See MIPS® Run

Second Edition

Dominic Sweetman
The MIPS architecture was born in the early 1980s from the work done by John Hennessy and his students at Stanford University. They were exploring the architectural concept of RISC (Reduced Instruction Set Computing), which theorized that relatively simple instructions, combined with excellent compilers and hardware that used pipelining to execute the instructions, could produce a faster processor with less die area. The concept was so successful that MIPS Computer Systems was formed in 1984 to commercialize the MIPS architecture.

Over the course of the next 14 years, the MIPS architecture evolved in a number of ways and its implementations were used very successfully in workstation and server systems. Over that time, the architecture and its implementations were enhanced to support 64-bit addressing and operations, support for complex memory-protected operating systems such as UNIX, and very high performance floating point. Also in that period, MIPS Computer Systems was acquired by Silicon Graphics and MIPS processors became the standard for Silicon Graphics computer systems. With 64-bit processors, high-performance floating point, and the Silicon Graphics heritage, MIPS processors became the solution of choice in high-volume gaming consoles.

In 1998, MIPS Technologies emerged from Silicon Graphics as a stand-alone company focused entirely on intellectual property for embedded markets. As a result, the pace of architecture development has increased to address the unique needs of these markets: high-performance computation, code compression, geometry processing for graphics, security, signal processing, and multithreading. Each architecture development has been matched by processor core implementations of the architecture, making MIPS-based processors the standard for high-performance, low-power applications.

The MIPS legacy in complex systems such as workstations and servers directly benefits today’s embedded systems, which have, themselves, become very complex. A typical embedded system is composed of multiple processing elements, high-performance memory, and one or more operating systems.
When compared with other embedded architectures, which are just now learning what is required to build a complex system, the MIPS architecture provides a proven base on which to implement such systems.

In many ways, the first edition of *See MIPS Run* was a ground-breaking book on the MIPS architecture and its implementations. While other books covered similar material, *See MIPS Run* focused on what the programmer needed to understand of the architecture and the software environment in order to effectively program a MIPS chip.

Increasing complexity of embedded systems has been matched by enhancements to the MIPS architecture to address the needs of such systems. The second edition of this book is required reading for any current developer of MIPS-based embedded systems. It adds significant new material, including the architectural standardization of the MIPS32 and MIPS64 architectures, brand new application-specific extensions such as multithreading, and a very nice treatment of the implementation of the popular Linux operating system on the MIPS architecture. Short of the MIPS architecture specifications, the second edition of *See MIPS Run* is the most current description of the state of the art of the architecture and is, bar none, the most readable.

I hope that you will find this as worthwhile and as entertaining to read as I did.

Michael Uhler,
Chief Technology Officer, MIPS Technologies, Inc.
Mountain View, CA
May 2006
Contents

Foreword v
Preface xv
  Style and Limits xviii
  Conventions xviii
  Acknowledgments xix

Chapter 1 RISCs and MIPS Architectures 1
  1.1 Pipelines 2
    1.1.1 What Makes a Pipeline Inefficient? 3
    1.1.2 The Pipeline and Caching 4
  1.2 The MIPS Five-Stage Pipeline 5
  1.3 RISC and CISC 7
  1.4 Great MIPS Chips of the Past and Present 8
    1.4.1 R2000 to R3000 Processors 8
    1.4.2 The R6000 Processor: A Diversion 9
    1.4.3 The First CPU Cores 11
    1.4.4 The R4000 Processor: A Revolution 12
    1.4.5 The Rise and Fall of the ACE Consortium 12
    1.4.6 SGI Acquires MIPS 13
    1.4.7 QED: Fast MIPS Processors for Embedded Systems 13
    1.4.8 The R10000 Processor and its Successors 14
    1.4.9 MIPS Processors in Consumer Electronics 15
    1.4.10 MIPS in Network Routers and Laser Printers 15
    1.4.11 MIPS Processors in Modern Times 17
    1.4.12 The Rebirth of MIPS Technologies 20
    1.4.13 The Present Day 21
  1.5 MIPS Compared with CISC Architectures 23
    1.5.1 Constraints on MIPS Instructions 23
    1.5.2 Addressing and Memory Accesses 24
    1.5.3 Features You Won’t Find 25
    1.5.4 Programmer-Visible Pipeline Effects 27
Chapter 2  MIPS Architecture

