

COMPUTER ARCHITECTURE

A Quantitative Approach



MK
HUBBARD SAUTMAN

John L. Hennessy and David A. Patterson

Computer Architecture Formulas

$$1. \text{ CPU time} = \text{Instruction count} \times \text{Clock cycles per instruction} \times \text{Clock cycle time}$$

$$2. \text{ Amdahl's Law: Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$3. \text{ Power}_{\text{dynamic}} = 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

$$4. \text{ Power}_{\text{static}} = \text{Current}_{\text{static}} \times \text{Voltage}$$

$$5. \text{ Average memory-access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$6. \text{ Availability} = \text{Mean Time To Fail} / (\text{Mean Time To Fail} + \text{Mean Time to Repair})$$

$$7. \text{ Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

where Wafer yield accounts for wafers that are so bad they need not be tested and α is a fitted parameter that approximates the number of masking levels critical to die yield (usually $\alpha \approx 4.0$)

$$8. \text{ Average memory-access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$9. \text{ Misses per instruction} = \text{Miss rate} \times \text{Memory access per instruction}$$

$$10. \text{ Cache index size: } 2^{\text{index}} = \text{Cache size} / (\text{Block size} \times \text{Set associativity})$$

11. Means—arithmetic (AM), weighted arithmetic (WAM), and geometric (GM):

$$\text{AM} = \frac{1}{n} \sum_{i=1}^n \text{Time}_i \quad \text{WAM} = \sum_{i=1}^n \text{Weight}_i \times \text{Time}_i \quad \text{GM} = \sqrt[n]{\prod_{i=1}^n \text{Time}_i} = \exp\left(\frac{1}{n} \times \sum_{i=1}^n \ln(\text{Time}_i)\right)$$

where Time_i is the execution time for the i th program of a total of n in the workload, Weight_i is the weighting of the i th program in the workload.

$$12. \text{ Geometric standard deviation} = \exp\left(\sqrt{\frac{\sum_{i=1}^n (\ln(\text{Time}_i) - \ln(\text{Geometric mean}))^2}{n}}\right)$$

Rules of Thumb

1. **Amdahl/Case Rule:** A balanced computer system needs about 1 MB of main memory capacity and 1 megabit per second of I/O bandwidth per MIPS of CPU performance.
2. **90/10 Locality Rule:** A program executes about 90% of its instructions in 10% of its code.
3. **Bandwidth Rule:** Bandwidth grows by at least the square of the improvement in latency.
4. **2:1 Cache Rule:** The miss rate of a direct-mapped cache of size N is about the same as a two-way set-associative cache of size $N/2$.
5. **Dependability Rule:** Design with no single point of failure.

John L. Hennessy is the president of Stanford University, where he has been a member of the faculty since 1977 in the departments of electrical engineering and computer science. Hennessy is a Fellow of the IEEE and ACM, a member of the National Academy of Engineering and the National Academy of Science, 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 one-year leave from the university to cofound MIPS Computer Systems, which developed one of the first commercial RISC microprocessors. After being acquired by Silicon Graphics in 1991, MIPS Technologies became an independent company in 1998, focusing on microprocessors for the embedded marketplace. As of 2006, over 500 million MIPS microprocessors have been shipped in devices ranging from video games and palmtop computers to laser printers and network switches.

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 Abacus Award from Upsilon Pi Epsilon, 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 for contributions to RISC and shared the IEEE Johnson Information Storage Award for contributions to RAID. He then 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, 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 a Distinguished Service Award from CRA.

At Berkeley, Patterson led the design and implementation of RISC I, likely the first VLSI reduced instruction set computer. This research became the foundation of the SPARC architecture, currently used by Sun Microsystems, Fujitsu, and others. He was a leader of the Redundant Arrays of Inexpensive Disks (RAID) project, which led to 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. These projects earned three dissertation awards from the ACM. His current research projects are the RAD Lab, which is inventing technology for reliable, adaptive, distributed Internet services, and the Research Accelerator for Multiple Processors (RAMP) project, which is developing and distributing low-cost, highly scalable, parallel computers based on FPGAs and open-source hardware and software.

Computer Architecture

A Quantitative Approach

Fourth Edition

John L. Hennessy

Stanford University

David A. Patterson

University of California at Berkeley

With Contributions by

Andrea C. Arpaci-Dusseau
University of Wisconsin-Madison

Remzi H. Arpaci-Dusseau
University of Wisconsin-Madison

Krste Asanovic
Massachusetts Institute of Technology

Robert P. Colwell
R&E Colwell & Associates, Inc.

