
# Digital Systems

## *Principles and Applications*

TWELFTH EDITION

Neal S. Widmer • Gregory L. Moss • Ronald J. Tocci

Pearson

TWELFTH EDITION

GLOBAL EDITION

# Digital Systems

## Principles and Applications


**Neal S. Widmer**
Purdue University

**Gregory L. Moss**
Purdue University

**Ronald J. Tocci**
Monroe Community College



Pearson

Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Dubai • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • Sao Paulo • Mexico City • Madrid • Amsterdam • Munich • Paris • Milan

**Editor-in-Chief:** Andrew Gilfillan
**Product Manager:** Anthony Webster
**Program Manager:** Holly Shufeldt
**Project Manager:** Rex Davidson
**Editorial Assistant:** Nancy Kesterson
**Team Lead Project Manager:** Bryan Pirrmann
**Team Lead Program Manager:** Laura Weaver
**Project Manager, Global Edition:** Sudipto Roy
**Senior Acquisitions Editor, Global Edition:** Sandhya Ghoshal
**Senior Project Editor, Global Edition:** Daniel Luiz
**Project Editor, Global Edition:** Rahul Arora
**Manager, Media Production, Global Edition:** M. Vikram Kumar
**Manufacturing Controller, Production, Global Edition:** Angela Hawksbee
**Director of Marketing:** David Gesell
**Senior Product Marketing Manager:** Darcy Betts
**Field Marketing Manager:** Thomas Hayward
**Procurement Specialist:** Deidra M. Skahill
**Creative Director:** Andrea Nix
**Art Director:** Diane Y. Ernsberger
**Cover Designer:** Lumina Datamatics, Inc.
**Full-Service Project Management:** Philip Alexander/Integra Software Services, Pvt, Ltd.

**Pearson Education Limited**
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

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



*Authorized adaptation from the United States edition, entitled Digital Systems, 12th edition, ISBN 978-0-134-22013-0, by Ronald Tocci, Neal Widmer, and Greg Moss, published by Pearson Education © 2017.*



ISBN 10: 129-2-16200-7
ISBN 13: 978-1-292-16200-3

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
14 13 12 11 10

Printed and bound in Vivar, Malaysia.
Typeset in Times Europa LT Std Roman by Integra Software Services, Pvt, Ltd.

# PREFACE

This book is a comprehensive study of the principles and techniques of modern digital systems. It teaches the fundamental principles of digital systems and covers thoroughly both traditional and modern methods of applying digital design and development techniques, including how to manage a systems-level project. The book is intended for use in two- and four-year programs in technology, engineering, and computer science. It can also be used for High School STEM education courses in these topical areas. Although a background in basic electronics is helpful, most of the material requires no electronics training. Portions of the text that use electronics concepts can be skipped without adversely affecting the comprehension of the logic principles.

## What's New in This Edition?

The following list summarizes the improvements in the twelfth edition of *Digital Systems*. Details can be found in the section titled "Specific Changes" on page 6.

- Every *section* of every chapter now has a short list of expected outcomes for that section.
- Chapter 1 has been revised extensively in response to feedback from users.
- New material on troubleshooting prototype circuits using systematic fault isolation techniques applied to digital logic circuits has been added to Section 4-13.
- Quadrature Shaft Encoders used to obtain absolute shaft position serve as a real example of flip-flop applications, and timing limitations.
- More material has been added to better explain the behavior of VHDL data objects and how they are updated in sequential processes.
- Throughout the text, obsolete technology has been deleted or abbreviated to provide only content appropriate to modern systems. More modern examples are used as needed.
- Some new problems have been added and outdated problems have been removed.

## General Features

In industry today, getting a product to market very quickly is important. The use of modern design tools, CPLDs, and FPGAs allows engineers to progress from concept to functional silicon very quickly. Microcontrollers have taken over many applications that once were implemented by digital circuits, and DSP has been used to replace many analog circuits. It is amazing that microcontrollers, DSP, and all the necessary glue logic can now be consolidated onto a single FPGA using a hardware description language with advanced development tools. Today's students must be exposed to these modern tools, even in an introductory course. It is every educator's responsibility to find the best way to prepare graduates for the work they will encounter in their professional lives.

The standard SSI and MSI parts that have served as "bricks and mortar" in the building of digital systems for over 40 years are now obsolete and becoming less available. Many of the techniques that have been taught over that time have focused on optimizing circuits that are built from these outmoded devices. The topics that are uniquely suited to applying the old technology *but do not contribute to an understanding of the new technology* are being de-emphasized. From an educational standpoint, however, these small ICs do offer a way to study simple digital circuits, and the wiring of circuits using breadboards is a valuable pedagogic exercise. They help to solidify concepts such as binary inputs and outputs, physical device operation, and practical limitations, using a very simple platform. Consequently, we have chosen to continue to introduce the conceptual descriptions of digital circuits and to offer examples using conventional standard logic parts. For instructors who continue to teach the fundamentals using SSI and MSI circuits, this edition retains those qualities that have made the text so widely accepted in the past. Many hardware design tools even provide an easy-to-use design entry technique that will employ the functionality of conventional standard parts with the flexibility of programmable logic devices. A digital design can be described using a schematic drawing with pre-created building blocks that are equivalent to conventional standard parts, which can be compiled and then programmed directly into a target PLD with the added capability of easily simulating the design within the same development tool.

We believe that graduates will actually apply the concepts presented in this book using higher-level description methods and more complex programmable devices. The major shift in the field is a greater need to understand the description methods, rather than focusing on the architecture of an actual device. Software tools have evolved to the point where there is little need for concern about the inner workings of the hardware but much more need to focus on what goes in, what comes out, and how the designer can describe what the device is supposed to do. We also believe that graduates will be involved with projects using state-of-the-art design tools and hardware solutions.