2.1 A Flavor of MIPS Assembly Language 33
2.2 Registers 34
    2.2.1 Conventional Names and Uses of General-Purpose Registers 35
2.3 Integer Multiply Unit and Registers 38
2.4 Loading and Storing: Addressing Modes 39
2.5 Data Types in Memory and Registers 39
    2.5.1 Integer Data Types 39
    2.5.2 Unaligned Loads and Stores 40
    2.5.3 Floating-Point Data in Memory 41
2.6 Synthesized Instructions in Assembly Language 42
2.7 MIPS I to MIPS64 ISAs: 64-Bit (and Other) Extensions 43
    2.7.1 To 64 Bits 45
    2.7.2 Who Needs 64 Bits? 45
    2.7.3 Regarding 64 Bits and No Mode Switch: Data in Registers 46
2.8 Basic Address Space 47
    2.8.1 Addressing in Simple Systems 49
    2.8.2 Kernel versus User Privilege Level 49
    2.8.3 The Full Picture: The 64-Bit View of the Memory Map 50
2.9 Pipeline Visibility 50

Chapter 3  Coprocessor 0: MIPS Processor Control

3.1 CPU Control Instructions 55
3.2 Which Registers Are Relevant When? 58
3.3 CPU Control Registers and Their Encoding 59
    3.3.1 Status Register (SR) 60
    3.3.2 Cause Register 64
    3.3.3 Exception Restart Address (EPC) Register 65
    3.3.4 Bad Virtual Address (BadVAddr) Register 67
    3.3.5 Count/Compare Registers: The On-CPU Timer 68
    3.3.6 Processor ID (PRId) Register 68
    3.3.7 Config Registers: CPU Resource Information and Configuration 69
    3.3.8 EBase and IntCtl: Interrupt and Exception Setup 73
    3.3.9 SRSCil and SRSMap: Shadow Register Setup 74
    3.3.10 Load-Linked Address (LLAddr) Register 75
3.4 CP0 Hazards—A Trap for the Unwary 75
    3.4.1 Hazard Barrier Instructions 76
    3.4.2 Instruction Hazards and User Hazards 77
    3.4.3 Hazards between CP0 Instructions 77

Chapter 4  How Caches Work on MIPS Processors

4.1 Caches and Cache Management 79
4.2 How Caches Work 80
4.3 Write-Through Caches in Early MIPS CPUs 83
4.4 Write-Back Caches in MIPS CPUs 84
4.5 Other Choices in Cache Design 84
4.6 Managing Caches 86
4.7 L2 and L3 Caches 88
4.8 Cache Configurations for MIPS CPUs 88
4.9 Programming MIPS32/64 Caches 90
  4.9.1 The Cache Instruction 91
  4.9.2 Cache Initialization and Tag/Data Registers 92
  4.9.3 CacheErr, ERR, and ErrorEPC Registers: Memory/Cache Error Handling 94
  4.9.4 Cache Sizing and Figuring Out Configuration 95
  4.9.5 Initialization Routines 96
  4.9.6 Invalidating or Writing Back a Region of Memory in the Cache 97
4.10 Cache Efficiency 98
4.11 Reorganizing Software to Influence Cache Efficiency 100
4.12 Cache Aliases 102

