Computer Organization and Architecture

Themes and Variations

Alan Clements

COMPUTER ORGANIZATION AND ARCHITECTURE

Themes and Variations

Alan Clements

Teesside University

CENGAGE Learning®

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

CENGAGE
Learning

***Computer Organization and Architecture: Themes and Variations*, First Edition**
Alan Clements

Publisher, Global Engineering: Christopher M. Shortt

Acquisitions Editor: Swati Meherishi

Developmental Editor: Amy Hill, Collaborative Concepts, LLC

Assistant Development Editor: Farah Naseem

Editorial Assistant: Tanya Altieri

Team Assistant: Carly Rizzo

Marketing Manager: Lauren Betsos

Media Editor: Chris Valentine

Content Project Manager: D. Jean Buttrom

Production Service: RPK Editorial Services, Inc.

Copyeditor: Shelly Gerger-Knechtl

Proofreader: Pat Daly

Indexer: Shelly Gerger-Knechtl

Compositor: MPS Limited

Senior Art Director: Michelle Kunkler

Cover and Internal Designer: Rokusek Design

Cover Image: © Dvpodt/Shutterstock

Rights Acquisition Director: Audrey Pettengill

Rights Acquisition Specialist, Text and Image: Amber Hosea

Text and Image Permissions Researcher: Kristiina Paul

Senior Manufacturing Planner: Doug Wilke

Library of Congress Control Number: 2012942868

ISBN-13: 978-1-111-98704-6

ISBN-10: 1-111-98704-1

Cengage Learning
200 First Stamford Place, Suite 400
Stamford, CT 06902
USA

Printed in the United States of America
1 2 3 4 5 6 7 15 14 13 12

Dedication

Dedicated to the memory of my friend, colleague, and mentor
Tomás López Jiménez.

Contents

Preface

The twenty-first century is an age of scientific and technological wonders. Computers have proved to be everything people expected—and more. Bioengineering has unraveled the mysteries of the cell and enabled scientists to synthesize drugs that were inconceivable a decade ago. Nanotechnology provides a glimpse into a world where the computer revolution is combined with engineering at the atomic level to create microscopic autonomous machines that may, one day, be injected into the body to carry out internal repairs. Ubiquitous computing has given us cell phones, MP3 players, and digital cameras that keep us in touch with each other via the Internet. The computer is at the core of almost all modern technologies. This book explains how the computer works.

The discipline called *computing* has been taught in universities since the 1950s. In the beginning, computing was dominated by the large mainframe, and the subject consisted of a study of computers themselves, the operating systems that controlled the computers, languages and their compilers, databases, and business computing. Since then, computing has expanded exponentially and now embraces so many different areas that it's impossible for any university to cover computing in a comprehensive fashion. We have to concentrate on the essential elements of computing. At the heart of this discipline lies the machine itself: the computer. Of course, computing as a theoretical concept could exist quite happily without computers. Indeed, a considerable amount of work on the theoretical foundations of computer science was carried out in the 1930s and 1940s before the computer revolution took place. However, the way in which computing has progressed over the last 40 years is intimately tied up with the rise of the microprocessor. The Internet could not have taken off in the way it has if people didn't have access to very low-cost computers.

Since the computer itself has had such an effect on both the growth of computing and the path computing has taken, it's intuitively reasonable to expect that the computing curriculum should include a course on how computers actually work. University-level Computer Science and Computer Engineering CS programs invariably include a course on how computers work. Indeed, professional and course accreditation bodies specify computer architecture as a core requirement; for example, computer architecture is central to the joint IEEE Computer Society and ACM Computing Curriculum.

Courses dealing with the embodiment or realization of the computer are known by a variety of names. Some call them hardware courses, some call them computer architecture courses, and some call them computer organization courses (with all manner of combinations in between). Throughout this text, I will use the expression *computer architecture* to describe the discipline that studies the way in which computers are designed and how they operate. I will, of course, explain why this discipline has so many different names and point out that the computer can be viewed in different ways.

Like all areas of computer science, the field of computer architecture is advancing rapidly as developments take place in instruction set design, instruction level parallelism, cache memory technology, bus systems, speculative execution, multi-core computing, and so on. We examine all these topics in this book.

Computer architecture underpins computer science; for example, *computer performance* is of greater importance today than ever before, because even those who buy personal computers have to understand systems architecture in order to make the best choice.