This book offers a strategic advantage for teaching the vital topic of hardware description languages to beginners in the digital field. VHDL is undisputedly an industry standard language at this time, but it is also very complex and has a steep learning curve. Beginning students are often discouraged by the rigorous requirements of various data types, and they struggle with understanding edge-triggered events in VHDL. Fortunately, Altera offers AHDL, a less demanding language that uses the same basic concepts as VHDL but is much easier for beginners to master. So, instructors can opt to use AHDL to teach introductory students or VHDL for more advanced classes. This edition offers more than 40 AHDL examples, more than 40 VHDL examples, and many examples of simulation testing. All of these design files are available on the website (http://www.pearsonglobaleditions.com/tocci).

Altera's software development system is Quartus II. The material in this text does not attempt to teach a particular hardware platform or the details of using a software development system. We have chosen to show what this tool can do, rather than train the reader how to use it.

Many laboratory hardware options are available to users of this book. Complete development boards are available that offer the normal types of inputs and outputs like logic switches, pushbuttons, clock signals, LEDs, and 7-segment displays. Many boards also offer standard connectors for readily available computer hardware, such as a standard keyboard, computer mouse, VGA video monitor, COM ports, audio in/out jacks, plus two 40-pin general-purpose I/O ribbon connectors that allow connection to any digital peripheral hardware.

Our approach to HDL and PLDs gives instructors several options:

1. The HDL material can be skipped entirely without affecting the continuity of the text.
2. HDL can be taught as a separate topic by skipping the material initially and then going back to the last sections of Chapters 3, 4, 5, 6, 7, and 9 and then covering Chapter 10.
3. HDL and the use of PLDs can be covered as the course unfolds—chapter by chapter—and woven into the fabric of the lecture/lab experience.

Among all specific hardware description languages, VHDL is clearly the industry standard and is most likely to be used by graduates in their careers. We have always felt that it is a bold proposition, however, to try to teach VHDL in an introductory course. The nature of the syntax, the subtle distinctions in object types, and the higher levels of abstraction can pose obstacles for a beginner. For this reason, we have included Altera's AHDL as the recommended introductory language for freshman and sophomore courses. We have also included VHDL as the recommended language for more advanced classes or introductory courses offered to more mature students. We do not recommend trying to cover both languages in the same course. Sections of the text that cover the specifics of a language are clearly designated with a color bar in the margin. The HDL code figures are set in a color to match the color-coded text explanation. The reader can focus only on the language of his or her choice and skip the other. Obviously, we have attempted to appeal to the diverse interests of our market, but we believe we have created a book that can be used in multiple courses and will serve as an excellent reference after graduation.

## Chapter Organization

Many instructors opt to not use the chapters of a textbook in the sequence in which they are presented. This book was written so that, for the most part, each chapter builds on previous material, but it is possible to alter the chapter sequence somewhat. The first part of Chapter 6 (arithmetic operations) can be covered right after Chapter 2 (number systems), although this will lead to a long interval before the arithmetic circuits of Chapter 6 are encountered. Much of the material in Chapter 8 (IC characteristics) can be covered earlier (e.g., after Chapter 4 or 5) without creating any serious problems.

This book can be used either in a one-term course or in a two-term sequence. In a one-term course, limits on available class hours might require omitting some topics. Obviously, the choice of deletions will depend on factors such as program or course objectives and student background. Sections

**FIGURE P1** Letters denote categories of problems, and asterisks indicate that corresponding solutions are provided at the end of the text.

## PROBLEMS

### SECTION 9-1

**B** 9-1. Refer to Figure 9-3. Determine the levels at each decoder output for the following sets of input conditions.

(a)*All inputs LOW

(b)*All inputs LOW except $E_3$ = HIGH

(c) All inputs HIGH except $\overline{E}_1 = \overline{E}_2$ = LOW

(d) All inputs HIGH

**B** 9-2.* What is the number of inputs and outputs of a decoder that accepts 128 different input combinations?

* Answers to problems marked with an asterisk can be found in the back of the text.

in each chapter that deal with troubleshooting, PLDs, HDLs, or microcomputer applications can be deferred to an advanced course.

**PROBLEM SETS** This edition includes six categories of problems: basic (B), challenging (C), troubleshooting (T), new (N), design (D), and HDL (H). Undesignated problems are considered to be of intermediate difficulty, between basic and challenging. Problems for which solutions are printed in the back of the text or on the website (http://www.pearsonglobaleditions.com/tocci) are marked with an asterisk (see Figure P1).

**PROJECT MANAGEMENT AND SYSTEM-LEVEL DESIGN** Several real-world examples are included in Chapter 10 to describe the techniques used to manage projects. These applications are generally familiar to most students studying electronics, and the primary example of a digital clock is familiar to everyone. Many texts talk about top-down design, but this text demonstrates the key features of this approach and how to use the modern tools to accomplish it.

**SIMULATION FILES** This edition also includes simulation files that can be loaded into Multisim®. The circuit schematics of many of the figures throughout the text have been captured as input files for this popular simulation tool. Each file has some way of demonstrating the operation of the circuit or reinforcing a concept. In many cases, instruments are attached to the circuit and input sequences are applied to demonstrate the concept presented in one of the figures of the text. These circuits can then be modified as desired to expand on topics or create assignments and tutorials for students. All figures in the text that have a corresponding simulation file on the website are identified by the icon shown in Figure P2.

## Specific Changes

The major changes in the topical coverage are listed here.