Thomas M. Conte
North Carolina State University

Jose Duato
Universitat Politecnica de Valencia and Simula

Diana Franklin
California Polytechnic State University, San Luis Obispo

David Goldberg
Xerox Palo Alto Research Center

Wen-mei W. Hwu
University of Illinois at Urbana-Champaign

Norman P. Jouppi
HP Labs

Timothy M. Pinkston
University of Southern California

John W. Sias
University of Illinois at Urbana-Champaign

David A. Wood
University of Wisconsin-Madison



Amsterdam • Boston • Heidelberg • London
New York • Oxford • Paris • San Diego
San Francisco • Singapore • Sydney • Tokyo



MORGAN KAUFMANN PUBLISHERS

Publisher Denise E. M. Penrose
Project Manager Dusty Friedman, The Book Company
In-house Senior Project Manager Brandy Lilly
Developmental Editor Nate McFadden
Editorial Assistant Kimberlee Honjo
Cover Design Elisabeth Beller and Ross Carron Design
Cover Image Richard I'Anson's Collection: Lonely Planet Images
Composition Nancy Logan
Text Design: Rebecca Evans & Associates
Technical Illustration David Ruppe, Impact Publications
Copy editor Ken Delia Penta
Proofreader Jamie Thaman
Indexer Nancy Ball
Printer Maple-Vail Book Manufacturing Group

Morgan Kaufmann Publishers is an Imprint of Elsevier
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

© 1990, 1996, 2003, 2007 by Elsevier, Inc.
All rights reserved.

Published 1990. Fourth edition 2007

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com. You may also complete your request on-line via the Elsevier Science homepage (<http://elsevier.com>), by selecting "Customer Support" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data

Hennessy, John L.

Computer architecture : a quantitative approach / John L. Hennessy, David A. Patterson ; with contributions by Andrea C. Arpaci-Dusseau . . . [et al.].
—4th ed.

p.cm.

Includes bibliographical references and index.

ISBN 13: 978-0-12-370490-0 (pbk. : alk. paper)

ISBN 10: 0-12-370490-1 (pbk. : alk. paper) 1. Computer architecture. I. Patterson, David A. II. Arpaci-Dusseau, Andrea C. III. Title.

QA76.9A73P377 2006

004.2'2—dc22

2006024358

For all information on all Morgan Kaufmann publications,
visit our website at www.mkp.com or www.books.elsevier.com

Printed in the United States of America

06 07 08 09 10 5 4 3 2 1

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

data loaded from memory. In the case of a store, the second register operand is the source of the data that is stored into memory. The instructions load word (LD) and store word (SD) load or store the entire 64-bit register contents.

3. *Branches and jumps*—Branches are conditional transfers of control. There are usually two ways of specifying the branch condition in RISC architectures: with a set of condition bits (sometimes called a condition code) or by a limited set of comparisons between a pair of registers or between a register and zero. MIPS uses the latter. For this appendix, we consider only comparisons for equality between two registers. In all RISC architectures, the branch destination is obtained by adding a sign-extended offset (16 bits in MIPS) to the current PC. Unconditional jumps are provided in many RISC architectures, but we will not cover jumps in this appendix.

A Simple Implementation of a RISC Instruction Set

To understand how a RISC instruction set can be implemented in a pipelined fashion, we need to understand how it is implemented *without* pipelining. This section shows a simple implementation where every instruction takes at most 5 clock cycles. We will extend this basic implementation to a pipelined version, resulting in a much lower CPI. Our unpipelined implementation is not the most economical or the highest-performance implementation without pipelining. Instead, it is designed to lead naturally to a pipelined implementation. Implementing the instruction set requires the introduction of several temporary registers that are not part of the architecture; these are introduced in this section to simplify pipelining. Our implementation will focus only on a pipeline for an integer subset of a RISC architecture that consists of load-store word, branch, and integer ALU operations.

Every instruction in this RISC subset can be implemented in at most 5 clock cycles. The 5 clock cycles are as follows.

1. *Instruction fetch cycle (IF):*

Send the program counter (PC) to memory and fetch the current instruction from memory. Update the PC to the next sequential PC by adding 4 (since each instruction is 4 bytes) to the PC.

2. *Instruction decode/register fetch cycle (ID):*

Decode the instruction and read the registers corresponding to register source specifiers from the register file. Do the equality test on the registers as they are read, for a possible branch. Sign-extend the offset field of the instruction in case it is needed. Compute the possible branch target address by adding the sign-extended offset to the incremented PC. In an aggressive implementation, which we explore later, the branch can be completed at the end of this stage, by storing the branch-target address into the PC, if the condition test yielded true.

Decoding is done in parallel with reading registers, which is possible because the register specifiers are at a fixed location in a RISC architecture.

the instruction to continue. In the VAX an additional bit of state records when an instruction has started updating the memory state, so that when the pipeline is restarted, the CPU knows whether to restart the instruction from the beginning or from the middle of the instruction. The IA-32 string instructions also use the registers as working storage, so that saving and restoring the registers saves and restores the state of such instructions.

A different set of difficulties arises from odd bits of state that may create additional pipeline hazards or may require extra hardware to save and restore. Condition codes are a good example of this. Many processors set the condition codes implicitly as part of the instruction. This approach has advantages, since condition codes decouple the evaluation of the condition from the actual branch. However, implicitly set condition codes can cause difficulties in scheduling any pipeline delays between setting the condition code and the branch, since most instructions set the condition code and cannot be used in the delay slots between the condition evaluation and the branch.

Additionally, in processors with condition codes, the processor must decide when the branch condition is fixed. This involves finding out when the condition code has been set for the last time before the branch. In most processors with implicitly set condition codes, this is done by delaying the branch condition evaluation until all previous instructions have had a chance to set the condition code.