Although most students will never design a new computer, today's students need a much broader overview of the computer than their predecessors. Students no longer have to be competent assembly language programmers, but they must understand how buses, interfaces, cache memories, and instruction set architectures determine the performance of a computer system.

Moreover, students with an understanding of computer architecture are better equipped to study other areas of computer science; for example, a knowledge of instruction set architectures gives students a valuable insight into the operation of compilers.

My motivation for writing this book springs from my experience in teaching an intermediate level course in computer architecture at the University of Teesside. I threw away the conventional curriculum that I'd inherited and taught what could be best described as *Great Ideas in Computer Architecture*. I used this course to teach topics that emphasized global concepts in computer science that helped my students with both their operating systems and C courses. This course was very successful, particularly in terms of student motivation.

Anyone writing a text on computer architecture must appreciate that this subject is taught in three different departments: electrical engineering (EE), electrical and computer engineering (ECE), and computer science (CS). These departments have their own cultures and each looks at the computer from their own viewpoint. EE and ECE departments focus on electronics and how the individual components of a computer operate. EE/ECE-oriented texts concentrate on gates, interfaces, signals, and computer organization. Many students in CS departments don't have the requisite background in electronics, so they can't follow texts that emphasize circuit design. Instead, computer science departments place more stress on the relationship between the low-level architecture of the processor and the higher-level abstractions in computer science.

Although it is near impossible to write a text optimized for use in both EE/ECE and CS departments, *Computer Organization and Architecture: Themes and Variations* is an effective compromise that provides sufficient detail at the logic and organizational levels for EE/ECE departments without including the degree of detail that would alienate CS readers.

Undergraduate computer architecture is taught at three levels: introductory, intermediate, and advanced. Some schools teach all three levels, some compress this sequence into two levels, and some provide only an introduction. This text is aimed at students taking first- and second-level courses in computer architecture and at professional engineers who would like an overview of current developments in microprocessor architecture. The only prerequisite is that the reader should be aware of the basic principles of a high-level language such as C and have a knowledge of basic algebra.

It is difficult to pitch a book at precisely the right level. Indeed, such an ideal level doesn't exist. Different students react in different ways to any specific text. If you make a book very focused and follow a narrow curriculum, you appeal only to students on a tiny handful of courses. *Computer Organization and Architecture: Themes and Variations* is well-suited to a wide range of courses, because it covers the basics and some of the more advanced topics in computer architecture.

Features of the Book

Why inflict yet another text on computer architecture on the world? Computer architecture is a fascinating topic. It's all about how you can take vast numbers of a single primitive element such as a NAND gate and make a computer. It's all about how common sense and technology meet. For example, the cache memory that makes processors so fast is conceptually no more complicated than the note on the back of an envelope. Equally, the way in which all processors operate uses a technique invented by Ford for car production: the *pipeline* or production line. I have tried to make the subject interesting and have covered a greater range of topics than absolutely necessary. For example, in this text we will look at memory devices that operate by moving an oxygen atom from one end of a crystal to the other.

The title of this text, *Computer Organization and Architecture*, emphasizes the structure of the complete computer system (CPU, memory, buses, and peripherals). The subtitle *Themes and Variations* indicates that there is a theme (i.e., the computer system) and also variations, for example, the different approaches to increasing the speed of a CPU or to organizing cache memory.

It is often easier to describe something in terms of what it isn't rather than what it is. This book is not concerned with the precise engineering details of microprocessor systems design, interfacing, and peripherals. It certainly isn't an assembly language primer. The central theme of this book is microcomputer *architecture* rather than microprocessor systems design. Computer architecture can be defined, for our present purposes, as the view of a computer seen by the machine language programmer. That is, a computer's architecture takes no account of its actual hardware or implementation and is concerned only with what it does. We will not consider some of the hardware and interfacing aspects of microprocessors, except where they impinge on its architecture (e.g., cache memory, memory management, and the bus).

The Target Architecture

Anyone writing an architecture text has to select a target architecture as a vehicle to teach the fundamentals of machine design and assembly language programming. Professors regularly debate with religious intensity the relative merits of illustrating a course with a real commercial processor or with a hypothetical generic processor. The generic machine is easy to understand and has a shallow learning curve. Students often find that absorbing the fine details of a real processor is time consuming and unrewarding. On the other hand, practical engineering is all about living with the limitations of the real world. Moreover, a real machine teaches students about the design decisions that engineers have to make in order to create a commercially viable product.

