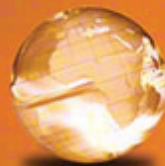


GLOBAL
EDITION



Digital Design

*With an Introduction to the Verilog
HDL, VHDL, and SystemVerilog*

SIXTH EDITION

M. Morris Mano • Michael D. Ciletti



Pearson

Digital Design

With an Introduction to the Verilog HDL,
VHDL, and SystemVerilog

This page intentionally left blank

Digital Design

With an Introduction to the Verilog HDL,
VHDL, and SystemVerilog

SIXTH EDITION
GLOBAL EDITION

M. Morris Mano

*Emeritus Professor of Computer Engineering
California State University, Los Angeles*

Michael D. Ciletti

*Emeritus Professor of Electrical and Computer Engineering
University of Colorado at Colorado Springs*



330 Hudson Street, NY NY 10013

Senior Vice President Courseware Portfolio
Management: *Marcia J. Horton*
Director, Portfolio Management: *Engineering, Computer
Science & Global Editions: Julian Partridge*
Higher Ed Portfolio Management: *Tracy Johnson
(Dunkelberger)*
Portfolio Management Assistant: *Kristy Alaura*
Assistant Acquisitions Editor, Global Edition: *Aditee
Agarwal*
Assistant Project Editor, Global Edition: *Aurko Mitra*
Managing Content Producer: *Scott Disanno*
Content Producer: *Robert Engelhardt*
Senior Manufacturing Controller, Global Edition: *Kay
Holman*

Web Developer: *Steve Wright*
Media Production Manager, Global Edition: *Vikram
Kumar*
Rights and Permissions Manager: *Ben Ferrini*
Manufacturing Buyer, Higher Ed, Lake Side
Communications Inc (LSC): *Maura Zaldivar-Garcia*
Inventory Manager: *Ann Lam*
Marketing Manager: *Demetrius Hall*
Product Marketing Manager: *Yvonne Vannatta*
Marketing Assistant: *Jon Bryant*
Cover Designer: *Lumina Datamatics, Inc.*
Cover Photo: *spainter_vfx/Shutterstock*
Full-Service Project Management: *Vimala Vinayakam,
SPi Global*

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text

Pearson Education Limited
KAO Two
KAO Park
Harlow
CM17 9NA
United Kingdom

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

© Pearson Education Limited 2019

The rights of M. Morris Mano and Michael D. Ciletti to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog, 6th Edition, ISBN 978-0-13-454989-7 by M. Morris Mano and Michael D. Ciletti, published by Pearson Education © 2018.

All rights reserved. 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, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

Typeset by SPi Global
Printed and bound by Vivar in Malaysia

ISBN 10: 1-292-23116-5
ISBN 13: 978-1-292-23116-7

Contents

Preface		9
1	Digital Systems and Binary Numbers	17
1.1	Digital Systems	17
1.2	Binary Numbers	20
1.3	Number-Base Conversions	22
1.4	Octal and Hexadecimal Numbers	25
1.5	Complements of Numbers	27
1.6	Signed Binary Numbers	33
1.7	Binary Codes	38
1.8	Binary Storage and Registers	47
1.9	Binary Logic	50
2	Boolean Algebra and Logic Gates	57
2.1	Introduction	58
2.2	Basic Definitions	58
2.3	Axiomatic Definition of Boolean Algebra	59
2.4	Basic Theorems and Properties of Boolean Algebra	63
2.5	Boolean Functions	66
2.6	Canonical and Standard Forms	72
2.7	Other Logic Operations	81
2.8	Digital Logic Gates	83
2.9	Integrated Circuits	89

6 Contents

3 Gate-Level Minimization 98

3.1	Introduction	99
3.2	The Map Method	99
3.3	Four-Variable K-Map	106
3.4	Product-of-Sums Simplification	111
3.5	Don't-Care Conditions	115
3.6	NAND and NOR Implementation	118
3.7	Other Two-Level Implementations	126
3.8	Exclusive-OR Function	131
3.9	Hardware Description Languages (HDLs)	137
3.10	Truth Tables in HDLs	154

4 Combinational Logic 163

4.1	Introduction	164
4.2	Combinational Circuits	164
4.3	Analysis of Combinational Circuits	165
4.4	Design Procedure	169
4.5	Binary Adder–Subtractor	172
4.6	Decimal Adder	184
4.7	Binary Multiplier	186
4.8	Magnitude Comparator	188
4.9	Decoders	191
4.10	Encoders	195
4.11	Multiplexers	198
4.12	HDL Models of Combinational Circuits	205
4.13	Behavioral Modeling	231
4.14	Writing a Simple Testbench	239
4.15	Logic Simulation	245

5 Synchronous Sequential Logic 261

5.1	Introduction	262
5.2	Sequential Circuits	262
5.3	Storage Elements: Latches	264
5.4	Storage Elements: Flip-Flops	269
5.5	Analysis of Clocked Sequential Circuits	277
5.6	Synthesizable HDL Models of Sequential Circuits	291
5.7	State Reduction and Assignment	316
5.8	Design Procedure	321

