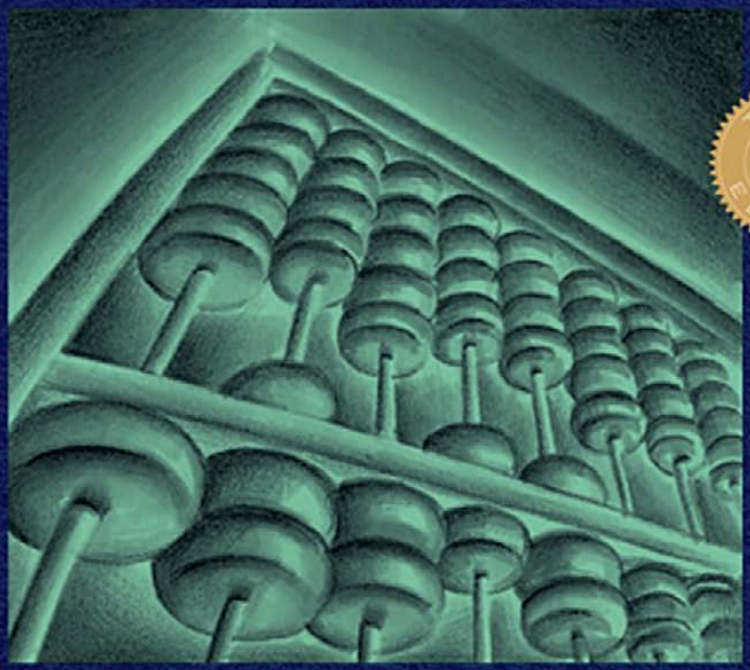
ELSEVIER

COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE



THIRD 3 EDITION

DAVID A. PATTERSON
JOHN L. HENNESSY

MORGAN KAUFMANN

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000001
IPR2024-00627 (Amazon.com, Inc., Amazon.com Services LLC v. Nokia Technologies Oy)

THIRD EDITION

Computer Organization Design

THE HARDWARE/SOFTWARE INTERFACE

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000002

ACKNOWLEDGEMENTS

Figures 1.9, 1.15 Courtesy of Intel.

Figure 1.11 Courtesy of Storage Technology Corp.

Figures 1.7.1, 1.7.2, 6.13.2 Courtesy of the Charles Babbage Institute, University of Minnesota Libraries, Minneapolis.

Figures 1.7.3, 6.13.1, 6.13.3, 7.9.3, 8.11.2 Courtesy of IBM.

Figure 1.7.4 Courtesy of Cray Inc.

Figure 1.7.5 Courtesy of Apple Computer, Inc.

Figure 1.7.6 Courtesy of the Computer History Museum.

Figure 7.33 Courtesy of AMD.

Figures 7.9.1, 7.9.2 Courtesy of Museum of Science, Boston.

Figure 7.9.4 Courtesy of MIPS Technologies, Inc.

Figure 8.3 ©Peg Skorpinski.

Figure 8.11.1 Courtesy of the Computer Museum of America.

Figure 8.11.3 Courtesy of the Commercial Computing Museum.

Figures 9.11.2, 9.11.3 Courtesy of NASA Ames Research Center.

Figure 9.11.4 Courtesy of Lawrence Livermore National Laboratory.

Computers in the Real World:

Photo of "A Laotian villager," courtesy of David Sanger.

Photo of an "Indian villager," property of Encore Software, Ltd., India.

Photos of "Block and students" and "a pop-up archival satellite tag," courtesy of Professor Barbara Block. Photos by Scott Taylor.

Photos of "Professor Dawson and student" and "the Mica micromote," courtesy of AP/World Wide Photos.

Photos of "images of pottery fragments" and "a computer reconstruction," courtesy of Andrew Willis and David B. Cooper, Brown University, Division of Engineering.

Photo of "the Eurostar TGV train," by Jos van der Kolk.

Photo of "the interior of a Eurostar TGV cab," by Andy Veitch.

Photo of "firefighter Ken Whitten," courtesy of World Economic Forum.

Graphic of an "artificial retina," © The San Francisco Chronicle. Reprinted by permission.

Image of "A laser scan of Michelangelo's statue of David," courtesy of Marc Levoy and Dr. Franca Falletti, director of the Galleria dell'Accademia, Italy.