In the 1970s and 1980s DEC's PDP-11 minicomputer was widely adopted as a teaching vehicle. The PDP-11 gradually dropped out of the curriculum with the advent of 16-bit microprocessors such as the Motorola 68K. From the academic's point of view, the 68K (loosely based on the earlier PDP-11) was a dream machine, because its architecture is relatively regular and that made it easy for students to write programs in 68K assembly language. A casual observer might have expected the ubiquitous Intel IA32 family, which is found in most PCs, to have played a significant role in computer architecture education. After all, countless students get hands-on experience of Intel's processors. The 80x86 family has never really caught on in the academic world because its complex architecture grew in an *ad-hoc* fashion as each new member of the family was released and this presents students with an excessive burden. Some academics illustrate their course with a high-performance RISC processor, such as MIPS, which is both powerful and easy to understand. Such high-end RISC processors are found in workstations but are relatively unknown to many students (professors have observed that students often request PC-based technology due to their familiarity with the PC). However, RISC processors are used in both high-performance computers and most cell phones.

I have selected the ARM processor as a vehicle to introduce assembly language and computer organization. It is a processor that is powerful, elegant, yet easy to learn. Moreover, development tools for the ARM processor are widely available which means that students can write programs in ARM assembly language and run them in the lab or at home on their PCs.

A strong contender for the role of target architecture in a modern text is Intel's IA64 Itanium processor. This is a device of immense power and sophistication, yet its basic architecture is simpler than the 80x86 family. The rich and innovative features of the Itanium's architecture illustrate numerous concepts found in a computer architecture course–from the data stack to speculative execution, and from pipelining to instruction level parallelism (ILP). Consequently, I also introduce some features of this processor when we look at high-performance computing.

Computer Organization and Architecture: Themes and Variations isn't a conventional computer architecture text. I go beyond the conventional curriculum and cover material that is interesting, important, and relevant. One of my principal objectives is to provide students with an appropriate body of knowledge that they can absorb. All too often, students graduate from university with embarrassingly large gaps in their knowledge. I know of no other text that adopts my approach. For example, all computer architecture texts introduce floating-point arithmetic, yet very few discuss the codes for data compression required to store large volumes of textual and video information, and none describe the MP3 data compression that's at the heart of an entire industry. Similarly, computer architecture texts often lack coverage of architectural features intended to support multimedia applications. Some of the highlights of this text are described below.

History

Books on computer architecture usually have a section on the history of the computer. Such history chapters are often inaccurate and have received criticism from experts in the field. However, I feel that a history chapter is important, because a knowledge of computer history helps students appreciate how and why developments took place. By knowing where computers came from, students are better able to understand how they are likely to develop in the future. In this text, I provide a short overview of the history of computing and include further historical background in the supplementary web-based material that accompanies this book.

OS Support

The operating system is intimately bound up with computer architecture. *Computer Organization and Architecture: Themes and Variations* covers topics in architecture of interest to those who also study operating systems (e.g., memory management, context switching, protection mechanisms).

Multimedia Support

The most important driving force behind modern computer architectures is the growth of multimedia systems with their insatiable demand for high performance and high bandwidths. This text demonstrates how modern architectures have been optimized for multimedia applications. We look at the effect of multimedia applications on both the architectures of computers and the design of buses and computer peripherals, such as hard disks for use in audiovisual applications.

Input/Output Systems

Today's computers are not only much faster than their predecessors, but they also provide more sophisticated means of getting information into and out of the computer. I/O was of relatively little importance when the typical computer was interfaced only to a keyboard, modem, and printer. Computers are now routinely interfaced to peripherals, such as digital video cameras that demand massive data transfer rates. We will look at some of the modern, high-performance I/O systems, such as the USB and FireWire interfaces. We will also delve more deeply into input/output-related topics such as handshaking and buffering.

Computer Memory Systems

Memory is the *Cinderella* of the computer world. Without high-density, high-performance memory systems, neither low-cost desktop systems nor digital cameras with 32GB of storage would be possible. I have divided memory systems into two chapters: the first dealing with semiconductor memory and the second dealing with magnetic and optical memory. We will also take a look at some of the interesting emerging memory technologies, such as Ovonic memory and ferroelectric memory.