6 Registers and Counters 342

6.1	Registers	342
6.2	Shift Registers	346

6.3	Ripple Counters	354
6.4	Synchronous Counters	359
6.5	Other Counters	367
6.6	HDL Models of Registers and Counters	372
7	Memory and Programmable Logic	393
7.1	Introduction	394
7.2	Random-Access Memory	395
7.3	Memory Decoding	402
7.4	Error Detection and Correction	407
7.5	Read-Only Memory	410
7.6	Programmable Logic Array	416
7.7	Programmable Array Logic	420
7.8	Sequential Programmable Devices	424
8	Design at the Register Transfer Level	445
8.1	Introduction	446
8.2	Register Transfer Level (RTL) Notation	446
8.3	RTL Descriptions	448
8.4	Algorithmic State Machines (ASMs)	466
8.5	Design Example (ASMD CHART)	475
8.6	HDL Description of Design Example	485
8.7	Sequential Binary Multiplier	503
8.8	Control Logic	508
8.9	HDL Description of Binary Multiplier	514
8.10	Design with Multiplexers	529
8.11	Race-Free Design (Software Race Conditions)	545
8.12	Latch-Free Design (Why Waste Silicon?)	548
8.13	SystemVerilog—An Introduction	549
9	Laboratory Experiments with Standard ICs and FPGAs	571
9.1	Introduction to Experiments	571
9.2	Experiment 1: Binary and Decimal Numbers	576
9.3	Experiment 2: Digital Logic Gates	579
9.4	Experiment 3: Simplification of Boolean Functions	581
9.5	Experiment 4: Combinational Circuits	583
9.6	Experiment 5: Code Converters	584
9.7	Experiment 6: Design with Multiplexers	586
9.8	Experiment 7: Adders and Subtractors	588
9.9	Experiment 8: Flip-Flops	591
9.10	Experiment 9: Sequential Circuits	593
9.11	Experiment 10: Counters	595

8 Contents

9.12	Experiment 11: Shift Registers	596
9.13	Experiment 12: Serial Addition	600
9.14	Experiment 13: Memory Unit	601
9.15	Experiment 14: Lamp Handball	603
9.16	Experiment 15: Clock-Pulse Generator	607
9.17	Experiment 16: Parallel Adder and Accumulator	609
9.18	Experiment 17: Binary Multiplier	611
9.19	HDL Simulation Experiments and Rapid Prototyping with FPGAs	615

10 Standard Graphic Symbols 621

10.1	Rectangular-Shape Symbols	621
10.2	Qualifying Symbols	624
10.3	Dependency Notation	626
10.4	Symbols for Combinational Elements	628
10.5	Symbols for Flip-Flops	630
10.6	Symbols for Registers	632
10.7	Symbols for Counters	635
10.8	Symbol for RAM	637

Appendix 640

Answers to Selected Problems 654

Index 699

Preface

The speed, density, and complexity of today's digital devices are made possible by advances in physical processing technology and digital design methodology. Aside from semiconductor technology, the design of leading-edge devices depends critically on hardware description languages (HDLs) and synthesis tools. Three public-domain languages, *Verilog*, *VHDL*, and *SystemVerilog*, all play a role in design flows for today's digital devices. HDLs, together with fundamental knowledge of digital logic circuits, provide an entry point to the world of digital design for students majoring in computer science, computer engineering, and electrical engineering.

In the not-too-distant past, it would be unthinkable for an electrical engineering student to graduate without having used an oscilloscope. Today, the needs of industry demand that undergraduate students become familiar with the use of at least one hardware description language. Their use of an HDL as a student will better prepare them to be productive members of a design team after they graduate.

Given the presence of three HDLs in the design arena, we have expanded our presentation of HDLs in *Digital Design* to treat *Verilog* and *VHDL*, and to provide an introduction to *SystemVerilog*. Our intent is not to require students to learn three, or even two, languages, but to *provide the instructor with a choice between Verilog and VHDL while teaching a systematic methodology for design, regardless of the language, and an optional introduction to SystemVerilog*. Certainly, *Verilog* and *VHDL* are widely used and taught, dominate the design space, and have common underlying concepts supporting combinational and sequential logic design, and both are essential to the synthesis of high-density integrated circuits. **Our text offers *parallel tracks of presentation of both languages, but allows concentration on a single language***. The level of treatment of *Verilog* and *VHDL* is essentially equal, without emphasizing one language over the other. **A language-neutral presentation of digital design is a common**

thread through the treatment of both languages. A large set of problems, which are stated in language-neutral terms, at the end of each chapter can be worked with either Verilog or VHDL.

The emphasis in our presentation is on digital design, with HDLs in a supporting role. Consequently, we present only those details of *Verilog*, *VHDL*, and *SystemVerilog* that are needed to support our treatment of an *introduction* to digital design. Moreover, although we present examples using each language, we identify and segregate the treatment of topics and examples so that *the instructor can choose a path of presentation for a single language*—either *Verilog* or *VHDL*. Naturally, a path that emphasizes *Verilog* can conclude with *SystemVerilog*, but it can be skipped without compromising the objectives. The introduction to *SystemVerilog* is selective—we present only topics and examples that are extensions of *Verilog*, and well within the scope of an introductory treatment. To be clear, we are *not* advocating simultaneous presentation of the languages. The instructor can choose either *Verilog/SystemVerilog* or *VHDL* as the core language supporting an introductory course in digital design. **Regardless of the language, our focus is on digital design.**

The language-based examples throughout the book are not just about the details of an HDL. We emphasize and demonstrate the modeling and verification of digital circuits having specified behavior. **Neither Verilog or VHDL are covered in their entirety.** *Some details of the languages will be left to the reader's continuing education and use of web resources.* Regardless of language, our examples introduce a design methodology based on the concept of computer-aided modeling of digital systems by means of a mainstream, IEEE-standardized, hardware description language.

This revision of *Digital Design* begins each chapter with a statement of its objectives. Problems at the end of each chapter are combined with in-chapter examples, and with in-chapter Practice Exercises. Together, these encounters with the subject matter bring the student closer to achieving the stated objectives and becoming skilled in digital design. Answers are given to selected problems at the end of the book. A solution manual gives detailed solutions to all of the problems at the end of the chapters. The level of detail of the solutions is such that an instructor can use individual problems to support classroom instruction.