"An image from the Sistine Chapel," courtesy of Luca Pezzati. IR image recorded using the scanner for IR reflectography of the INOA (National Institute for Applied Optics, http://arte.ino.it) at the Opificio delle Pietre Dure in Florence.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000003

THIRD EDITION

Computer Organization and Design

THE HARDWARE/SOFTWARE INTERFACE

David A. Patterson
University of California, Berkeley

John L. Hennessy
Stanford University

With a contribution by
Peter J. Ashenden
Ashenden Designs Pty Ltd

James R. Larus
Microsoft Research

Daniel J. Sorin
Duke University

ELSEVIER

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

Morgan Kaufmann is an imprint of Elsevier

MORGAN KAUFMANN PUBLISHERS

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000004

Senior Editor	Denise E. M. Penrose
Publishing Services Manager	Simon Crump
Editorial Assistant	Summer Block
Cover Design	Ross Caron Design
Cover and Chapter Illustration	Chris Asimoudis
Text Design	GGS Book Services
Composition	Nancy Logan and Dartmouth Publishing, Inc.
Technical Illustration	Dartmouth Publishing, Inc.
Copyeditor	Ken DellaPenta
Proofreader	Jacqui Brownstein
Indexer	Linda Buskus
Interior printer	Courier
Cover printer	Courier

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.

© 2005 by Elsevier Inc. All rights reserved.

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.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

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.uk. You may also complete your request on-line via the Elsevier homepage (http://elsevier.com) by selecting "Customer Support" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data
Application submitted

ISBN: 1-55860-604-1

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com.

Printed in the United States of America
04 05 06 07 08 5 4 3 2 1

Contents

Preface ix

CHAPTERS

1 Computer Abstractions and Technology 2

1.1 Introduction 3
1.2 Below Your Program 11
1.3 Under the Covers 15
1.4 Real Stuff: Manufacturing Pentium 4 Chips 28
1.5 Fallacies and Pitfalls 33
1.6 Concluding Remarks 35
1.7 Historical Perspective and Further Reading 36
1.8 Exercises 36

COMPUTERS IN THE REAL WORLD
Information Technology for the 4 Billion without IT 44

2 Instructions: Language of the Computer 46

2.1 Introduction 48
2.2 Operations of the Computer Hardware 49
2.3 Operands of the Computer Hardware 52
2.4 Representing Instructions in the Computer 60
2.5 Logical Operations 68
2.6 Instructions for Making Decisions 72
2.7 Supporting Procedures in Computer Hardware 79
2.8 Communicating with People 90
2.9 MIPS Addressing for 32-Bit Immediates and Addresses 95
2.10 Translating and Starting a Program 106
2.11 How Compilers Optimize 116
2.12 How Compilers Work: An Introduction 121

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000006

2.13 A C Sort Example to Put It All Together 121
2.14 Implementing an Object-Oriented Language 130
2.15 Arrays versus Pointers 130
2.16 Real Stuff: IA-32 Instructions 134
2.17 Fallacies and Pitfalls 143
2.18 Concluding Remarks 145
2.19 Historical Perspective and Further Reading 147
2.20 Exercises 147

COMPUTERS IN THE REAL WORLD
Helping Save Our Environment with Data 156

3 Arithmetic for Computers 158

3.1 Introduction 160
3.2 Signed and Unsigned Numbers 160
3.3 Addition and Subtraction 170
3.4 Multiplication 176
3.5 Division 183
3.6 Floating Point 189
3.7 Real Stuff: Floating Point in the IA-32 217
3.8 Fallacies and Pitfalls 220
3.9 Concluding Remarks 225
3.10 Historical Perspective and Further Reading 229
3.11 Exercises 229

COMPUTERS IN THE REAL WORLD
Reconstructing the Ancient World 236

4 Assessing and Understanding Performance 238

4.1 Introduction 240
4.2 CPU Performance and Its Factors 246
4.3 Evaluating Performance 254
4.4 Real Stuff: Two SPEC Benchmarks and the Performance of Recent Intel Processors 259
4.5 Fallacies and Pitfalls 266
4.6 Concluding Remarks 270
4.7 Historical Perspective and Further Reading 272
4.8 Exercises 272