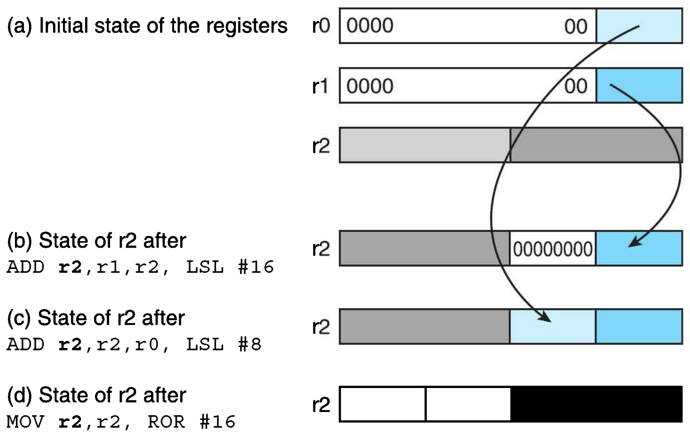
Approach

The books that I've most enjoyed are those where a little of the author's personality and view of the subject shines through. I hope that the same is true of this book. Computer architecture isn't something that can be expressed as a set of cold equations to be learned; it is a culture that has developed over the years. At conferences, you will meet academics who passionately argue the relative merits of this computer over that one. I would be a poor educator if I did not at least hint to students that computer architecture can be as much fad as fact.

It's also true that different authors emphasize different topics. For example, most authors stress the design of the processor and have relatively little to say about memory, buses, and peripherals–even though all of these elements are necessary to create a computer system. Perhaps some feel that one aspect of a computer is more intrinsically interesting than others. I provide more coverage of memory, buses, and interfaces than many texts, because I feel that these topics are as important as the processor itself. Similarly, I've included details of memory elements such as Ovonic devices, which store data by melting a bead of glass and then storing a 1 or 0 by selecting how fast it cools. This is such a remarkable example of engineering ingenuity that I felt I had to include it. I'd like students to share the enthusiasm I have for this subject.

I find that a significant shortcoming of many texts is the quality of diagrams and illustrations. All too often a figure has far too little annotation, and its meaning is entirely lost. I have drawn virtually all of the included diagrams myself, and I hope that they do a good job in illustrating the meaning of the text.

Here's an example of a diagram that illustrates the effect of a sequence of three instructions on a register. The purpose of the code is to take a byte from two registers and then concatenate them in a third register. The use of color makes it easy to see how the data is being processed.



Contents Overview

I have divided the book into five parts.

Part I: An Introduction to Computer Architecture covers the enabling material that allows us to discuss computer architecture. Chapter 1 takes an unusual approach to the introduction of a computer. I begin by presenting a problem, solving it, and then inventing a system to implement the solution–the computer. My aim is to demonstrate that the computer closely models the way in which we solve problems. Because so many topics in computer architecture are interrelated, I provide a brief overview of the computer system to allow me to mention topics, like cache memory, before I discuss them in detail.

We will look at the way in which information is encoded and represented in binary form; for example, I introduce the computer arithmetic of both signed and unsigned integers, demonstrate how floating-point numbers represent very large and very small quantities, and then briefly introduce more complex mathematical functions. An introduction to gates and logic design is also included in Chapter 2. This topic always presents the author of a text on computer architecture with some difficulty. Should logic be omitted because some students take a separate course on logic design? Or should logic be relegated to an appendix so that only those who require it need delve into it? I've decided to put it in this section, because it allows a natural progression from numbers to computers and helps students understand some of the material that follows. Even students who have already taken a course in logic should go over this chapter lightly. I end the section on logic design with a very simple proto-computer.

In an ideal world, Part I would include a detailed history of the way in which computers have developed from simple mechanical calculators to today's powerful processors, because all students should have an understanding of how the profession developed. The history of the computer is a fascinating story that begins with a desire to make tedious arithmetical calculations easier, then moves to the telegraph and transatlantic cable project that did so much to "shrink" the world. In the last century, modern computing grew out of the technology used to create the telephone system and was driven by the need for high-speed calculation by scientists, the business community, and the military. No other human artifact has developed as fast as the computer in terms of its increase in performance and its decrease in cost. It is said that if the automobile had developed as rapidly as the computer we'd be driving cars at many times the speed of light using a single drop of gasoline a year.

Unfortunately, I just can't do justice to the history of the computer here. Students have to cover a certain body of knowledge in a computer architecture course, and they have to develop a range of skills in order to become practitioners in their chosen discipline. Consequently, I have located the material on computer history on the book's companion website, the details of which can be found in the Supplements and Resources section below. . . All students are strongly encouraged to read this material. Computer scientists have a tendency to reinvent the wheel, so concepts developed one year sometimes appear in a different context the following year.