MULTIMODAL LEARNING

Like the previous editions, this edition of *Digital Design* supports a multimodal approach to learning. The so-called VARK^{1,2} characterization of learning modalities identifies four major modes by which we learn: (V) visual, (A) aural (hearing), (R) reading, and (K) kinesthetic. The relatively high level of illustrations and graphical content of our text

¹Kolb, David A. (2015) [1984]. *Experiential learning: Experience as the source of learning and development* (2nd ed.). Upper Saddle River, NJ: Pearson Education. ISBN 9780133892406. OCLC 909815841.

²Fleming, Neil D. (2014). “The VARK modalities?” *vark-learn.com*.

addresses the visual (V) component of VARK; discussions and numerous examples address the reading (R) component. Students who exploit the availability of free Verilog, VHDL and SystemVerilog simulators and synthesis tools to work assignments are led through a kinesthetic learning experience, including the delight of designing a digital circuit that actually works. The remaining element of VARK, the aural/auditory (A) experience depends on the instructor and the attentiveness of the student (Put away the smart phone!). We have provided an abundance of materials and examples to support classroom lectures. Thus, a course using *Digital Design*, can provide a rich, balanced, learning experience and address all the modes identified by VARK.

For skeptics who might still question the need to present and use HDLs in a first course in digital design, we note that industry does not rely on schematic-based design methods. Schematic entry creates a representation of functionality that is implicit in the constructs and layout of the schematic. Unfortunately, it is difficult for anyone in a reasonable amount of time to determine the functionality represented by the schematic of a logic circuit without having been instrumental in its construction, or without having additional documentation expressing the design intent. Consequently, industry today relies almost exclusively on HDLs to describe the functionality of a design and to serve as a basis for documenting, simulating, testing, and synthesizing the hardware implementation of the design in a standard cell-based ASIC or an FPGA. The utility of a schematic depends on the detailed documentation of a carefully constructed hierarchy of design units. In the past, designers relied on their years of experience to create a schematic of a circuit to implement functionality. Today's designers using HDLs, can express functionality directly and explicitly, without years of accumulated experience, and use synthesis tools to generate the schematic as a byproduct, automatically. Industry adopted HDL-based design flows because schematic entry dooms us to inefficiency, if not failure, in understanding and designing large, complex, ICs.

Introduction of HDLs in a first course in digital design is not intended to replace fundamental understanding of the building blocks of such circuits, or to eliminate a discussion of manual methods of design. It is still essential for students to understand *how hardware works*. Thus, this edition of *Digital Design* retains a thorough treatment of combinational and sequential logic design and a foundation in Boolean algebra. Manual design practices are presented, and their results are compared with those obtained using HDLs. What we are presenting, however, is an emphasis on how hardware is designed today, to better prepare a student for a career in today's industry, where HDL-based design practices are dominant.

FLEXIBILITY

We include both manual and HDL-based design examples. Our end-of-chapter problems cross-reference problems that access a manual design task with a companion problem that uses an HDL to accomplish the assigned task. We also link the manual and HDL-based approaches by presenting annotated results of simulations in the text, in answers to selected problems at the end of the text, and extensively in the solution manual.

NEW TO THIS EDITION

This edition of *Digital Design* uses the latest features of IEEE Standard 1364, but only insofar as they support our pedagogical objectives. The revisions and updates to the text include:

- Elimination of specialized circuit-level content not typically covered in a first course in logic circuits and digital design (e.g., RTL, DTL, and emitter-coupled logic circuits)
- Addition of “Web Search Topics” at the end of each chapter to point students to additional subject matter available on the web
- Revision of approximately one-third of the problems at the end of the chapters
- A solution manual for the entire text, including all new problems
- Streamlining of the discussion of Karnaugh maps
- Integration of treatment of basic CMOS technology with treatment of logic gates
- Inclusion of an appendix introducing semiconductor technology
- Treatment of digital design with VHDL and SystemVerilog

DESIGN METHODOLOGY

A highlight of our presentation is a systematic methodology for designing a state machine to control the data path of a digital system. The framework in which this material is presented treats the realistic situation in which status signals from the datapath are used by the controller, i.e., the system has feedback. Thus, our treatment provides a foundation for designing complex and interactive digital systems. Although it is presented with an emphasis on HDL-based design, the methodology is also applicable to manual-based approaches to design and is language-neutral.

JUST ENOUGH HDL

We present only those elements of Verilog, VHDL, and SystemVerilog that are matched to the level and scope of this text. Also, correct syntax does not guarantee that a model meets a functional specification or that it can be synthesized into physical hardware. So, we introduce students to a disciplined use of industry-based practices for writing models to ensure that a behavioral description can be synthesized into physical hardware, and that the behavior of the synthesized circuit will match that of the behavioral description. Failure to follow this discipline can lead to software race conditions in the HDL models of such machines, race conditions in the test bench used to verify them, and a mismatch between the results of simulating a behavioral model and its synthesized physical counterpart. Similarly, failure to abide by industry practices may lead to designs that simulate correctly, but which have hardware latches that are introduced into the design accidentally as a consequence of the modeling style used by the designer. The industry-based methodology we present leads to race-free and latch-free designs. It is important that students learn and follow industry practices in using HDL models, independent of whether a student’s curriculum has access to synthesis tools.

VERIFICATION

In industry, significant effort is expended to verify that the functionality of a circuit is correct. Yet not much attention is given to verification in introductory texts on digital design, where the focus is on design itself, and testing is perhaps viewed as a secondary undertaking. Our experience is that this view can lead to premature “high-fives” and declarations that “the circuit works beautifully.” Likewise, industry gains repeated returns on its investment in an HDL model by ensuring that it is readable, portable, and reusable. We demonstrate naming practices and the use of parameters to facilitate reusability and portability. We also provide test benches for all of the solutions and exercises to (1) verify the functionality of the circuit; (2) underscore the importance of thorough testing; and (3) introduce students to important concepts, such as self-checking test benches. Advocating and illustrating the development of a *test plan* to guide the development of a test bench, we introduce test plans, albeit simply, in the text and expand them in the solutions manual and in the answers to selected problems at the end of the text.