COMPUTERS IN THE REAL WORLD
Moving People Faster and More Safely 280

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000007

5 The Processor: Datapath and Control 282

5.1 Introduction 284
5.2 Logic Design Conventions 289
5.3 Building a Datapath 292
5.4 A Simple Implementation Scheme 300
5.5 A Multicycle Implementation 318
5.6 Exceptions 340
5.7 Microprogramming: Simplifying Control Design 346
5.8 An Introduction to Digital Design Using a Hardware Design Language 346
5.9 Real Stuff: The Organization of Recent Pentium Implementations 347
5.10 Fallacies and Pitfalls 350
5.11 Concluding Remarks 352
5.12 Historical Perspective and Further Reading 353
5.13 Exercises 354

COMPUTERS IN THE REAL WORLD
Empowering the Disabled 366

6 Enhancing Performance with Pipelining 368

6.1 An Overview of Pipelining 370
6.2 A Pipelined Datapath 384
6.3 Pipelined Control 399
6.4 Data Hazards and Forwarding 402
6.5 Data Hazards and Stalls 413
6.6 Branch Hazards 416
6.7 Using a Hardware Description Language to Describe and Model a Pipeline 426
6.8 Exceptions 427
6.9 Advanced Pipelining: Extracting More Performance 432
6.10 Real Stuff: The Pentium 4 Pipeline 448
6.11 Fallacies and Pitfalls 451
6.12 Concluding Remarks 452
6.13 Historical Perspective and Further Reading 454
6.14 Exercises 454

COMPUTERS IN THE REAL WORLD
Mass Communication without Gatekeepers 464

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000008

7 Large and Fast: Exploiting Memory Hierarchy 466

7.1 Introduction 468
7.2 The Basics of Caches 473
7.3 Measuring and Improving Cache Performance 492
7.4 Virtual Memory 511
7.5 A Common Framework for Memory Hierarchies 538
7.6 Real Stuff: The Pentium P4 and the AMD Opteron Memory Hierarchies 546
7.7 Fallacies and Pitfalls 550
7.8 Concluding Remarks 552
7.9 Historical Perspective and Further Reading 555
7.10 Exercises 555

COMPUTERS IN THE REAL WORLD
Saving the World's Art Treasures 562

8 Storage, Networks, and Other Peripherals 564

8.1 Introduction 566
8.2 Disk Storage and Dependability 569
8.3 Networks 580
8.4 Buses and Other Connections between Processors, Memory, and I/O Devices 581
8.5 Interfacing I/O Devices to the Processor, Memory, and Operating System 588
8.6 I/O Performance Measures: Examples from Disk and File Systems 597
8.7 Designing an I/O System 600
8.8 Real Stuff: A Digital Camera 603
8.9 Fallacies and Pitfalls 606
8.10 Concluding Remarks 609
8.11 Historical Perspective and Further Reading 611
8.12 Exercises 611

COMPUTERS IN THE REAL WORLD
Saving Lives through Better Diagnosis 622

9 Multiprocessors and Clusters 9-2

9.1 Introduction 9-4
9.2 Programming Multiprocessors 9-8
9.3 Multiprocessors Connected by a Single Bus 9-11

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000009

9.4 Multiprocessors Connected by a Network 9-20
9.5 Clusters 9-25
9.6 Network Topologies 9-27
9.7 Multiprocessors Inside a Chip and Multithreading 9-30
9.8 Real Stuff: The Google Cluster of PCs 9-34
9.9 Fallacies and Pitfalls 9-39
9.10 Concluding Remarks 9-42
9.11 Historical Perspective and Further Reading 9-47
9.12 Exercises 9-55

APPENDICES

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-8
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-47
B.8 Memory Elements: Flip-flops, Latches, and Registers B-49
B.9 Memory Elements: SRAMs and DRAMs B-57
B.10 Finite State Machines B-67
B.11 Timing Methodologies B-72

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000010

B.12 Field Programmable Devices B-77
B.13 Concluding Remarks B-78
B.14 Exercises B-79

C Mapping Control to Hardware C-2

C.1 Introduction C-3
C.2 Implementing Combinational Control Units C-4
C.3 Implementing Finite State Machine Control C-8
C.4 Implementing the Next-State Function with a Sequencer C-21
C.5 Translating a Microprogram to Hardware C-27
C.6 Concluding Remarks C-31
C.7 Exercises C-32