- **Chapter 1.** Chapter 1 has been revised extensively in response to feedback from users. The significance of how Digital Systems will impact innovations of the future is emphasized.

  New material focuses on interpretation of terminology and introduction to concepts used throughout the text. Basic concepts of binary

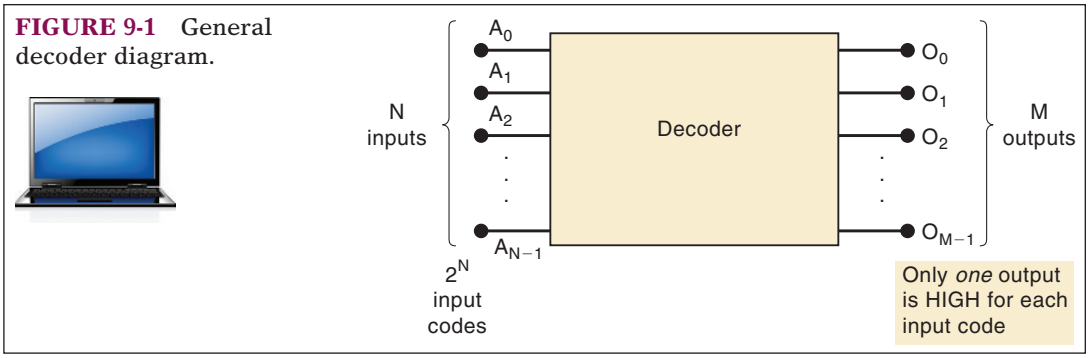



**FIGURE P2** The icon denotes a corresponding simulation file on the Web.

signals are introduced and explained through examples. New material on periodic cycles and measurements on digital waveforms is presented, setting the stage for understanding these issues in later chapters. The basics of digital signals and sampling are explained at a very introductory level.

This chapter in the 11th edition had material that has now become very outdated since its publication. Some of the historic analogies used in that edition were ineffective. The revisions have replaced or eliminated these.

- **Chapter 2.** The Gray Code is used to introduce the concept of a quadrature encoder: a device that produces a 2-bit Gray Code sequence capable of discerning the direction and angular rotation of a shaft.
- **Chapter 3.** New problems at the end of this chapter focus on logic circuits common to automobiles.
- **Chapter 4.** The material introducing PLD programming and development software has been updated and improved. The section on troubleshooting has been expanded to teach structured problem solving as it applies to hardware debugging of traditional prototyped digital circuits. The VHDL material has been enhanced to explain some subtle but very important aspects of data objects in this language. The role of the "PROCESS" is also more thoroughly covered improving the foundation that Chapter 5 builds on.
- **Chapter 5.** High-speed digital systems are easily affected by timing limitations of the circuitry. New material in this chapter explains the adverse effects caused when setup and hold time requirements are violated by explaining meta-stability. A teaching example that can be reproduced in the laboratory environment has been added. The focus is on the many applications of D flip-flops but it is presented in the context of a quadrature shaft encoder that must reliably and repeatedly keep track of absolute shaft position as it is rotated back and forth over many cycles. Design techniques from Chapter 4 are employed to design a circuit that should meet the system's needs. The initial circuit's marginal performance demonstrates what happens when real-timing constraints are not taken into account. A way to correct this problem is presented using even more applications of D flip-flops.
- **Chapter 6.** An Example from the 11th edition used some features of Quartus software that have since become obsolete. The example has been modified to align with more recent updates of Quartus.
- **Chapter 7.** Very few and minor changes were made to Chapter 7.
- **Chapter 8.** The section on the obsolete Emitter Coupled Logic (ECL) was deleted along with other minor updates.

- **Chapter 9.** The concept of Time Division Multiplexing is added to provide an example of how many digital signals are able to share a common data pathway. A simple system is presented that can easily be reproduced in a laboratory exercise.
- **Chapter 10.** No changes were made in Chapter 10.
- **Chapter 11.** No changes were made in Chapter 11.
- **Chapter 12.** The coverage of floating gate MOSFETS, the technology behind flash memory, is enhanced.
- **Chapter 13.** This chapter has been generalized with references to older series of CPLDs and FPGAs abbreviated.

## Retained Features

This edition retains all of the features that made the previous editions so widely accepted. It utilizes a block diagram approach to teach the basic logic operations without confusing the reader with the details of internal operation. All but the most basic electrical characteristics of the logic ICs are withheld until the reader has a firm understanding of logic principles. In Chapter 8, the reader is introduced to the internal IC circuitry. At that point, the reader can interpret a logic block's input and output characteristics and "fit" it properly into a complete system.

The treatment of each new topic or device typically follows these steps: the principle of operation is introduced; thoroughly explained examples and applications are presented, often using actual ICs; short review questions are posed at the end of the section; and finally, in-depth problems are available at the end of the chapter. These problems, ranging from simple to complex, provide instructors with a wide choice of student assignments. These problems are often intended to reinforce the material without simply repeating the principles. They require students to demonstrate comprehension of the principles by applying them to different situations. This approach also helps students to develop confidence and expand their knowledge of the material.

The material on PLDs and HDLs is distributed throughout the text, with examples that emphasize key features in each application. These topics appear at the end of each chapter, making it easy to relate each topic to the general discussion earlier in the chapter or to address the general discussion separately from the PLD/HDL coverage.

The extensive troubleshooting coverage is spread over Chapters 4 through 12 and includes presentation of troubleshooting principles and techniques, case studies, 17 troubleshooting examples, and 46 *real* troubleshooting problems. When supplemented with hands-on lab exercises, this material can help foster the development of good troubleshooting skills.