HDL CONTENT

We have ensured that all examples in the text and all answers in the solution manual conform to accepted industry practices for modeling digital hardware. As in the previous edition, HDL material is inserted in separate sections so that it can be covered or skipped as desired, does not diminish treatment of manual-based design, and does not dictate the sequence of presentation. The treatment is at a level suitable for beginning students who are learning digital circuits and an HDL at the same time. The text prepares students to work on significant independent design projects and to succeed in a later course in computer architecture and advanced digital design.

Instructor Resources

Instructors can obtain the following classroom-ready resources from the publisher:

- Source code and test benches for all Verilog HDL examples in the text
- All figures and tables in the text
- Source code for all HDL models in the solutions manual
- A downloadable solutions manual with graphics suitable for classroom presentation

HDL Simulators

Two free simulators can be downloaded from www.Syncad.com. The first simulator is *VeriLogger Pro*, a traditional Verilog simulator that can be used to simulate the HDL examples in the book and to verify the solutions of HDL problems. This simulator accepts the syntax of the IEEE-1995 standard and will be useful to those who have legacy models. As an interactive simulator, *VeriLogger Extreme* accepts the syntax of IEEE-2001 as well as IEEE-1995, allowing the designer to simulate and analyze design

ideas before a complete simulation model or schematic is available. This technology is particularly useful for students because they can quickly enter Boolean and *D* flip-flop or latch input equations to check equivalency or to experiment with flip-flops and latch designs. Free design tools that support design entry, simulation and synthesis (of FPGAs) are available from www.altera.com and from www.xilinx.com.

Chapter Summary

The following is a brief summary of the topics that are covered in each chapter.

Chapter 1 presents the various binary systems suitable for representing information in digital systems. The binary number system is explained and binary codes are illustrated. Examples are given for addition and subtraction of signed binary numbers and decimal numbers in binary-coded decimal (BCD) format.

Chapter 2 introduces the basic postulates of Boolean algebra and shows the correlation between Boolean expressions and their corresponding logic diagrams. All possible logic operations for two variables are investigated, and the most useful logic gates used in the design of digital systems are identified. This chapter also introduces basic CMOS logic gates.

Chapter 3 covers the map method for simplifying Boolean expressions. The map method is also used to simplify digital circuits constructed with AND–OR, NAND, or NOR gates. All other possible two-level gate circuits are considered, and their method of implementation is explained. Verilog and VHDL are introduced together with simple examples of gate-level models.

Chapter 4 outlines the formal procedures for the analysis and design of combinational circuits. Some basic components used in the design of digital systems, such as adders and code converters, are introduced as design examples. Frequently used digital logic functions such as parallel adders and subtractors, decoders, encoders, and multiplexers are explained, and their use in the design of combinational circuits is illustrated. HDL examples are given in gate-level, dataflow, and behavioral models to show the alternative ways available for describing combinational circuits in Verilog and VHDL. The procedure for writing a simple test bench to provide stimulus to an HDL design is presented.

Chapter 5 outlines the formal procedures for analyzing and designing clocked (synchronous) sequential circuits. The gate structure of several types of flip-flops is presented together with a discussion on the difference between level and edge triggering. Specific examples are used to show the derivation of the state table and state diagram when analyzing a sequential circuit. A number of design examples are presented with emphasis on sequential circuits that use D-type flip-flops. Behavioral modeling in Verilog and VHDL for sequential circuits is explained. HDL examples are given to illustrate Mealy and Moore models of sequential circuits.

Chapter 6 deals with various sequential circuit components such as registers, shift registers, and counters. These digital components are the basic building blocks from which more complex digital systems are constructed. HDL descriptions of shift registers and counters are presented.

Chapter 7 introduces random access memory (RAM) and programmable logic devices. Memory decoding and error correction schemes are discussed. Combinational and sequential programmable devices such as ROMs, PLAs, PALs, CPLDs, and FPGAs are presented.

Chapter 8 deals with the register transfer level (RTL) representation of digital systems. The algorithmic state machine (ASM) chart is introduced. A number of examples demonstrate the use of the ASM chart, ASMD chart, RTL representation, and HDL description in the design of digital systems. The design of a finite state machine to control a datapath is presented in detail, including the realistic situation in which status signals from the datapath are used by the state machine that controls it. This chapter provides the student with a systematic approach to more advanced design projects.

Chapter 9 presents experiments that can be performed in the laboratory with hardware that is readily available commercially. The operation of the ICs used in the experiments is explained by referring to diagrams of similar components introduced in previous chapters. Each experiment is presented informally and the student is expected to design the circuit and formulate a procedure for checking its operation in the laboratory. The lab experiments can be used in a stand-alone manner too and can be accomplished by a traditional approach, with a breadboard and TTL circuits, or with an HDL/synthesis approach using FPGAs. Today, software for synthesizing an HDL model and implementing a circuit with an FPGA is available at no cost from vendors of FPGAs, allowing students to conduct a significant amount of work in their personal environment before using prototyping boards and other resources in a lab. Circuit boards for rapid prototyping circuits with FPGAs are available at a nominal cost, and typically include push buttons, switches, seven-segment displays, LCDs, keypads, and other I/O devices. With these resources, students can work prescribed lab exercises or their own projects and get results immediately.

Chapter 10 presents the standard graphic symbols for logic functions recommended by an ANSI/IEEE standard. These graphic symbols have been developed for small-scale integration (SSI) and medium-scale integration (MSI) components so that the user can recognize each function from the unique graphic symbol assigned. The chapter shows the standard graphic symbols of the ICs used in the laboratory experiments.

Acknowledgments

We are grateful to the reviewers of *Digital Design*, 6e. Their expertise, careful reviews, and suggestions helped shape this edition.