Chapter 5
Exceptions, Interrupts, and Initialization 105
  5.1 Precise Exceptions 107
    5.1.1 Nonprecise Exceptions—The Multiplier in Historic MIPS CPUs 108
  5.2 When Exceptions Happen 109
  5.3 Exception Vectors: Where Exception Handling Starts 109
  5.4 Exception Handling: Basics 113
  5.5 Returning from an Exception 114
  5.6 Nesting Exceptions 114
  5.7 An Exception Routine 115
  5.8 Interrupts 115
    5.8.1 Interrupt Resources in MIPS CPUs 116
    5.8.2 Implementing Interrupt Priority in Software 118
    5.8.3 Atomicity and Atomic Changes to SR 120
    5.8.4 Critical Regions with Interrupts Enabled: Semaphores the MIPS Way 121
    5.8.5 Vectored and EIC Interrupts in MIPS32/64 CPUs 123
    5.8.6 Shadow Registers 124
  5.9 Starting Up 124
    5.9.1 Probing and Recognizing Your CPU 126
    5.9.2 Bootstrap Sequences 127
    5.9.3 Starting Up an Application 128
  5.10 Emulating Instructions 128

Chapter 6
Low-level Memory Management and the TLB 131
  6.1 The TLB/MMU Hardware and What It Does 131
  6.2 TLB/MMU Registers Described 132
    6.2.1 TLB Key Fields—EntryHi and PageMask 134
    6.2.2 TLB Output Fields—EntryLo0-1 136
Contents

Chapter 8  Complete Guide to the MIPS Instruction Set  183

8.1 A Simple Example  183
8.2 Assembly Instructions and What They Mean  185
  8.2.1 U and Non-U Mnemonics  186
  8.2.2 Divide Mnemonics  187
  8.2.3 Inventory of Instructions  188
8.3 Floating-Point Instructions  210
8.4 Differences in MIPS32/64 Release 1  216
  8.4.1 Regular Instructions Added in Release 2  216
  8.4.2 Privileged Instructions Added in Release 2  218
8.5 Peculiar Instructions and Their Purposes  218
  8.5.1 Load Left/Load Right: Unaligned Load and Store  218
  8.5.2 Load-Linked/Store-Conditional  223
  8.5.3 Conditional Move Instructions  224
  8.5.4 Branch-Like  225
  8.5.5 Integer Multiply-Accumulate and Multiply-Add Instructions  226
  8.5.6 Floating-Point Multiply-Add Instructions  227
  8.5.7 Multiple FP Condition Bits  228
  8.5.8 Prefetch  228
  8.5.9 Sync: A Memory Barrier for Loads and Stores  229
  8.5.10 Hazard Barrier Instructions  231
  8.5.11 Synci: Cache Management for Instruction Writers  232
  8.5.12 Read Hardware Register  232
8.6 Instruction Encodings  233
  8.6.1 Fields in the Instruction Encoding Table  233
  8.6.2 Notes on the Instruction Encoding Table  251
  8.6.3 Encodings and Simple Implementation  251
8.7 Instructions by Functional Group  252
  8.7.1 No-op  252
  8.7.2 Register/Register Moves  252
  8.7.3 Load Constant  253
  8.7.4 Arithmetical/Logical  253
  8.7.5 Integer Multiply, Divide, and Remainder  255
  8.7.6 Integer Multiply-Accumulate  256
  8.7.7 Loads and Stores  257
  8.7.8 Jumps, Subroutine Calls, and Branches  259
  8.7.9 Breakpoint and Trap  260
  8.7.10 CP0 Functions  260
  8.7.11 Floating Point  261
  8.7.12 Limited User-Mode Access to “Under the Hood” Features  261

Chapter 9  Reading MIPS Assembly Language  263

9.1 A Simple Example  264
9.2 Syntax Overview  268
  9.2.1 Layout, Delimiters, and Identifiers  268
9.3 General Rules for Instructions  269
Chapter 10 Porting Software to the MIPS Architecture