This edition offers more than 220 worked-out examples, more than 660 review questions, and more than 640 chapter problems/exercises. Some of these problems are applications that show how the logic devices presented in the chapter are used in a typical microcomputer system. Answers to a majority of the problems immediately follow the Glossary. The Glossary provides concise definitions of all terms in the text that have been highlighted in boldface type.

An IC index is provided at the back of the book to help readers locate easily material on any IC cited or used in the text. The back endsheets provide tables of the most often used Boolean algebra theorems, logic gate summaries, and flip-flop truth tables for quick reference when doing problems or working in the lab.

## Supplements

An extensive complement of teaching and learning tools has been developed to accompany this textbook. Each component provides a unique function, and each can be used independently or in conjunction with the others.

### WEB RESOURCES

- **Quartus II Web Version software from Altera.** This development system software is available from Altera.
- **Design files from the textbook figures.** More than 40 design files in each language are presented in figures throughout the text. Students can load these into the Altera software and test them.
- **Circuits from the text rendered in Multisim®.** Students can open and work interactively with approximately 100 circuits to increase their understanding of concepts and prepare for laboratory activities. The Multisim circuit files are provided for use by anyone who has Multisim software.

### INSTRUCTOR RESOURCES

- ***Online Instructor's Resource Manual.*** This manual contains worked-out solutions for all end-of-chapter problems in this textbook.
- ***Online PowerPoint® presentations.*** Figures from the text, in addition to Lecture Notes for each chapter, are available.
- ***Online TestGen.*** A computerized test bank is available.

To access supplementary materials online, instructors need to request an instructor access code. Go to **www.pearsonglobaleditions.com/tocci**, where you can register for an instructor access code. Within 48 hours after registering, you will receive a confirming e-mail, including an instructor access code. Once you have received your code, go to the site and log on for full instructions on downloading the materials you wish to use.

## Acknowledgments

We are grateful to all those who evaluated the eleventh edition and provided answers to an extensive questionnaire:

Their comments, critiques, and suggestions were given serious consideration and were invaluable in determining the final form of the twelfth edition.

We also are greatly indebted to Professor Frank Ambrosio, Monroe Community College, for his usual high-quality work on the *Instructor's*

*Resource Manual*; and Professor Daniel Leon-Salas, Purdue University, for his technical review of topics and many suggestions for improvements.

A writing project of this magnitude requires conscientious and professional editorial support, and Pearson came through again in typical fashion. We thank the staff at Pearson for their help to make this publication a success.

And finally, we want to let our wives, children, and grandchildren know how much we appreciate their support and their understanding. We hope that we can eventually make up for all the hours we spent away from them while we worked on this revision.

Neal S. Widmer
Ronald J. Tocci
Gregory L. Moss

## Acknowledgments for the Global Edition

Pearson would like to thank the following people for their work on the content of the Global Edition:

**Contributors:**

Moumita Mitra Manna, University of Calcutta

Ankita Pramanik, Indian Institute of Engineering Science and Technology, Shibpur

**Reviewers:**

Chih-Wei Liu, National Chiao Tung University

Hung-Ming Chen, National Chiao Tung University

Ankita Pramanik, Indian Institute of Engineering Science and Technology, Shibpur

# CONTENTS

*To you, Cap, for loving me for so long; and for the million and one ways you brighten the lives of everyone you touch.*

*—RJT*

*To my wife and best friend, Kris, who has sacrificed the most to complete this work. To our children John and Brooke, Brad and Amber, Blake and Tashi, Matt and Tamara, Katie and Matthew, and to our grandchildren Jersey, Judah, and the two we have yet to meet, who are in production along with this book.*

*—NSW*

*To my expanding family, Marita, David, Ryan, Christy, Jeannie, Taylor, Micah, Brayden, and Lorelei.*

*—GLM*

CHAPTER 1

# INTRODUCTORY CONCEPTS

## OUTLINE

## CHAPTER OUTCOMES

*Upon completion of this chapter, you will be able to:*

- Distinguish between analog and digital representations.
- Describe how information can be represented using just two states (1s and 0s).
- Cite the advantages and drawbacks of digital techniques compared with analog.
- Describe the purpose of analog-to-digital converters (ADCs) and digital-to-analog converters (DACs).
- Recognize the basic characteristics of the binary number system.
- Convert a binary number to its decimal equivalent.
- Count in the binary number system.
- Identify typical digital signals.
- Identify a timing diagram.
- State the differences between parallel and serial transmission.
- Describe the property of memory.
- Describe the major parts of a digital computer and understand their functions.
- Distinguish among microcomputers, microprocessors, and microcontrollers.

## INTRODUCTION

In today's world, the term *digital* has become part of our everyday vocabulary because of the dramatic way that digital circuits and digital techniques have become so widely used in almost all areas of life: computers, automation, robots, medical science and technology, transportation, telecommunications, entertainment, space exploration, and on and on. You are about to begin an exciting educational journey in which you will discover the fundamental principles, concepts, and operations that are common to all digital systems, from the simplest on/off switch to the most complex computer.

This chapter will introduce many of the underlying concepts that you will encounter as you learn more about your digital world. As new terms and concepts are presented, you will be directed to the chapters later in the text that expand and clarify the points. We want you to realize just how deeply digital systems impact your life. Then we want you to wonder how they work and how you might use digital systems to make the future better.