Vijay Madiseti, Georgia Tech

Dmitri Donetski, SUNY Stony Brook

David Potter, Northeastern

Xiaolong Wu, California State-Long Beach

Avinash Kodi, Ohio University

Lee Belfore, Old Dominion University

16 Preface

We also wish to express our gratitude to the editorial and publication team at Pearson Education for supporting this edition of our text. We are grateful, too, for the ongoing support and encouragement of our wives, Sandra and Jerilynn.

M. MORRIS MANO
Emeritus Professor of Computer Engineering
California State University, Los Angeles

MICHAEL D. CILETTI
Emeritus Professor of Electrical and Computer Engineering
University of Colorado at Colorado Springs

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge the following people for their contributions to this Global Edition.

Contributors

Moumita Mitra Manna, Bangabasi College

Reviewers

Debaprasad Das, Assam University

Nikhil Marriwala, University Institute of Engineering and Technology, Kanpur

Tapas Kumar Saha, National Institute of Technology, Durgapur

Chapter 1

Digital Systems and Binary Numbers

CHAPTER OBJECTIVES

1. Understand binary number system.
2. Know how to convert between binary, octal, decimal, and hexadecimal numbers.
3. Know how to take the complement and reduced radix complement of a number.
4. Know how to form the code of a number.
5. Know how to form the parity bit of a word.

1.1 DIGITAL SYSTEMS

Digital systems have such a prominent role in everyday life that we refer to the present technological period as the *digital age*. Digital systems are used in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises. We have digital telephones, digital televisions, digital versatile discs (DVDs), digital cameras, personal, handheld, touch-screen devices, and, of course, digital computers. We enjoy music downloaded to our portable media player (e.g., iPod Touch[®]) and other handheld devices having high-resolution displays and touch-screen graphical user interfaces (GUIs). GUIs enable them to execute commands that appear to the user to be simple, but which, in fact, involve precise execution of a sequence of complex internal instructions. Most, if not all, of these devices have a special-purpose digital computer, or

processor, embedded within them. The most striking property of the digital computer is its generality. It can follow a sequence of instructions, called a program, which operates on given data. The user can specify and change the program or the data according to the specific need. Because of this flexibility, general-purpose digital computers can perform a variety of information-processing tasks that range over a wide spectrum of applications and provide unprecedented access to massive repositories of information and media.

One characteristic of digital systems is their ability to represent and manipulate discrete elements of information. Any set that is restricted to a finite number of elements contains discrete information. Examples of discrete sets are the 10 decimal digits, the 26 letters of the alphabet, the 52 playing cards, and the 64 squares of a chessboard. Early digital computers were used for numeric computations. In this case, the discrete elements were the digits. From this application, the term *digital* computer emerged.

Discrete elements of information are represented in a digital system by physical quantities called *signals*. Electrical signals such as voltages and currents are the most common. Electronic devices called transistors predominate in the circuitry that implement, represent, and manipulate these signals. The signals in most present-day electronic digital systems use just two discrete values and are therefore said to be *binary*. A binary digit, called a *bit*, has two numerical values: 0 and 1. Discrete elements of information are represented with groups of bits called *binary codes*. For example, the decimal digits 0 through 9 are represented in a digital system with a code of four bits (e.g., the number 7 is represented by 0111). How a pattern of bits is interpreted as a number depends on the code system in which it resides. To make this distinction, we could write $(0111)_2$ to indicate that the pattern 0111 is to be interpreted in a binary system, and $(0111)_{10}$ to indicate that the reference system is decimal. Then $0111_2 = 7_{10}$, which is not the same as 0111_{10} , or one hundred eleven. The subscript indicating the base for interpreting a pattern of bits will be used only when clarification is needed. Through various techniques, groups of bits can be made to represent discrete symbols, not necessarily numbers, which are then used to develop the system in a digital format. Thus, a digital system is a system that manipulates discrete elements of information represented internally in binary form. In today's technology, binary systems are most practical because, as we will see, they can be implemented with electronic components.

Discrete quantities of information either emerge from the nature of the data being processed or may be quantized from a continuous process. On the one hand, a payroll schedule is an inherently discrete process that contains employee names, social security numbers, weekly salaries, income taxes, and so on. An employee's paycheck is processed by means of discrete data values such as letters of the alphabet (names), digits (salary), and special symbols (such as \$). On the other hand, a research scientist may observe a continuous process, e.g., temperature, but record only specific quantities in tabular form. The scientist is thus quantizing continuous data, making each number in the table a discrete quantity. In many cases, the quantization of a process can be performed automatically by an analog-to-digital converter, a device that forms a digital (discrete) representation of an analog (continuous) quantity. Digital cameras rely on this technology to quantify the measurements of exposure captured from an image.

The general-purpose digital computer is the best-known example of a digital system. The major parts of a computer are a memory unit, a central processing unit, and input–output units. The memory unit stores programs as well as input, output, and intermediate data. The central processing unit performs arithmetic and other data-processing operations as specified by the program. The program and data prepared by a user are transferred into memory by means of an input device such as a keyboard or a touch-screen video display. An output device, such as a printer, receives the results of the computations, and the printed results are presented to the user. A digital computer can accommodate many input and output devices. One very useful device is a communication unit that provides interaction with other users through the Internet. A digital computer is a powerful instrument that can perform not only arithmetic computations but also logical operations. In addition, it can be programmed to make decisions based on internal and external conditions.