10.1 Low-Level Software for MIPS Applications: A Checklist of Frequently Encountered Problems 280
10.2 Endianness: Words, Bytes, and Bit Order 281
10.2.1 Bits, Bytes, Words, and Integers 281
10.2.2 Software and Endianness 284
10.2.3 Hardware and Endianness 287
10.2.4 Bi-endian Software for a MIPS CPU 293
10.2.5 Portability and Endianness-Independent Code 295
10.2.6 Endianness and Foreign Data 295
10.3 Trouble with Visible Caches 296
10.3.1 Cache Management and DMA Data 298
10.3.2 Cache Management and Writing Instructions: Self-Modifying Code 299
10.3.3 Cache Management and Uncached or Write-Through Data 300
10.3.4 Cache Aliases and Page Coloring 301
10.4 Memory Access Ordering and Reordering 301
10.4.1 Ordering and Write Buffers 304
10.4.2 Implementing wbflush 304
10.5 Writing it in C 305
10.5.1 Wrapping Assembly Code with the GNU C Compiler 305
10.5.2 Memory-Mapped I/O Registers and “Volatile” 307
10.5.3 Miscellaneous Issues When Writing C for MIPS Applications 308

Chapter 11 MIPS Software Standards (ABIs)

11.1 Data Representations and Alignment 312
11.1.1 Sizes of Basic Types 312
11.1.2 Sizes of “long” and Pointer Types 313
11.1.3 Alignment Requirements 313
11.1.4 Memory Layout of Basic Types and How It Changes with Endianness 313
11.1.5 Memory Layout of Structure and Array Types and Alignment 315
11.1.6 Bitfields in Structures 315
11.1.7 Unaligned Data from C 318
11.2 Argument Passing and Stack Conventions for MIPS ABIs 319
11.2.1 The Stack, Subroutine Linkage, and Parameter Passing 320
11.2.2 Stack Argument Structure in o32 320
11.2.3 Using Registers to Pass Arguments 321
### Contents

11.2.4 Examples from the C Library 322  
11.2.5 An Exotic Example: Passing Structures 323  
11.2.6 Passing a Variable Number of Arguments 324  
11.2.7 Returning a Value from a Function 325  
11.2.8 Evolving Register-Use Standards: SGIs n32 and n64 326  
11.2.9 Stack Layouts, Stack Frames, and Helping Debuggers 329  
11.2.10 Variable Number of Arguments and stdargs 337

**Chapter 12** Debugging MIPS Designs—Debug and Profiling Features 339

12.1 The “EJTAG” On-chip Debug Unit 341  
12.1.1 EJTAG History 343  
12.1.2 How the Probe Controls the CPU 343  
12.1.3 Debug Communications through JTAG 344  
12.1.4 Debug Mode 344  
12.1.5 Single-Stepping 346  
12.1.6 The dseg Memory Decode Region 346  
12.1.7 EJTAG CP0 Registers, Particularly Debug 348  
12.1.8 The DCR (Debug Control) Memory-Mapped Register 351  
12.1.9 EJTAG Breakpoint Hardware 352  
12.1.10 Understanding Breakpoint Conditions 355  
12.1.11 Imprecise Debug Breaks 356  
12.1.12 PC Sampling with EJTAG 356  
12.1.13 Using EJTAG without a Probe 356  
12.2 Pre-EJTAG Debug Support—Break Instruction and CP0 Watchpoints 358  
12.3 PDtrace 359  
12.4 Performance Counters 360

**Chapter 13** GNU/Linux from Eight Miles High 363

13.1 Components 364  
13.2 Layering in the Kernel 368  
13.2.1 MIPS CPU in Exception Mode 368  
13.2.2 MIPS CPU with Some or All Interrupts off 369  
13.2.3 Interrupt Context 370  
13.2.4 Executing the Kernel in Thread Context 370

**Chapter 14** How Hardware and Software Work Together 371

14.1 The Life and Times of an Interrupt 371  
14.1.1 High-Performance Interrupt Handling and Linux 374  
14.2 Threads, Critical Regions, and Atomicity 375  
14.2.1 MIPS Architecture and Atomic Operations 376  
14.2.2 Linux Spinlocks 377  
14.3 What Happens on a System Call 378  
14.4 How Addresses Get Translated in Linux/MIPS Systems 380
### Chapter 15  
**MIPS Specific Issues in the Linux Kernel**