Let's go through a typical example of starting a day. The alarm clock wakes me up and I look at the time of day displayed on big bright

seven-segment LEDs (see Chapter 9). The digital alarm has compared the time of day with my alarm setting and when they were equal it activated the alarm (see Chapter 10). The alarm is "latched" on until I reset it with "off" or "snooze"(see Chapter 5 for latches). I go to the bathroom and decide to weigh myself before showering. The bathroom scales respond to the tap of my toe by awaking from its sleep mode, clearing the digital display and waiting for me to step on. It measures my weight and displays it in pounds. After a few seconds, it goes back to sleep. I grab my cordless shaver from the charger. A digital circuit inside the shaver has been controlling the charging cycle. I pick up my electric toothbrush. It can operate in three modes or "states" depending on how many times I push the button (see state machines in Chapter 7). It also keeps track of how long I brush and signals every 30 seconds in a 2 minute brush cycle. This is all controlled by a digital system inside the toothbrush hand-piece. I flip on the closet light. It has an energy saver feature that turns it off in case I forget, thanks to a small digital circuit in the light bulb (see interfacing in Chapter 8). I walk into my bedroom and turn the lights on low using the dimmer switch. The dimmer switch is an old analog circuit, but the new LED light bulbs can still be dimmed by it! This is because of a digital circuit inside the LED light bulb that controls the LEDs (see pulse width modulation in Chapter 11). I disconnect my cell phone from its charger. What a digital miracle I am holding in my hand!

I have not left the bedroom and already my life has been touched by seven digital systems. We could continue but you get the idea. Digital systems are everywhere around you and new applications are constantly being developed. All of the digital systems in the world are built from a surprisingly small number of basic circuits or building blocks. There are many instances of each block in most systems but only a few different blocks. This book will introduce you to those basic digital circuits and help you to understand the purpose, role, capabilities, and limitations of each one. Then you can use your innovation skills and the knowledge from this book to meet the next new demand.

## 1-1 INTRODUCTION TO DIGITAL 1s AND 0s

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Correlate new terms with their definition.
- Identify two states and assign a digit to each.
- Correlate each state with its representation in a given circuit.
- Recognize which state will activate a device in a given system.
- Identify the state of a digital signal under various physical conditions.
- Assign proper names to signals in a digital system.

Digital systems deal with things that are in one of two distinct states. The easiest example is anything that is either on or off. If you look at many devices today, you will find that the on/off switch is a single push button with the symbol shown in **Figure 1-1**. This icon represents a 1 and a 0, the numerical digits used to describe the two states in a digital system. We use numeric digits 0 and 1 to represent the two states off and on, respectively. Since there are only two digits, we call them **binary digits,** or **bits.** It is often said that digital systems are just a bunch of 1s and 0s and that is pretty

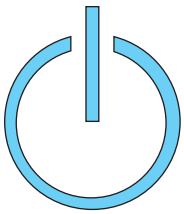


**FIGURE 1-1** The ubiquitous on/off symbol.

accurate. When we organize groups of numeric digits, we can create number systems and number systems are very powerful ways to represent things. As can be seen from all the digital systems around us, a lot can be done with just two possible states when circuits that can represent these two states are strategically organized.

Let's try to identify some things that must be categorized in one of two states in a system that is familiar to everyone: the automobile. The doors are either locked or unlocked. There is no such thing as being partially locked. We could also say a door is either open or closed. Now we know that a door can be partially opened, but in an automotive system the important thing to know is when the door is completely closed and safely latched. One state is considered to be closed and latched, while the other state is anything from slightly ajar to wide open. The parking brake is either set (engaged to any degree) or it is not set (completely disengaged). The engine is either running (at any speed) or it is not running. A button on the trunk lid is either pressed or not pressed. On some cars, opening the trunk when the engine is running requires the parking break to be set, the doors unlocked, and the trunk button to be pressed. When the engine is not running the trunk can be opened whenever the trunk button is pressed and the doors are unlocked. Digital circuits observe the state of each component and make a "logical" decision to either open or not open the trunk. For this reason, these conditions are often referred to as **logic states**.

After the two states of a system component are defined, one of the digital values (1 or 0) is assigned to each state. For example, on a Ford perhaps a door that is open may be assigned a 1 (closed = 0), but on a Lexus a door that is open may be assigned a state of 0 (closed = 1). In Chapter 3, we will discuss naming conventions for digital signals that help avoid confusion regarding the meaning of 1s and 0s in any system.

How are the states of 1 and 0 represented electrically in a digital system? The answer depends on the technology of the electrical system but the simplest answer is that a 0 is generally represented by a low voltage (close to 0 V) and a 1 is generally represented by a higher voltage. Consider, as an example, common electrical circuits in a home and in an automobile. In electrical systems, a voltage must be applied to a complete circuit to cause current to flow through the active device and "turn it on." **Figure 1-2(a)** demonstrates a light bulb in your home which requires 110 V AC (alternating current) to turn the light on. When no voltage is applied (0 volts AC), the light is off. Any light bulb in your car requires 12 V DC (direct current) to turn the light on and 0 V DC to turn it off, as demonstrated in Figure 1-2(b). The two systems are very similar but the technology of the systems differs. Consequently, the representations of a HIGH state (i.e., higher voltage) must match the system. In these simple wiring examples, the HIGH voltage is either connected to or disconnected from the lamp. A more accurate model of a digital logic circuit reflects that the output is always connected to either the source of the high voltage (HIGH state) or the source of the low voltage (LOW state). Figures 1-2(c) and (d) illustrate how this would look