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

D.1 Introduction D-3
D.2 Addressing Modes and Instruction Formats D-5
D.3 Instructions: The MIPS Core Subset D-9
D.4 Instructions: Multimedia Extensions of the Desktop/Server RISCs D-16
D.5 Instructions: Digital Signal-Processing Extensions of the Embedded RISCs D-19
D.6 Instructions: Common Extensions to MIPS Core D-20
D.7 Instructions Unique to MIPS64 D-25
D.8 Instructions Unique to Alpha D-27
D.9 Instructions Unique to SPARC v.9 D-29
D.10 Instructions Unique to PowerPC D-32
D.11 Instructions Unique to PA-RISC 2.0 D-34
D.12 Instructions Unique to ARM D-36
D.13 Instructions Unique to Thumb D-38
D.14 Instructions Unique to SuperH D-39
D.15 Instructions Unique to M32R D-40
D.16 Instructions Unique to MIPS16 D-41
D.17 Concluding Remarks D-43
D.18 Acknowledgments D-46
D.19 References D-47

Index I-1
Glossary G-1
Further Reading FR-1

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000011

can use it as a temporary register (see the Hardware Software Interface section on page 96 in Chapter 2), compilers also abstain from using registers `$k0` and `$k1` to make them available for the operating system. Exception routines place the return address in one of these registers and then use jump register to restore the instruction address.

Multiplication is vexation,
Division is as bad;
The rule of three doth puzzle me,
And practice drives me mad.

Anonymous, Elizabethan manuscript, 1570

3.4 Multiplication

Now that we have completed the explanation of addition and subtraction, we are ready to build the more vexing operation of multiply.

But first let's review the multiplication of decimal numbers in longhand to remind ourselves of the steps and the names of the operands. For reasons that will become clear shortly, we limit this decimal example to using only the digits 0 and 1. Multiplying 1000_{ten} by 1001_{ten}:

```
Multiplicand                          1000ten
Multiplier                  x         1001ten
                                     -------
                                      1000
                                     0000
                                    0000
                                   1000
                                  ---------
Product                           1001000ten
```

The first operand is called the *multiplicand* and the second the *multiplier*. The final result is called the *product*. As you may recall, the algorithm learned in grammar school is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by the single digit of the multiplier and shifting the intermediate product one digit to the left of the earlier intermediate products.

The first observation is that the number of digits in the product is considerably larger than the number in either the multiplicand or the multiplier. In fact, if we ignore the sign bits, the length of the multiplication of an n-bit multiplicand and an m-bit multiplier is a product that is $n + m$ bits long. That is, $n + m$ bits are required to represent all possible products. Hence, like add, multiply must cope with overflow because we frequently want a 32-bit product as the result of multiplying two 32-bit numbers.

In this example we restricted the decimal digits to 0 and 1. With only two choices, each step of the multiplication is simple:

1. Just place a copy of the multiplicand (1 ¥ multiplicand) in the proper place if the multiplier digit is a 1, or
2. Place 0 (0 ¥ multiplicand) in the proper place if the digit is 0.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000012

Although the decimal example above happened to use only 0 and 1, multiplication of binary numbers must always use 0 and 1, and thus always offers only these two choices.

Now that we have reviewed the basics of multiplication, the traditional next step is to provide the highly optimized multiply hardware. We break with tradition in the belief that you will gain a better understanding by seeing the evolution of the multiply hardware and algorithm through multiple generations. For now, let's assume that we are multiplying only positive numbers.

Sequential Version of the Multiplication Algorithm and Hardware

This design mimics the algorithm we learned in grammar school; the hardware is shown in Figure 3.5. We have drawn the hardware so that data flows from top to bottom to more closely resemble the paper-and-pencil method.

Let's assume that the multiplier is in the 32-bit Multiplier register and that the 64-bit Product register is initialized to 0. From the paper-and-pencil example above, it's clear that we will need to move the multiplicand left one digit each step as it may be added to the intermediate products. Over 32 steps a 32-bit multiplicand would move 32 bits to the left. Hence we need a 64-bit Multiplicand register, initialized with the 32-bit multiplicand in the right half and 0 in the left half. This register is then shifted left 1 bit each step to align the multiplicand with the sum being accumulated in the 64-bit Product register.