15. Explicit Cache Management
   15.1.1 DMA Device Accesses
   15.1.2 Writing Instructions for Later Execution
   15.1.3 Cache/Memory Mapping Problems
   15.1.4 Cache Aliases
15.2 CP0 Pipeline Hazards
15.3 Multiprocessor Systems and Coherent Caches
15.4 Demon Tweaks for a Critical Routine

### Chapter 16  
**Linux Application Code, PIC, and Libraries**

16.1 How Link Units Get into a Program
16.2 Global Offset Table (GOT) Organization

### Appendix A  
**MIPS Multithreading**

A.1 What Is Multithreading?
A.2 Why Is MT Useful?
A.3 How to Do Multithreading for MIPS
A.4 MT in Action

### Appendix B  
**Other Optional Extensions to the MIPS Instruction Set**

B.1 MIPS16 and MIPS16e ASEs
   B.1.1 Special Encodings and Instructions in the MIPS16 ASE
   B.1.2 The MIPS16 ASE Evaluated
B.2 The MIPS DSP ASE
B.3 The MDMX ASE

**MIPS Glossary**

**References**

**Books and Articles**

**Online Resources**

**Index**
Preface

This book is about MIPS, the cult hit from the mid-1980s’ crop of RISC CPU designs. These days MIPS is not the highest-volume 32-bit architecture, but it is in a comfortable second place. Where it wins, hands down, is its range of applications. A piece of equipment built around a MIPS CPU might have cost you $35 for a wireless router or hundreds of thousands of dollars for an SGI supercomputer (though with SGI’s insolvency, those have now reached the end of the line). Between those extremes are Sony and Nintendo games machines, many Cisco routers, TV set-top boxes, laser printers, and so on.

The first edition of this book has sold close to 10,000 English copies over the years and has been translated into Chinese. I’m pleased and surprised; I didn’t know there were so many MIPS programmers out there.

This second edition is *See MIPS Run... Linux*. The first edition struggled to motivate some features of the MIPS architecture, because they don’t make sense unless you can see how they help out inside an OS kernel. But now a lot of you have some sense of how Linux works, and I can quote its source code; more importantly, I can refer to it knowing that those of you who get interested can read the source code and find out how it’s really done.

So this is a book about the MIPS architecture, but the last three chapters stroll through the Linux kernel and application-programming system to cast light on what those weird features do. I hope Linux experts will forgive my relative ignorance of Linux details, but the chance to go for a description of a real OS running on a real architecture was too good to pass up.

MIPS is a RISC: a useful acronym, well applied to the common features of a number of computer architectures invented in the 1980s, to realize efficient pipelined implementation. The acronym CISC is vaguer. I’ll use it in a narrow sense, for the kind of features found in x86 and other pre-1982 architectures, designed with microcoded implementations in mind.

Some of you may be up in arms: He’s confusing implementation with architecture! But while computer architecture is supposed to be a contract with the
programmer about what programs will run correctly, it’s also an engineering
design in its own right. A computer architecture is designed to make for good
CPUs. As chip design becomes more sophisticated, the trade-offs change.

This book is for programmers, and that’s the test we’ve used to decide what
gets included—if a programmer might see it, or is likely to be interested, it’s
here. That means we don’t get to discuss, for example, the strange system inter-
faces with which MIPS has tortured two generations of hardware design engi-
neers. And your operating system may hide many of the details we talk about
here; there is many an excellent programmer who thinks that C is quite low
level enough, portability a blessing, and detailed knowledge of the architecture
irrelevant. But sometimes you do need to get down to the nuts and bolts—and
human beings are born curious as to how bits of the world work.

A result of this orientation is that we’ll tend to be rather informal when
describing things that may not be familiar to a software engineer—particularly
the inner workings of the CPU—but we’ll get much more terse and technical
when we’re dealing with the stuff programmers have met before, such as
registers, instructions, and how data is stored in memory.