There are fundamental reasons that commercial products are made with digital circuits. Like a digital computer, most digital devices are programmable. By changing the program in a programmable device, the same underlying hardware can be used for many different applications, thereby allowing its cost of development to be spread across sales to a wider customer base. Dramatic cost reductions in digital devices have come about because of advances in digital integrated circuit technology. As the number of transistors that can be put on a piece of silicon increases to produce complex functions, the cost per unit decreases, and digital devices can be bought at an increasingly reduced price. Equipment built with digital integrated circuits can perform at a speed of hundreds of millions of operations per second. Digital systems can be made to operate with extreme reliability by using error-correcting codes. An example of this strategy is the digital versatile disk (DVD), in which digital information representing photos, video, audio, and other data is recorded without the loss of a single item. Digital information on a DVD is recorded in such a way that, by examining the code in each digital sample before it is played back, any error can be automatically identified and corrected.

A digital system is an interconnection of digital modules. **To understand the operation of each digital module, it is necessary to have a basic knowledge of digital circuits and their logical function.** The first seven chapters of this book present the basic tools of digital design, such as logic gate structures, combinational and sequential circuits, and programmable logic devices. Chapter 8 introduces digital design at the register transfer level (RTL) using a modern, public-domain hardware description language (HDL). Chapter 9 concludes the text with laboratory exercises using digital circuits.

Today’s array of inexpensive digital devices is made possible by the convergence of fabrication technology and computer-based design methodology. Today’s “best practice” in digital design methodology uses HDLs to describe and simulate the functionality of a digital circuit. An HDL resembles a programming language and is suitable for describing digital circuits in textual form. It is used to simulate a digital system to verify its operation before hardware is built. It is also used in conjunction with logic synthesis tools to automate the design process. Because **it is important that students become familiar with an HDL-based design methodology**, HDL descriptions of digital circuits are presented throughout the book. While these examples help illustrate the features of an HDL, they

also demonstrate the best practices used by industry to exploit HDLs. Ignorance of these practices will lead to cute, but worthless, HDL models that may simulate a phenomenon, but that cannot be synthesized by design tools, or to models which waste silicon area or synthesize to hardware that does not operate correctly.

As previously stated, digital systems manipulate discrete quantities of information that are represented in binary form. Operands used for calculations may be expressed in the binary number system. Other discrete elements, including the decimal digits and characters of the alphabet, are represented in binary codes. Digital circuits, also referred to as *logic circuits*, process data by means of binary logic elements (logic gates) using binary signals. Quantities are stored in binary (two-valued) storage elements (flip-flops). The purpose of this chapter is to introduce the various binary concepts and provide a foundation for further study in the succeeding chapters.

1.2 BINARY NUMBERS

A decimal number such as 7392 represents a quantity equal to 7 thousands, plus 3 hundreds, plus 9 tens, plus 2 units. The thousands, hundreds, etc., are powers of 10 implied by the position of the coefficients (symbols) in the number. To be more exact, 7392 is a shorthand notation for what should be written as

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

However, the convention is to write only the numeric coefficients and, from their position, deduce the necessary powers of 10, with powers increasing from right to left. In general, a number with a decimal point is represented by a series of coefficients:

$$a_5 a_4 a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}$$

The coefficients a_j are any of the 10 digits (0, 1, 2, . . . , 9), and the subscript value j gives the place value and, hence, the power of 10 by which the coefficient must be multiplied. Thus, the preceding decimal number can be expressed as

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$$

with $a_3 = 7$, $a_2 = 3$, $a_1 = 9$, and $a_0 = 2$, and the other coefficients equal to zero.

The radix of a number system determines the number of distinct values that can be used to represent any arbitrary number. The decimal number system is said to be of *base*, or *radix*, 10 because it uses 10 digits and the coefficients are multiplied by powers of 10. The *binary* system is a different number system. The coefficients of the binary number system have only two possible values: 0 and 1. Each coefficient a_j is multiplied by a power of the radix, for example, 2^j , and the results are added to obtain the decimal equivalent of the number. The radix point (e.g., the decimal point when 10 is the radix) distinguishes positive powers of 10 from negative powers of 10. For example, the decimal equivalent of the binary number 11010.11 is 26.75, as shown from the multiplication of the coefficients by powers of 2:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

There are many different number systems. In general, a number expressed in a base- r system has coefficients multiplied by powers of r :

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} \\ + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

The coefficients a_j range in value from 0 to $r - 1$. To distinguish between numbers of different bases, we enclose the coefficients in parentheses and write a subscript equal to the base used (except sometimes for decimal numbers, where the content makes it obvious that the base is decimal). An example of a base-5 number is

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

The coefficient values for base 5 can be only 0, 1, 2, 3, and 4. The octal number system is a base-8 system that has eight digits: 0, 1, 2, 3, 4, 5, 6, 7. An example of an octal number is $(127.4)_8$. To determine its equivalent decimal value, we expand the number in a power series with a base of 8:

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

Note that the digits 8 and 9 cannot appear in an octal number.

It is customary to borrow the needed r digits for the coefficients from the decimal system when the base of the number is less than 10. The letters of the alphabet are used to supplement the 10 decimal digits when the base of the number is greater than 10. For example, in the *hexadecimal* (base-16) number system, the first 10 digits are borrowed from the decimal system. The letters A, B, C, D, E, and F are used for the digits 10, 11, 12, 13, 14, and 15, respectively. An example of a hexadecimal number is

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46,687)_{10}$$

The hexadecimal system is used commonly by designers to represent long strings of bits in the addresses, instructions, and data in digital systems. For example, B65F is used to represent 1011011001011111.

As noted before, the digits in a binary number are called *bits*. When a bit is equal to 0, it does not contribute to the sum during the conversion. Therefore, the conversion from binary to decimal can be obtained by adding only the numbers with powers of two corresponding to the bits that are equal to 1. For example,

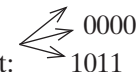
$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

There are four 1's in the binary number. The corresponding decimal number is the sum of the four powers of two. Zero and the first 24 numbers obtained from 2 to the power of n are listed in Table 1.1. In computer work, 2^{10} is referred to as K (kilo), 2^{20} as M (mega), 2^{30} as G (giga), and 2^{40} as T (tera). Thus, $4K = 2^{12} = 4,096$ and $16M = 2^{24} = 16,777,216$. Computer memory capacity and word size are usually given in bytes. A *byte* is equal to eight bits and can accommodate (i.e., represent the code of) one keyboard character. A computer hard disk with four gigabytes of storage has a capacity of $4G = 2^{32}$ bytes (approximately 4 billion bytes). A terabyte is 1024 gigabytes, approximately 1 trillion bytes.

