked to provide readers with flexibility in their approach to the material, without making the discussions redundant. The chapters have been written as self-contained units with cross-references to other chapters when related text or figures should be considered. The book has been used successfully in a variety of Beta courses with different goals and student backgrounds. Specific choice of materials as well as the sequence of presentation varied significantly among the Beta sites. Table 1 samples some of these differences.

Students	EE/CS soph/jr	CS jr/sr	CS soph/jr	EE sr/gr	EE/CS jr/sr	EE/CS jr/sr
Prerequisites	HLL	Assembly	Assembly HLL	Assembly Some logic	Assembly Digital fund.	Assembly Digital design
Term (in weeks)	10	14	15	10	16	10
1 Introduction	1	—	2	1	Reference	1
2 Performance	2	—	3	2	Reference	2
3 Instructions	3 (p)	4 (p)	4	3	1	3 (p)
4 Arithmetic	4	2	6	4	2/6	4
5 Processor	5	3	7	5 (p)	3/5	5
6 Pipelining	6	5	8 (p)	6	7	6
7 Memory	7	6	9	7	8	7
8 I/O	8	7	10	8	9	8 (p)
9 Parallel		8	11	9 (p)		
A Assembly	3	Reference				
B Logic	Reference	1	5			
C Control	Reference				4	
Other topics	VAX (App E)		1 C language		RISC machines	

Table 1 (p) = partial coverage or cursory. Numbers refer to the sequence of chapter coverage. Numbers separated by / indicates chapter was covered in parts out of sequence.

Concluding Remarks

In our last book we alternated the gender of a pronoun chapter by chapter. In this book we believe we have removed all such pronouns, except of course for specific people.

If you read the following acknowledgement 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 errors@cs.berkeley.edu or by low-tech mail using the address found on the copyright page. The first person to report a technical error will be awarded a $1.00 bounty upon its implementation in future printings of the book!

Finally, like the last book there is no strict ordering of the authors' names. About half the time you will see Hennessy and Patterson, both in this book and in advertisements, and half the time you will see Patterson and Hennessy. You'll even find it listed both ways in bibliographic publications such as *Books In Print*. This again reflects the true collaborative nature of this book: Together we brainstormed about the ideas and method of presentation, then individually wrote about one-half of the chapters and acted as reviewer for every draft of the other. The page count suggests we again wrote almost exactly the same number of pages. Thus, we equally share the blame for what you are about to read.

Acknowledgements

We wish first to acknowledge the encouragement and suggestions offered by the readers of *Computer Architecture: A Quantitative Approach* and the reviewers of the proposal originally produced for this book. We would not have written this book without their support and directions.

Before we started this book, we received valuable comments on an outline of our ideas from

Alan Berenbaum, AT&T; **Douglas W. Clark**, Digital Equipment Corporation/Princeton University; **David Culler**, University of California at Berkeley; **Stephen J. Hartley**, University of Texas at San Antonio; **Monica Lam**, Stanford; **Daniel McCrackin**, McMaster University; **William R. Michalson**, Worcester Polytechnic Institute; **Yuval Tamir**, University of California at Los Angeles; **Philip A. Wilsey**, University of Cincinnati

The early comments from these reviewers convinced us that there was a need for a book with the goals we have used for this effort.

We'd like to express our appreciation to **Jim Larus** for his willingness in contributing his expertise on assembly language programming, as well as for welcoming readers of this book to use the simulator he developed and maintains at the University of Wisconsin.

Thanks go to about 50 students at Berkeley taking CS 152 during Spring semester 1992 and about 80 students at Stanford taking CS 182 during Winter Quarter 1992 for debugging the alpha version of the text. Professors **John**

Wawrzynek and **John Hennessy** taught the two courses. **Jeff Kuskin**, who served as the teaching assistant at Stanford, provided valuable advice and generated the original versions of a number of exercises that appear in this book.

In addition to the student comments, we appreciate the feedback from these reviewers of the alpha version:

Alan Berenbaum, AT&T; **Douglas W. Clark**, Digital Equipment Corporation/Princeton University; **Rajan Chandra**, California State Polytechnic University at Pomona; **Edward W. Czeck**, Northeastern University; **Chris Edmondson**, Yurkanan University of Texas at Austin; **Robert Fowler**, University of Rochester; **Gideon Frieder**, George Washington University (Chapter 1); **Mark Hill**, University of Wisconsin at Madison; **Kai Li**, Princeton University; **Bart Locanthi**, AT&T (Chapter 8); **David Meyer**, Purdue University; **William R. Michalson**, Worcester Polytechnic Institute; **Mark Smotherman**, Clemson University ; **Evan Tick**, University of Oregon (careful review of figures); **Shlomo Weiss**, Tel Aviv University (Chapter 8); **Alan Zaring**, Ohio Wesleyan University (Chapter 9 and Appendix B)

Mark Smotherman's comments on the role of assembly language were especially helpful in deciding how to deal with this topic. Many of the reviewers provided helpful suggestions for exercises. **William Kahn** of UC Berkeley provided the material for the history section for the computer arithmetic chapter.

The Beta reviewers included:

David Douglas, Thinking Machines (Chapter 9); **Alan Fekete**, University of Sydney; **Corinna Lee**, University of Toronto; **William R. Michalson** Worcester Polytechnic Institute; **Ned Okie**, Radford University; **Klaus Erik Schauser**, University of California at Berkeley; **Guri Sohi**, University of Wisconsin at Madison (Chapter 9); **Arun Somani**, University of Washington; **Philp Tromovitch**, SUNY at Stony Brook; **David Ward**, Brigham Young University; **James Van Orman**, Brigham Young University (who provided extensive figure review both in the Beta and for the final edition)

Special thanks go to **Doug Clark** for his inputs on both the Alpha and Beta versions. As with *Computer Architecture*, Doug provided a wealth of comments to us. His insights, as well as his persistence in urging us to simplify and improve the pedagogy, are deeply appreciated.

The Beta Edition was released for class testing in the Fall of 1992 by the following instructors and institutions:

Rajendra Boppana, University of Texas at San Antonio; **Barry S. Fagin,** Dartmouth; **Michael Faiman**, Univ. of Illinois, Urbana–Champaign; **Mark A. Friedman,** Trinity College; **Anoop Gupta,** Stanford University; **Brian Harvey,** University of California at Berkeley; **Roy Jenevein**, University of Texas at Austin; **Corinna Lee,** University of Toronto; **Ned Okie,** Radford University; **Parameswaran Ramanathan,** University of Wisconsin at Madison; **Arun K. Somani,** University of Washington; **David M. Ward,** Brigham Young University; **John Wawrzynek,** University of California at Berkeley