We’ll assume some familiarity and comfort with the C language. Much of
the reference material in the book uses C fragments as a way of compress-
ing operation descriptions, particularly in the chapters on the details of the
instruction set and assembly language.

Some parts of the book are targeted at readers who’ve seen some assembly
language: the ingenuity and peculiarity of the MIPS architecture shows up best
from that viewpoint. But if assembly is a closed book to you, that’s probably
not a disaster.

This book aims to tell you everything you need to know about program-
ing generic MIPS CPUs. More precisely, it describes the architecture as it’s
defined by MIPS Technologies’ MIPS32 and MIPS64—specifically, the second
release of those specifications from 2003. We’ll shorten that to “MIPS32/64.”
But this is not just a reference manual: To keep an architecture in your head
means coming to understand it in the round. I also hope the book will interest
students of programming (at college or enrolled in the school of life) who want
to understand a modern CPU architecture all the way through.

If you plan to read this book straight through from front to back, you will
expect to find a progression from overview to detail, and you won’t be disap-
pointed. But you’ll also find some progression through history; the first time
we talk about a concept we’ll usually focus on its first version. Hennessy and
Patterson call this “learning through evolution,” and what’s good enough for
them is certainly good enough for me.

We start in Chapter 1 with some history and background, and set MIPS in
context by discussing the technological concerns and ideas that were uppermost
in the minds of its inventors. Then in Chapter 2 we discuss the characteristics
of the MIPS machine language that follow from their approach.
To help you see the big picture, we leave out the details of processor control until Chapter 3, which introduces the ugly but eminently practical system that allows MIPS CPUs to deal with their caches, exceptions and startup, and memory management. Those last three topics, respectively, become the subjects of Chapters 4 through 6.

The MIPS architecture has been careful to separate out the part of the instruction set that deals with floating-point numbers. That separation allows MIPS CPUs to be built with various levels of floating-point support, from none at all through partial implementations to somewhere near the state of the art. So we have also separated out the floating-point functions, and we keep them back until Chapter 7.

Up to this point, the chapters follow a reasonable sequence for getting to know MIPS. The following chapters change gear and are more like reference manuals or example-based tutorials.

In Chapter 8, we go through the whole machine instruction set; the intention is to be precise but much more terse than the standard MIPS reference works—we cover in 10 pages what takes a hundred in other sources. Chapter 9 is a brief introduction to reading and writing assembly, and falls far short of an assembly programming manual.

Chapter 10 is a checklist with helpful hints for those of you who have to port software between another CPU and a MIPS CPU. The longest section tackles the troublesome problem of endianness in CPUs, software, and systems.

Chapter 11 is a bare-bones summary of the software conventions (register use, argument passing, etc.) necessary to produce interworking software with different toolkits. Chapter 12 introduces the debug and profiling features standardized for MIPS CPUs.

Then we’re on to seeing how MIPS runs GNU/Linux. We describe relationship between the Linux kernel and a computer architecture in Chapter 13; then Chapters 14 and 15 dig down into some of the detail as to how the MIPS architecture does what the Linux kernel needs. Chapter 16 gives you a quick look at the dynamic linking magic that makes GNU/Linux applications work.

Appendix A covers the MIPS MT (multithreading) extension, probably the most important addition to the architecture in many years. And Appendix B describes the more important add-ons: MIPS16, the new MIPS DSP extensions, and MDMX.

You will also find at the end of this book a glossary of terms—a good place to look for specialized or unfamiliar usage and acronyms—and a list of books, papers, and online references for further reading.

1. I have taken considerable care in the generation of these tables, and they are mostly right. But if your system depends on it, be sure to cross-check this information. An excellent source of fairly reliable information can be found in the behavior and source code of the GNU tool collection—but I referred to that too, so it’s not completely independent.
Style and Limits

Every book reflects its author, so we’d better make a virtue of it.