Of course, architectures with explicitly set condition codes allow the delay between condition test and the branch to be scheduled; however, pipeline control must still track the last instruction that sets the condition code to know when the branch condition is decided. In effect, the condition code must be treated as an operand that requires hazard detection for RAW hazards with branches, just as MIPS must do on the registers.

A final thorny area in pipelining is multicycle operations. Imagine trying to pipeline a sequence of VAX instructions such as this:

MOVL	R1,R2	;moves between registers
ADDL3	42(R1),56(R1)+,@(R1)	;adds memory locations
SUBL2	R2,R3	;subtracts registers
MOVC3	@(R1)[R2],74(R2),R3	;moves a character string

These instructions differ radically in the number of clock cycles they will require, from as low as one up to hundreds of clock cycles. They also require different numbers of data memory accesses, from zero to possibly hundreds. The data hazards are very complex and occur both between and within instructions. The simple solution of making all instructions execute for the same number of clock cycles is unacceptable because it introduces an enormous number of hazards and bypass conditions and makes an immensely long pipeline. Pipelining the VAX at the instruction level is difficult, but a clever solution was found by the VAX 8800 designers. They pipeline the *microinstruction* execution: a microinstruction is a simple instruction used in sequences to implement a more complex instruction set. Because the microinstructions are simple (they look a lot like MIPS), the pipeline control is much easier. Since 1995, all Intel IA-32 microprocessors have

Conditional Branch Options

Since most changes in control flow are branches, deciding how to specify the branch condition is important. Figure B.16 shows the three primary techniques in use today and their advantages and disadvantages.

One of the most noticeable properties of branches is that a large number of the comparisons are simple tests, and a large number are comparisons with zero. Thus, some architectures choose to treat these comparisons as special cases, especially if a *compare and branch* instruction is being used. Figure B.17 shows the frequency of different comparisons used for conditional branching.

Procedure Invocation Options

Procedure calls and returns include control transfer and possibly some state saving; at a minimum the return address must be saved somewhere, sometimes in a special link register or just a GPR. Some older architectures provide a mechanism to save many registers, while newer architectures require the compiler to generate stores and loads for each register saved and restored.

There are two basic conventions in use to save registers: either at the call site or inside the procedure being called. *Caller saving* means that the calling procedure must save the registers that it wants preserved for access after the call, and thus the called procedure need not worry about registers. *Callee saving* is the opposite: the called procedure must save the registers it wants to use, leaving the caller unrestrained. There are times when caller save must be used because of

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Figure B.16 The major methods for evaluating branch conditions, their advantages, and their disadvantages.

Although condition codes can be set by ALU operations that are needed for other purposes, measurements on programs show that this rarely happens. The major implementation problems with condition codes arise when the condition code is set by a large or haphazardly chosen subset of the instructions, rather than being controlled by a bit in the instruction. Computers with compare and branch often limit the set of compares and use a condition register for more complex compares. Often, different techniques are used for branches based on floating-point comparison versus those based on integer comparison. This dichotomy is reasonable since the number of branches that depend on floating-point comparisons is much smaller than the number depending on integer comparisons.

COMPUTER ARCHITECTURE

A Quantitative Approach, Fourth Edition

John L. Hennessy and David A. Patterson

The multiprocessor is here and it can no longer be avoided. As we bid farewell to single-core processors and move into the chip multiprocessing age, it is great timing for a new edition of Hennessy and Patterson's classic. Few books have had as significant an impact on the way their discipline is taught, and the current edition will ensure its place at the top for some time to come.

—Luiz André Barroso, Google Inc.

What do the following have in common: Beatles' tunes, HP calculators, chocolate chip cookies, and *Computer Architecture*? They are all classics that have stood the test of time.

—Robert P. Colwell, Intel lead architect

The best-selling computer architecture book redefines the field with a new edition updated to address the historic shift from single-core to multicore processors. Using their acclaimed quantitative approach to computing design, Hennessy and Patterson explore a variety of techniques for achieving parallelism—the key to unlocking the power of multiple-processor architectures. While refocusing on multiprocessor issues, the authors also cover design factors beyond processor performance, including power, reliability, availability, and dependability.

Sustaining the last century's improvements in cost and performance will require continuing innovations—innovations founded in large part on the fundamental techniques covered in this classic text.

Fourth Edition Features

- **Trademark Putting It All Together** sections highlight the latest technology from industry, including the Sun Niagara, AMD Opteron, and Pentium 4.
- **Review appendices** cover the basic and intermediate principles that the main text relies upon.
- **Reference appendices** on the companion CD—some guest-authored by subject experts—cover a range of topics including embedded systems, vector processors, interconnection networks, and large-scale multiprocessors.
- **Case Studies** contributed by experts in industry and academia are accompanied by increasingly complex exercises to explore the key concepts covered in each chapter.

Online support materials for this book are available at textbooks.elsevier.com/0123704901.



MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER
www.mkp.com

Computer Systems and Design
Computer Engineering