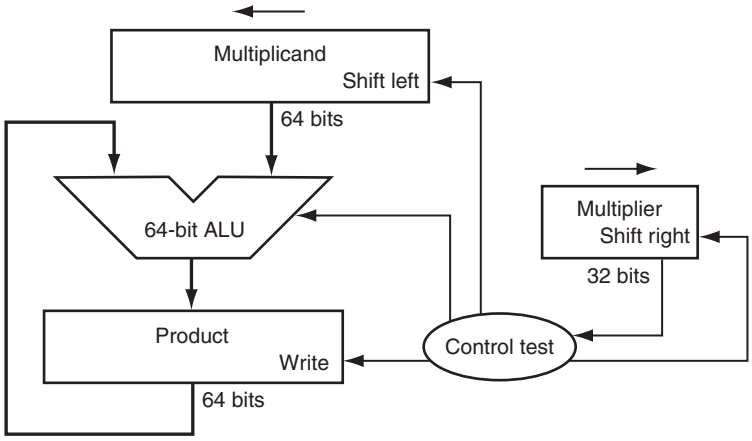


FIGURE 3.5 First version of the multiplication hardware. The Multiplicand register, ALU, and Product register are all 64 bits wide, with only the Multiplier register containing 32 bits. The 32-bit multiplicand starts in the right half of the Multiplicand register and is shifted left 1 bit on each step. The multiplier is shifted in the opposite direction at each step. The algorithm starts with the product initialized to 0. Control decides when to shift the Multiplicand and Multiplier registers and when to write new values into the Product register.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000013

Figure 3.6 shows the three basic steps needed for each bit. The least significant bit of the multiplier (Multiplier0) determines whether the multiplicand is added to the Product register. The left shift in step 2 has the effect of moving the intermediate operands to the left, just as when multiplying by hand. The shift right in step 3 gives us the next bit of the multiplier to examine in the following iteration. These three steps are repeated 32 times to obtain the product. If each step took a

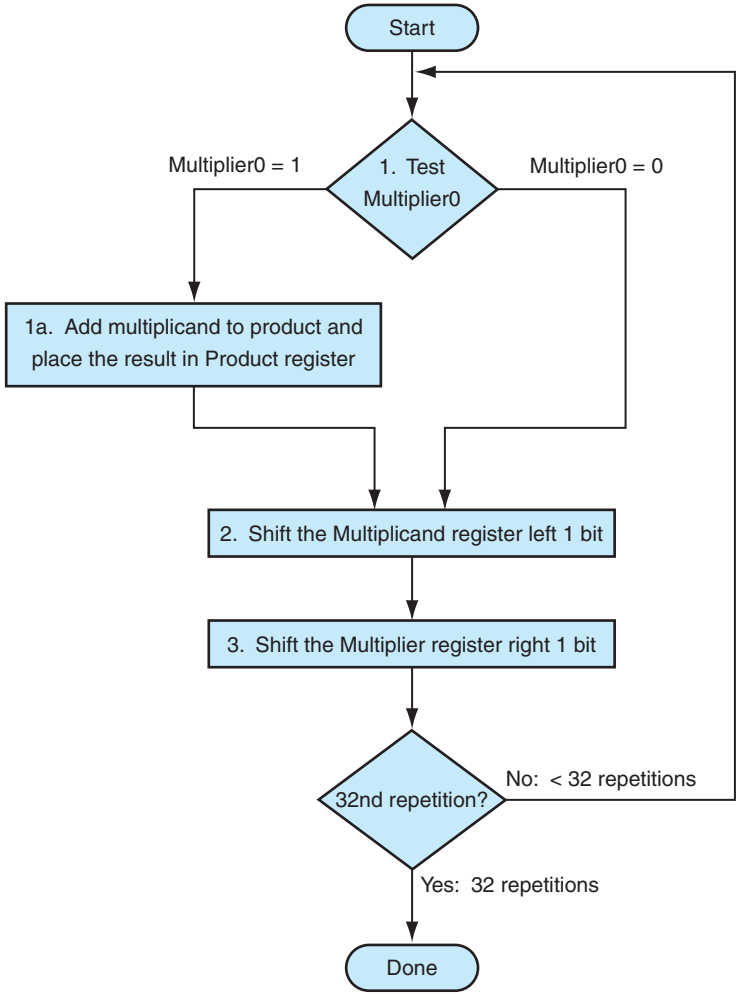


FIGURE 3.6 The first multiplication algorithm, using the hardware shown in Figure 3.5. If the least significant bit of the multiplier is 1, add the multiplicand to the product. If not, go to the next step. Shift the multiplicand left and the multiplier right in the next two steps. These three steps are repeated 32 times.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000014

clock cycle, this algorithm would require almost 100 clock cycles to multiply two 32-bit numbers. The relative importance of arithmetic operations like multiply varies with the program, but addition and subtraction may be anywhere from 5 to 100 times more popular than multiply. Accordingly, in many applications, multiply can take multiple clock cycles without significantly affecting performance. Yet Amdahl's law (see Chapter 4, page 267) reminds us that even a moderate frequency for a slow operation can limit performance.