Since some of you will be students, I wondered whether I should distinguish general use from MIPS use. I decided not to; I aim to be specific except where it costs the reader nothing to be general. I also try to be concrete rather than abstract. I don’t worry overmuch about whatever meaning terms like “TLB” have in the wider industry, but I explain them in a MIPS context. Human beings are great generalizers, and this is unlikely to damage your learning much.

It’s 20 years since I started working with MIPS CPUs in the fall of 1986. Some of the material in this book goes back as far as 1988, when I started giving training courses on MIPS architecture. In 1993, I gathered them together to make a software manual focused on IDT’s R3051 family CPUs. It took quite a lot of extra material to create the first edition, published in 1999.

A lot has happened since 1999. MIPS is now at the very end of its life in servers with SGI but has carved out a significant niche in embedded systems. Linux has emerged as the most-used OS for embedded MIPS, but there’s still a lot of diversity in the embedded market. The MIPS specifications have been reorganized around MIPS32 and MIPS64 (which this edition regards as the baseline). This second edition has been in the works for about three years.

The MIPS story continues; if it did not, we’d only be writing this book for historians, and Morgan Kaufmann wouldn’t be very interested in publishing it. MIPS developments that weren’t announced by the end of 2005 are much too late for this edition.

Conventions

A quick note on the typographical conventions used in this book:

- Type in this font (Minion) is running text.
- Type in this font (Futura) is a sidebar.
- **Type in this font (Courier bold) is used for assembly code and MIPS register names.**
- Type in this font (Courier) is used for C code and hexadecimals.
- **Type in this font (Minion italic, small) is used for hardware signal names.**
Acknowledgments

The themes in this book have followed me through my computing career. Mike Cole got me excited about computing, and I’ve been trying to emulate his skill in picking out good ideas ever since. In the brief but exciting life of Whitechapel Workstations (1983–1988), many colleagues taught me something about computer architecture and about how to design hardware—Bob Newman and Rick Filipkiewicz probably the most. I also have to thank Whitechapel’s salesperson, Dave Gravell, for originally turning me on to MIPS. My fellow engineers during the lifetime of Algorithmics Ltd. (Chris Dearman, Rick Filipkiewicz, Gerald Onions, Nigel Stephens, and Chris Shaw) have to be doubly thanked, both for all I’ve learned through innumerable discussions, arguments, and designs and for putting up with the book’s competition for my time.

Many thanks are due to the reviewers who’ve read chapters over a long period of time: Phil Bourekas of Integrated Device Technology, Inc.; Thomas Daniel of the LSI Logic Corporation; Mike Murphy of Silicon Graphics, Inc.; and David Nagle of Carnegie Mellon University.

On the second edition: I’ve known Paul Cobb for a long time, as we both worked around MIPS companies. Paul contributed material updating the historical survey of MIPS CPUs and the programming chapter and cleaning up the references. In all cases, though, I’ve had a final edit—so any errors are mine.

During the preparation of this edition I’ve been employed by MIPS Technologies Inc. It’s dangerous to pick out some colleagues and not others, but I’ll do it anyway.

Ralf Baechle runs the www.linux-mips.org site, which coordinates MIPS work on the Linux kernel. He’s been very helpful in dispelling some of the illusions I’d formed about Linux: I started off thinking it was like other operating systems... (Robert Love’s Linux Kernel Development book helped too; I warmly recommend it to anyone who wants a more educated guidebook to the kernel).

Thanks to MIPS Technologies and my various managers for being flexible about my time, and to many colleagues at MIPS Technologies (too many to name) who have read and commented on drafts.

Todd Bezenek has been my most persistent colleague/reviewer of this edition. Reviewers outside MIPS Technologies did it for their love and respect for the field: notable contributors were Steven Hill (Reality Diluted, Inc.), Jun Sun (DoCoMo USA Labs), Eric DeVolder, and Sophie Wilson.

Denise Penrose is easily the Best Editor Ever. Not many people in Finsbury Park (my home in North London) can say they’re just flying to San Francisco for brunch with their publisher.

Last but not least, thanks to Carol O’Brien, who was rash enough to marry me in the middle of this rewrite.