Part II: Instruction Set Architectures is the heart of this course and includes three chapters. I first introduce the concept of an instruction set and then examine some of the important issues in computer architecture, such as the data structures that are accessed and manipulated by computers. Chapter 3 introduces the ARM family of microprocessors that have a truly elegant architecture and include facilities allowing us to cover several interesting topics, such as ***predicated execution*** where an instruction is executed only if certain conditions are met. It would be nice to cover all of the variations in the architecture of the computer, but unless you are going to dedicate the rest of your life to that end, there is not time. In Chapter 4, we look at some of the variations in computer architecture. For example, memory indirect addressing modes that let you access complex data structures with remarkable ease, and special compressed-code processors that squeeze conventional code to a fraction of its normal size.

The final chapter of Part II looks at the way in which processors have been adapted to suit modern multimedia applications such as video encoding and decoding. In this chapter, we also take a brief look at the background of multimedia processing and explain why high-performance computing is so necessary.

Part III: Organization and Efficiency is concerned with computer organization; that is, how computers work and are organized internally. Chapter 6 begins with an introduction to performance, which is the way the speed of computers is measured and how computers are compared. Some might argue that performance should be introduced earlier because of its importance. However, because so many topics influence performance, I thought it is better placed here where the reader is able to understand more of the background.

Chapter 7 looks at how computers actually work. We begin by looking at microprogramming, which is a technique that makes it possible to design computers of arbitrary complexity and is still used to implement some of the more complex instructions in Intel's IA32 processors. Then we introduce pipelining, which is fundamental to all modern computer design. Pipelining mirrors the industrial production line where instructions are executed bit by bit as they flow through the computer. However, pipelining encounters problems when non-sequential instructions (branches) are encountered. The final part of Chapter 7 looks at how the problems caused by branch instructions can be limited.

Chapter 8 continues from where Chapter 7 left off. Here we look at how the performance of processors has been raised by executing instructions in parallel and even out-of-order. In Chapter 8, I introduce the very long instruction word computer (VLIW), which bundles several operations into a single instruction and executes them in parallel. As an example of modern high-performance computing, I introduce the remarkable IA64 Itanium computer that incorporates several powerful ways of accelerating the processor. You could even call the Itanium *a computer architecture course on a chip*.

Part IV: The Computer System turns our attention away from the central processing unit and devotes four chapters to the computer system, concentrating on memory, buses, and input/output. Chapter 9 focuses on two related topics: cache and virtual memory. Although the capacity of memory systems has expanded rapidly over the last few decades, their speed or access time has not improved at anything like the same rate as the CPUs. This situation has led to a bottleneck where memories can't provide data at the speed the CPUs can process it. Cache technology has been developed to make the best of a bad job by using a small amount of fast memory to do the work of a large amount of fast memory. I show how cache memories operate and describe their principal features. Chapter 9 also looks at virtual memory, which integrates main store with disk storage.

Chapters 10 and 11 cover memory technologies ranging from static semiconductor memory to disk and optical storage technologies.

Chapter 12 concludes Part IV by looking at the strategies used to get information into and out of the computer and then describing some of the modern high-speed interfaces used to support multimedia systems.

One of the key components of the modern computer is the bus, which distributes information between the various functional parts of the system. Indeed, the bus is one of the components most critical to the computer's performance. We look at the structure of computer buses, their functionality, and the way in which they permit competing devices to access the bus in multiprocessor systems. Finally, I describe one of the world's most popular high-performance buses: the PCI bus found in PCs.

Part V: The Near Future looks at the next direction in computer architecture, multiprocessing. Once upon a time, you could make a powerful computer by interconnecting an array of individual processors. Such computers had astronomical price tags. Today, it's possible to fabricate several processors on a single chip. Such multi-core processors are becoming increasingly popular, because they both increase processing power and reduce the amount of electrical energy needed. In Chapter 13, I provide a broad overview of multiprocessing covering the background, typical devices, interconnection topology, and the software of parallel processing.

Supplements and Resources

This book comes with a host of support material for both the instructor and the student. The supplements are housed on the book's companion website. To access the additional course materials, please visit www.cengagebrain.com. At the cengagebrain.com home page, search for the ISBN of your title (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where these resources can be found.