Table 1.1
Powers of Two

<i>n</i>	2^n	<i>n</i>	2^n	<i>n</i>	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

Arithmetic operations with numbers in base *r* follow the same rules as for decimal numbers. When a base other than the familiar base 10 is used, one must be careful to use only the *r*-allowable digits. Examples of addition, subtraction, and multiplication of two binary numbers are as follows:

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	<u>+100111</u>	subtrahend:	<u>-100111</u>	multiplier:	<u>× 101</u>
sum:	1010100	difference:	000110		1011
				partial product:	
				product:	110111

The sum of two binary numbers is calculated by the same rules as in decimal, except that the digits of the sum in any significant position can be only 0 or 1. Any carry obtained in a given significant position is used by the pair of digits one significant position higher. Subtraction is slightly more complicated. The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.) Multiplication is simple: The multiplier digits are always 1 or 0; therefore, the partial products are equal either to a shifted (left) copy of the multiplicand or to 0.

Practice Exercise 1.1

What is the decimal value of $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$?

Answer: 21

1.3 NUMBER-BASE CONVERSIONS

Representations of a number in a different radix are said to be equivalent if they have the same decimal representation. For example, $(0011)_8$ and $(1001)_2$ are equivalent—both

have decimal value 9. The conversion of a number in base r to decimal is done by expanding the number in a power series and adding all the terms as shown previously. We now present a general procedure for the reverse operation of converting a decimal number to a number in base r . If the number includes a radix point, it is necessary to separate the number into an integer part and a fraction part, since each part must be converted differently. The conversion of a decimal integer to a number in base r is done by *dividing the number and all successive quotients by r and accumulating the remainders*. This procedure is best illustrated by example.

EXAMPLE 1.1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = answer

Conversion from decimal integers to any base- r system is similar to this example, except that division is done by r instead of 2.

EXAMPLE 1.2

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently tabulated as follows:

153		
19		1
2		3
0		$2 = (231)_8$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers instead of remainders are accumulated. Again, the method is best explained by example.

EXAMPLE 1.3

Convert $(0.6875)_{10}$ to binary. First, 0.6875 is multiplied by 2 to give an integer and a fraction. Then the new fraction is multiplied by 2 to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficients of the binary number are obtained from the integers as follows:

	Integer		Fraction	Coefficient
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

Therefore, the answer is $(0.6875)_{10} = (0.a_{-1} a_{-2} a_{-3} a_{-4})_2 = (0.1011)_2$.

To convert a decimal fraction to a number expressed in base r , a similar procedure is used. However, multiplication is by r instead of 2, and the coefficients found from the integers may range in value from 0 to $r - 1$ instead of 0 and 1.

EXAMPLE 1.4

Convert $(0.513)_{10}$ to octal.

$$0.513 \times 8 = 4.104$$

$$0.104 \times 8 = 0.832$$

$$0.832 \times 8 = 6.656$$

$$0.656 \times 8 = 5.248$$

$$0.248 \times 8 = 1.984$$

$$0.984 \times 8 = 7.872$$

The answer, to six significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517 \dots)_8$$

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers. Using the results of Examples 1.1 and 1.3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

From Examples 1.2 and 1.4, we have

$$(153.513)_{10} = (231.406517)_8$$

**Practice Exercise 1.2**

Convert $(117.23)_{10}$ to octal.

Answer: $(117.23)_{10} = (165.1656)_8$

1.4 OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important role in digital computers, because shorter patterns of hex characters are easier to recognize than long patterns of 1's and 0's. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The first 16 numbers in the decimal, binary, octal, and hexadecimal number systems are listed in Table 1.2.

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

to the left and to the right. The corresponding octal digit is then assigned to each group. The following example illustrates the procedure:

$$\begin{array}{cccccccccc} (10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110)_2 & = & (26153.7406)_8 \\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 & & \end{array}$$

Conversion from binary to hexadecimal is similar, except that the binary number is divided into groups of *four* digits:

$$\begin{array}{cccccccc} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 & = & (2C6B.F2)_{16} \\ 2 & C & 6 & B & & F & 2 & & \end{array}$$

The corresponding hexadecimal (or octal) digit for each group of binary digits is easily remembered from the values listed in Table 1.2.

Conversion from octal or hexadecimal to binary is done by reversing the preceding procedure. Each octal digit is converted to its three-digit binary equivalent. Similarly, each hexadecimal digit is converted to its four-digit binary equivalent. The procedure is illustrated in the following examples:

$$\begin{array}{cccccccc} (673.124)_8 & = & (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2 \\ & & 6 & 7 & 3 & & 1 & 2 & 4 \end{array}$$

and

$$\begin{array}{cccccccc} (306.D)_{16} & = & (0011 & 0000 & 0110 & \cdot & 1101)_2 \\ & & 3 & 0 & 6 & & D \end{array}$$

Binary numbers are difficult to work with because they require three or four times as many digits as their decimal equivalents. For example, the binary number 111111111111 is equivalent to decimal 4095. However, digital computers use binary representation of numbers, and it is sometimes necessary for the human operator or user to communicate directly with the machine by means of such numbers. One scheme that retains the binary system in the computer, but reduces the number of digits the human must consider,¹ utilizes the relationship between the binary number system and the octal or hexadecimal system. By this method, the human thinks in terms of octal or hexadecimal numbers and performs the required conversion by inspection when direct communication with the machine is necessary. Thus, the binary number 111111111111 has 12 digits and is expressed in octal as 7777 (4 digits) or in hexadecimal as FFF (3 digits). During communication between people (about binary numbers in the computer), the octal or hexadecimal representation is more desirable because it can be expressed more compactly with a third or a quarter of the number of digits required for the equivalent binary number. Thus, **most computer manuals use either octal or hexadecimal numbers to specify instructions and other binary quantities.** The choice between them is arbitrary, although hexadecimal tends to win out, since it can represent a byte with two digits.