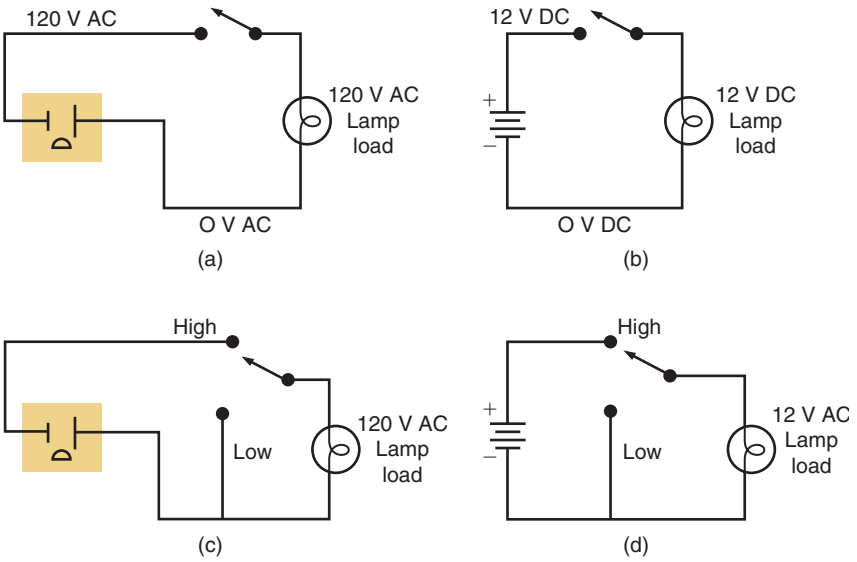



**FIGURE 1-2** (a) Typical 120 V AC house wiring; (b) typical 12 V DC automotive wiring; (c) 120 V AC model of a logic circuit; (d) 12 V AC model of a logic circuit.

for a simple light circuit. Chapter 8 will thoroughly explain why digital logic circuits operate like Figures 1-2(c) and (d) rather than like simple electrical wiring in your home or car, as depicted in Figures 1-2(a) and (b). The main point is that a 0 is typically represented by the LOW voltage or value near 0 V. The state designated as 1 is typically represented by a HIGH voltage and the value of that voltage depends on the technology of the system. These values of HIGH and LOW are often referred to as **logic levels.**

Some digital devices are activated by applying a HIGH, while others are activated by applying a LOW. **Figure 1-3** demonstrates these two scenarios for a simple light circuit. Notice that in Figure 1-3(a) the switch supplies the HIGH by connecting the voltage source which supplies current from the battery to the light and activates the light. In Figure 1-3(b), the switch supplies the LOW by connecting the return path from the light to the battery in order to activate the light. In Chapter 3, we will further investigate this concept of a device being active-HIGH or active-LOW.

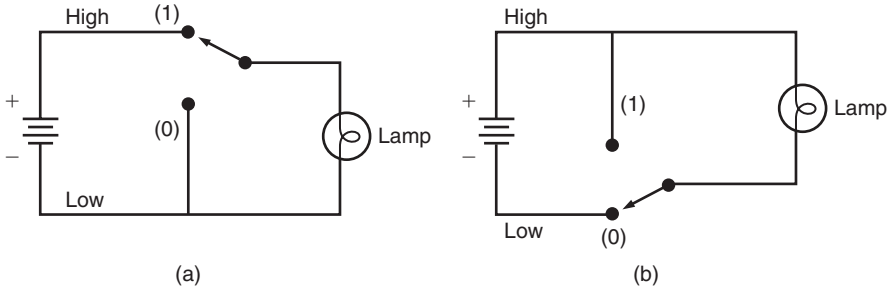



**FIGURE 1-3** (a) Applying HIGH turns the lamp ON; (b) applying LOW turns the lamp ON.

Sensors that serve as inputs to digital systems also can be wired in many different ways. For example, consider a circuit that can determine if the key to a car has been inserted into the ignition switch. As we are often reminded, this piece of information is used to sound an alarm if the car door is opened when the key is still in the ignition. **Figure 1-4** demonstrates two possible ways to wire this switch and the affect each method has on the meaning of the digital output level. In Figure 1-4(a), the contacts are open, producing a LOW when no key is present. When the key is inserted, as in Figure 1-4(b), it pushes contact points to the +12 V position, producing a

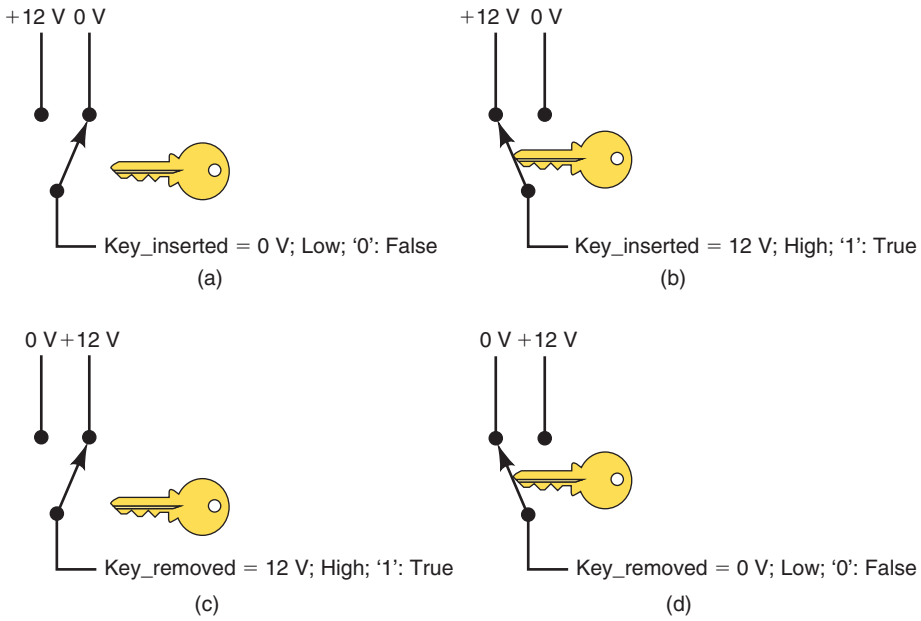



**FIGURE 1-4** Physical conditions, logic levels, and signal labels: (a) false that key is inserted, (b) true that key is inserted, (c) true that key is removed, (d) false that key is removed.