Instructor Resources

An instructor's solutions manual (ISM) is available in both print and digital formats. The digital version is available to registered instructors at the publisher's website. This website also includes both a full set of PowerPoint slides containing all graphical images and tables in the text, and a set of LectureBuilder PowerPoint slides of all equations and example problems.

Student Resources

The student resources for this product include:

- a student workbook that provides a detailed introduction to writing programs in ARM processor assembly language and then running them on a simulator;
- additional study material, including a chapter on computer history and an overview of Karnaugh maps;
- useful links, including a link to download the student version of the Kiel simulator from ARM;
- code segments from the book; and
- handouts for all the lecture slides posted on the instructor website.

Acknowledgments

No one writes a textbook in a vacuum. All subjects have a history, background, and culture, and computer architecture is no exception. An author can go along with the flow or take a new direction. I could not have written this book without the contribution of all those texts that have preceded this, for example, the texts I used to study computer architecture myself when I was a student. Equally, I have to acknowledge the countless researchers that contributed to the body of knowledge making up computer architecture. The role of the writer is to take all of this information and create a path for students to follow. The writer has to decide what information is important and what can be omitted, what trends should be followed, and what trends can be relegated to the background. However, the writer is indebted to all those who have contributed to the body of knowledge.

Many people are involved with the production of a book as complex as this, but one stands out, the acquisitions editor. It was Swati Meherishi at Cengage Learning who began the long process of transforming a rough manuscript into the final polished article. An acquisitions editor has the ability to see beyond the manuscript and to understand how the book will fit into a complicated market. Swati had sufficient faith in me to help me endure hours of endless writing and rewriting. I'd like to thank Swati for putting up with me. The other key person in the creation of this book is the developmental editor, Amy Hill. Amy has spent a lot of time going through my manuscript to improve its structure and the way in which it all fits together. She has provided me with unfailingly good advice and gentle guidance. I appreciate the hard work and dedication of the entire team at Cengage Learning; without them this project would have remained a set of files on my hard disk. Also deserving of thanks are Rose Kernan and her team at RPK Editorial Services for their smooth management of all production related tasks, and Kristiina Paul for painstakingly researching and obtaining permissions for all third party material used in the book.

I must thank all the reviewers and editors that helped review and correct the manuscript. They made great suggestions about ways of improving it and helped point me in the right direction when I had misinterpreted source material.

I would like to acknowledge the technical reviewers and those who carefully read the text for errors and omissions. In particular, Loren Schwiebert did a great job of debugging Chapter 3. Sohum Sohoni of Oklahoma State University painstakingly provided a technical review of the entire manuscript. I must thank him for his hard work, especially for his excellent guidance on Chapter 8. My thanks also to Shuo Qin of the University of Southern California, who worked all the homework problems and checked the instructor's solution manual for errors.

The manuscript was reviewed at various stages of development by a number of instructors. I am thankful to all of them for their constructive comments. Below are the names of those who chose to be acknowledged:

Mokhtar Aboelaze
York University

Manoj Franklin
University of Maryland–College Park

Israel Koren
University of Massachusetts–Amherst

Mikko Lipasti
University of Wisconsin–Madison

Rabi Mahapatra
Texas A&M University

Xiannong Meng
Bucknell University

Prabhat Mishra
University of Florida

William Mongan
Drexel University

Vojin G. Oklobdzija
Erik Jonsson School of Engineering,
University of Texas-Dallas

Soner Onder
Michigan Technological University

Füsun Özgüner
Ohio State University

Richard J. Povinelli
Marquette University

Norman Ramsey
Tufts University

Bill Reid
Clemson University

William H. Robinson
Vanderbilt University

Carolyn J C Schauble
Colorado State University

Aviral Shrivastava
Arizona State

Sohum Sohoni
Oklahoma State University

Nozar Tabrizi
Kettering University

Dean Tullsen
University of California–San Diego

Charles Weems
Univ. of Mass.

Bilal Zafar
University of Southern California

Huiyang Zhou
North Carolina State University

Finally, I'd like to thank my wife Sue for her help in debugging the manuscript. Although Sue does not have a technical background, she has been very helpful in removing some of the ambiguities and clumsiness in my use of English.

Feedback from the readers, both critical and appreciative, is welcome. I would be particularly interested in suggestions for additional material that can be added to the companion website in order to help students with their studies. Please send your comments, concerns, and suggestions to globalengineering@cengage.com. You may also contact me directly at alanclements@ntlworld.com.

Alan Clements