Practice Exercise 1.3

Find the binary representation of 135_{10} .

Answer: $135_{10} = 1110\ 0001_2$

Practice Exercise 1.4

Find the octal representation of $(135)_{10}$.

Answer: $135_{10} = 702_8$

1.5 COMPLEMENTS OF NUMBERS

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations. There are two types of complements for each base- r system: the *radix complement* and the *diminished radix complement*. The first is referred to as the r 's complement and the second as the $(r - 1)$'s complement. When the value of the base r is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

¹ Machines having a word length of 64 bits are common.

Diminished Radix Complement

Given a number N in base r having n digits, the $(r - 1)$'s complement of N , that is, its diminished radix complement, is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$. In this case, 10^n represents a number that consists of a single 1 followed by n 0's. $10^n - 1$ is a number represented by n 9's. For example, if $n = 4$, we have $10^4 = 10,000$ and $10^4 - 1 = 9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. Here are some numerical examples:

The 9's complement of 546700 is $999999 - 546700 = 453299$.

The 9's complement of 012398 is $999999 - 012398 = 987601$.

For binary numbers, $r = 2$ and $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's. $2^n - 1$ is a binary number represented by n 1's. For example, if $n = 4$, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can have either $1 - 0 = 1$ or $1 - 1 = 0$, which causes the bit to change from 0 to 1 or from 1 to 0, respectively. Therefore, **the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's**. The following are some numerical examples:

The 1's complement of 1011000 is 0100111.

The 1's complement of 0101101 is 1010010.

The $(r - 1)$'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

Radix Complement

The r 's complement of an n -digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r 's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 2389 is $7610 + 1 = 7611$ and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's-complement value.

Since 10 is a number represented by a 1 followed by n 0's, $10^n - N$, which is the 10's complement of N , can be formed also by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9. Thus,

the 10's complement of 012398 is 987602

and

the 10's complement of 246700 is 753300

The 10's complement of the first number (012398) is obtained by subtracting 8 from 10 in the least significant position and subtracting all other digits from 9. The 10's complement of the second number (246700) is obtained by leaving the two least significant 0's unchanged, subtracting 7 from 10, and subtracting the other three digits from 9.

Practice Exercise 1.5

Find (a) the diminished radix (9's) complement and (b) the radix (10's) complement of 135_{10} .

Answer:

(a) 9's complement: 864_{10}

(b) 10's complement: 865_{10}

Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged and replacing 1's with 0's and 0's with 1's in all other higher significant digits. For example,

the 2's complement of 1101100 is 0010100

and

the 2's complement of 0110111 is 1001001

The 2's complement of the first number is obtained by leaving the two least significant 0's and the first 1 unchanged and then replacing 1's with 0's and 0's with 1's in the other four most significant digits. The 2's complement of the second number is obtained by leaving the least significant 1 unchanged and complementing all other digits.

In the previous definitions, it was assumed that the numbers did not have a radix point. If the original number N contains a radix point, the point should be removed temporarily in order to form the r 's or $(r - 1)$'s complement. The radix point is then restored to the complemented number in the same relative position. It is also worth mentioning that **the complement of the complement restores the number to its original value**. To see this relationship, note that the r 's complement of N is $r^n - N$, so that the complement of the complement is $r^n - (r^n - N) = N$ and is equal to the original number.

Subtraction with Complements

The direct method of subtraction taught in elementary schools uses the borrow concept. In this method, we borrow a 1 from a higher significant position when the minuend digit is smaller than the subtrahend digit. The method works well when people perform subtraction with paper and pencil. However, when subtraction is implemented with digital hardware, the method is less efficient than the method that uses complements.

The subtraction of two n -digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r 's complement of the subtrahend N . Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n , which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r 's complement of $(N - M)$. To obtain the answer in a familiar form, take the r 's complement of the sum and place a negative sign in front.

The following examples illustrate the procedure:

EXAMPLE 1.5

Using 10's complement, subtract $72532 - 3250$.

$$\begin{array}{r}
 M = \quad 72532 \\
 10\text{'s complement of } N = + \underline{96750} \\
 \text{Sum} = \quad 169282 \\
 \text{Discard end carry } 10^5 = -\underline{100000} \\
 \text{Answer} = \quad 69282
 \end{array}$$

Note that M has five digits and N has only four digits. Both numbers must have the same number of digits, so we write N as 03250. Taking the 10's complement of N produces a 9 in the most significant position. The occurrence of the end carry signifies that $M \geq N$ and that the result is therefore positive. ■

EXAMPLE 1.6

Using 10's complement, subtract $3250 - 72532$.

$$\begin{array}{r}
 M = \quad 03250 \\
 10\text{'s complement of } N = +27468 \\
 \text{Sum} = \quad 30718
 \end{array}$$

There is no end carry. Therefore, the answer is written with a minus sign as $-(10\text{'s complement of } 30718) = -69282$.

Note that since $3250 < 72532$, the result is negative. Because we are dealing with unsigned numbers, there is really no way to get an unsigned result for this case. When subtracting with complements, we recognize the negative answer from the absence of the end carry and the complemented result. When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with complements is done with binary numbers in a similar manner, using the procedure outlined previously. ■