HIGH at the output. A good label for the output signal from this circuit would be *key_inserted* because the logic level of HIGH represents the state of 1 or true. *Key_inserted* is true when the output is HIGH. Contrast this circuit with Figure 1-4(c) in which the switch contacts are wired in the opposite way. In this case, inserting the key produces a LOW (Figure 1-4(d)) and removing the key produces a HIGH (Figure 1-4(c)). A good label for this signal is *key_removed* because it is true that the key is removed when the output is HIGH. The name of the signal describes a physical condition which should be true when the level is HIGH or 1. Chapters 3 and 4 will expand on these concepts using HIGHs and LOWs to activate/deactivate other circuits. This is fundamental to understanding all digital systems.

Now that we know that 1s are represented by a HIGH voltage and 0s by LOW voltage, all that remains is defining how high the voltage must be to be considered a 1 and how low a voltage must be to be considered a 0. The answer to this question also depends on the technology used to implement the digital system. Electronic digital systems have gone through many changes as technology has advanced. But the principles of representing 1s and 0s remain the same. In all systems, a defined range of higher voltages is acceptable as a HIGH (1). Another defined range of lower voltages is acceptable as a LOW (0). In between is a range of voltages that is considered neither HIGH nor LOW. Voltages in this range are considered invalid. **Figure 1-5**

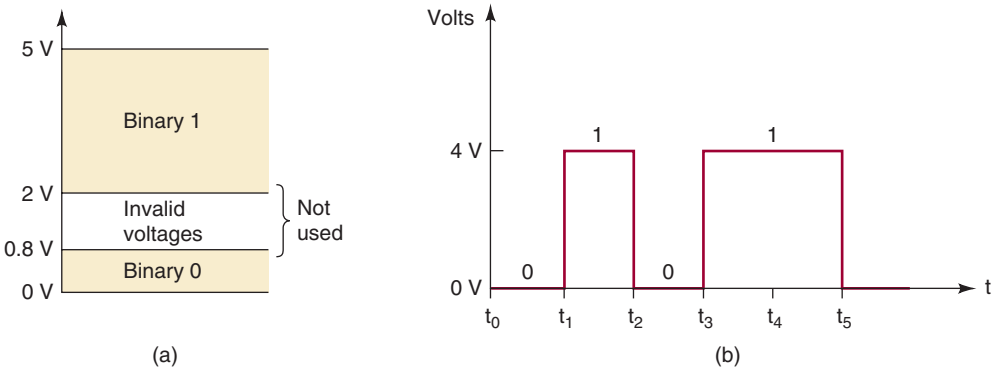



**FIGURE 1-5** Logic levels and timing (a) typical voltage ranges for a given technology of digital circuits. (b) a graph of signal levels changing over time.

demonstrates this concept for 5-volt logic systems that were based on bipolar transistor technology. Figure 1-5a indicates that in order for circuits using this technology to recognize the input as a '1' it must be a voltage greater than two but less than five. The input voltage must be less than 0.8 V to recognize it as a '0'. In the evolution of digital systems, various technologies such as electromechanical switches (relays), vacuum tubes, bipolar transistors, and MOSFET transistors have been used to implement digital logic circuits, each with their own characteristic definition of how to represent a 1 and a 0.

It is quite common and often necessary to depict the activity of a logic level over time. We called this a **timing diagram**. Figure 1-5(b) represents a typical digital waveform for the voltage ranges defined in part (a). The time axis is labeled at specific points in time, $t_1$, $t_2$, ... $t_5$. Notice that the HIGH voltage level between $t_1$ and $t_2$ is at 4 V. In digital systems, the exact value of a voltage is not important. A HIGH voltage of 3.7 V or 4.3 V would represent the exact same information. Likewise, a LOW voltage of 0.3 V represents the same information as 0 V. This points out a significant difference between analog and digital systems. In an analog system, the exact voltage is important. For example, if the analog voltage coming from a sensor is proportional to temperature, then 3.7 V would represent a different value of temperature than 4.3 V. In other words, the voltage carries significant information in the analog system. Circuits that can preserve exact voltages are much more complicated than digital circuits that simply need to recognize a voltage in one of two ranges. Digital circuits are designed to produce output voltages that fall within the prescribed 0 and 1 voltage ranges such as those defined in Figure 1-5. Likewise, digital circuits are designed to respond predictably to input voltages that are within the defined 0 and 1 ranges. What this means is that a digital circuit will respond in the same way to all input voltages that fall within the allowed 0 range; similarly, it will not distinguish between input voltages that lie within the allowed 1 range. To illustrate, Figure 1-6 represents a typical digital circuit with input $v_i$ and output $v_o$. The output is shown for two different input signal waveforms. Note that $v_o$ is the same for both cases because the two input waveforms, while differing in their exact voltage levels, are at the same binary levels.