This algorithm and hardware are easily refined to take 1 clock cycle per step. The speedup comes from performing the operations in parallel: the multiplier and multiplicand are shifted while the multiplicand is added to the product if the multiplier bit is a one. The hardware just has to ensure that it tests the right bit of the multiplier and gets the preshifted version of the multiplicand. The hardware is usually further optimized to halve the width of the adder and registers by noticing where there are unused portions of registers and adders. Figure 3.7 shows the revised hardware.

Hardware Software Interface

Replacing arithmetic by shifts can also occur when multiplying by constants. Some compilers replace multiplies by short constants with a series of shifts and adds. Because one bit to the left represents a number twice as large in base 2, shifting the bits left has the same effect as multiplying by a power of 2. As mentioned in Chapter 2, almost every compiler will perform the strength reduction optimization of substituting a left shift for a multiply by a power of 2.

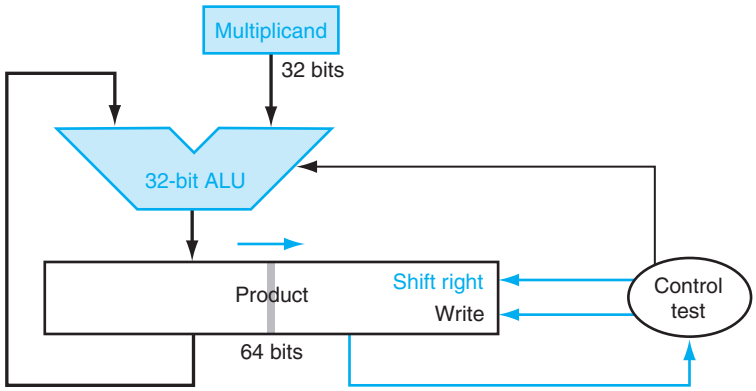


FIGURE 3.7 Refined version of the multiplication hardware. Compare with the first version in Figure 3.5. The Multiplicand register, ALU, and Multiplier register are all 32 bits wide, with only the Product register left at 64 bits. Now the product is shifted right. The separate Multiplier register also disappeared. The multiplier is placed instead in the right half of the Product register. These changes are highlighted in color.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000015

EXAMPLE

A Multiply Algorithm

Using 4-bit numbers to save space, multiply 2_{ten} ¥ 3_{ten}, or 0010_{two} ¥ 0011_{two}.

ANSWER

Figure 3.8 shows the value of each register for each of the steps labeled according to Figure 3.6, with the final value of $0000\ 0110_{two}$ or 6_{ten}. Color is used to indicate the register values that change on that step, and the bit circled is the one examined to determine the operation of the next step.

Signed Multiplication

So far we have dealt with positive numbers. The easiest way to understand how to deal with signed numbers is to first convert the multiplier and multiplicand to positive numbers and then remember the original signs. The algorithms should then be run for 31 iterations, leaving the signs out of the calculation. As we learned in grammar school, we need negate the product only if the original signs disagree.

It turns out that the last algorithm will work for signed numbers provided that we remember that the numbers we are dealing with have infinite digits, and that we are only representing them with 32 bits. Hence, the shifting steps would need to extend the sign of the product for signed numbers. When the algorithm completes, the lower word would have the 32-bit product.

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 fi Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 fi Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 fi no operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 fi no operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

FIGURE 3.8 Multiply example using algorithm in Figure 3.6. The bit examined to determine the next step is circled in color.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000016

Faster Multiplication

Moore's Law has provided so much more in resources that hardware designers can now build a much faster multiplication hardware. Whether the multiplicand is to be added or not is known at the beginning of the multiplication by looking at each of the 32 multiplier bits. Faster multiplications are possible by essentially providing one 32-bit adder for each bit of the multiplier: one input is the multiplicand ANDed with a multiplier bit and the other is the output of a prior adder. Figure 3.9 shows how they would be connected.

Why is this much hardware faster? The sequential multiplier pays the overhead of a clock for each bit of the product. This multiplier array of adders does not. A second reason is this large collection of adders lends itself to many optimizations to gain further improvements. One example is using *carry save adders* to add such a large column of numbers; see Exercises 3.24 and 3.49. A third reason is that it is easy to pipeline such a design to be able to support many multiplies simultaneously (see Chapter 6).

Multiply in MIPS

MIPS provides a separate pair of 32-bit registers to contain the 64-bit product, called *Hi* and *Lo*. To produce a properly signed or unsigned product, MIPS has two instructions: multiply (`mult`) and multiply unsigned (`multu`). To fetch the integer 32-bit product, the programmer uses *move from lo* (`mflo`). The MIPS assembler generates a pseudoinstruction for multiply that specifies three general-purpose registers, generating `mflo` and `mfhi` instructions to place the product into registers.

Summary

Multiplication is accomplished by simple shift and add hardware, derived from the paper-and-pencil method learned in grammar school. Compilers even use shift instructions for multiplications by powers of two.

Hardware Software Interface

Both MIPS multiply instructions ignore overflow, so it is up to the software to check to see if the product is too big to fit in 32 bits. There is no overflow if Hi is 0 for `multu` or the replicated sign of Lo for `mult`. The instruction *move from hi* (`mfhi`) can be used to transfer Hi to a general-purpose register to test for overflow.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000017

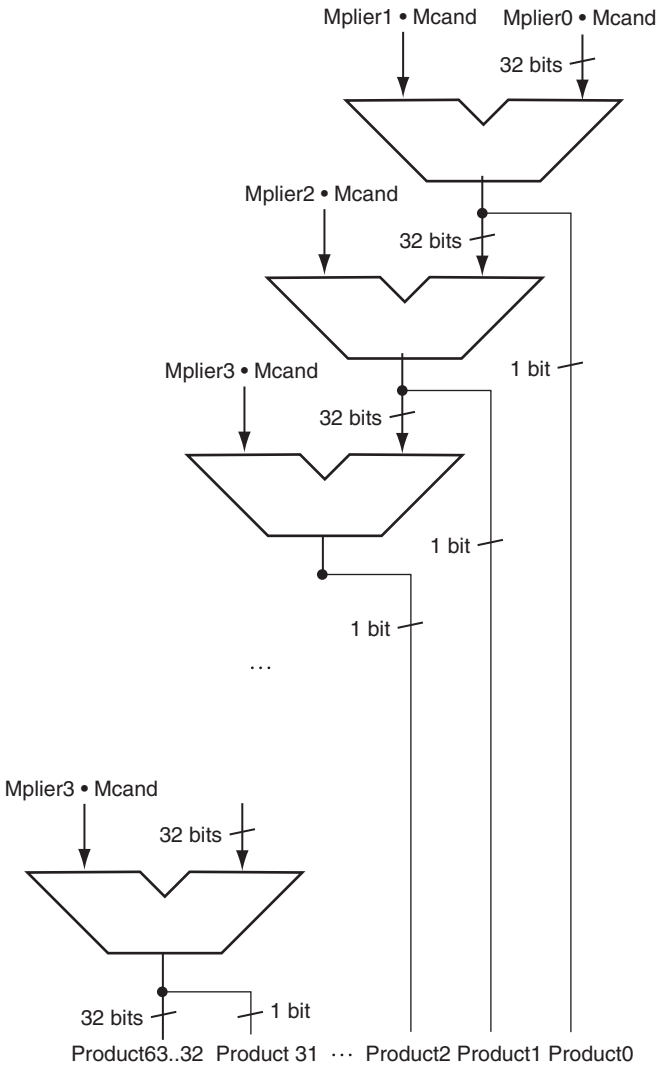


FIGURE 3.9 Fast multiplication hardware. Rather than use a single 32-bit adder 32 times, this hardware "unrolls the loop" to use 32 adders. Each adder produces a 32-bit sum and a carry out. The least significant bit is a bit of the product, and the carry out and the upper 31 bits of the sum are passed along to the next adder.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1026, Page 000018