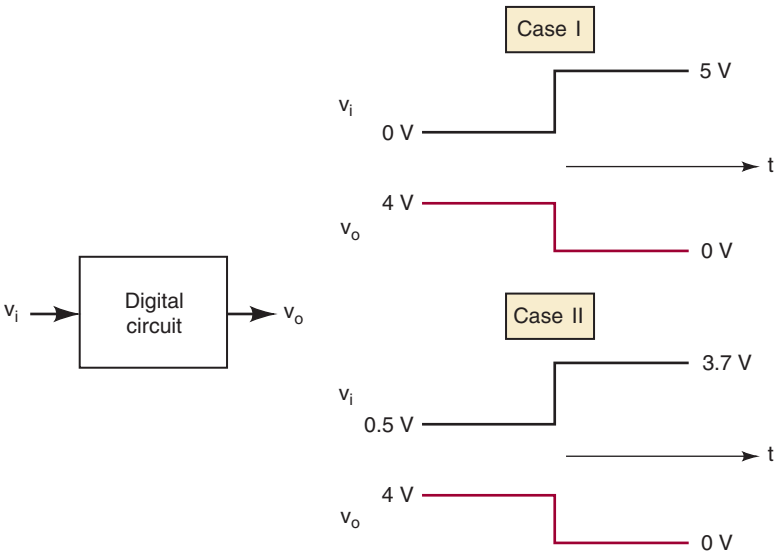



**FIGURE 1-6** A digital circuit responds to an input's binary level (0 or 1) and not to its actual voltage.

**OUTCOME ASSESSMENT QUESTIONS***

1. What are the two numeric digits used to represent states in a digital system?
2. What are the two terms used to represent the two logic levels?
3. What is the abbreviation for binary digit?
4. Which binary digit value is typically represented by low (near-zero) voltage?
5. What voltage represents the binary digit value of 1?
6. Which logic level is typically assigned a value of 1?
7. What is the logic level produced in Figure 1-4(a) when the key is removed?
8. According to Figure 1-5, what is the lowest voltage that would be recognized as a logic 1?
9. According to Figure 1-5, what is the highest voltage that would be recognized as a logic 0?
10. According to Figure 1-5, how would a voltage of 1.0 V be recognized?

## 1-2 DIGITAL SIGNALS

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Determine if a waveform is periodic or not.
- Measure period and frequency.
- Measure duty cycle.
- Identify events and classify edges as rising or falling.
- Recognize valid/invalid inputs.
- Recognize a timing diagram.

Suppose we have a light sensor that is intended to turn on the streetlights at night. An example of a circuit that could perform this task is shown in **Figure 1-7(a)**. Chapter 8 will explain more about analog comparators. This circuit's output will produce a logic 1 when no light is present (darkness). It outputs a logic 0 (0 V) when a certain level of light is present. The signal that comes from the sensor should be labelled with a signal name. It will always be either a 1 (HIGH) or a 0 (LOW) but it should be named something that informs the user about the physical condition represented by the signal. For example, if this sensor is intended to control a street lamp, the name of the output signal should be something like "night_time". When the signal is "1" it is true that it is nighttime. When the output is "0" we can say that it is false that it is nighttime. Chapter 3 will expand on these labelling techniques.

When a circuit like this is placed in service, it will output a 1 at night and a 0 during the day. At some point around dawn, it will change from a 1 to a 0. Around dusk, it will change from a 0 to a 1. This transition between the two states is called an **edge**. *At dawn, when the signal proceeds from HIGH to LOW, it is considered a* ***falling edge****, or* ***negative edge****.* Graphing the logic state over time tells us something about the operation of the system. **Figure 1-7(b)** shows the graph over time of the output of the light sensor.

*Answers to outcome assessment questions are found at the end of the chapter in which they occur.

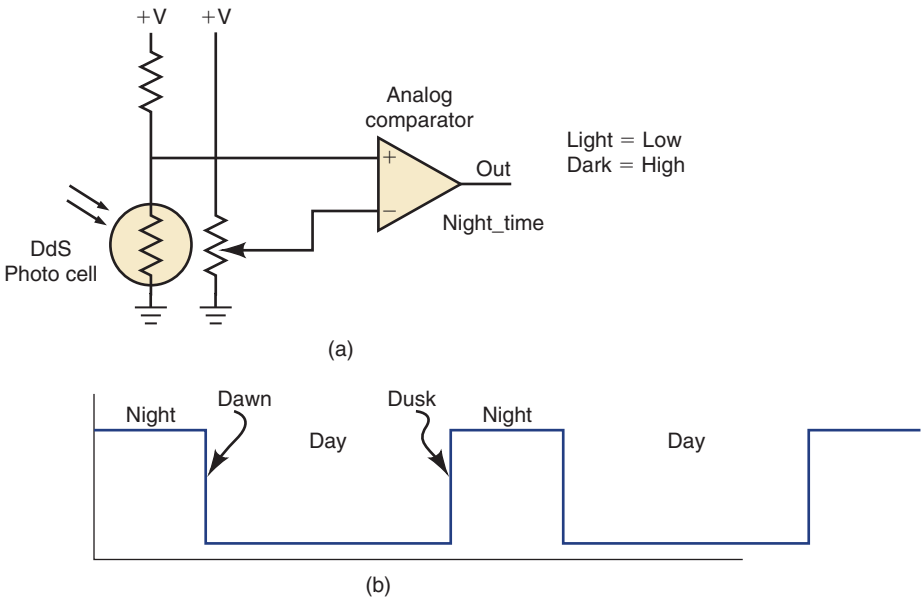



**FIGURE 1-7** (a) Darkness sensor; (b) a timing diagram of the output.

## Need for Timing

Digital circuits have inputs that are in one of two states: 1 or 0. The outputs are also either producing a 1 or a 0. In the previous section, we learned that 1s and 0s are represented by prescribed voltages and that voltage changes on the inputs result in changes in the output voltage. It can be very helpful to show the relationship between changes at the input and changes at the output in order to demonstrate the operation of the system. This means the logic states must be observed over time. Timing diagrams show the relationship, over time, between many digital "signals." It is very important that you understand timing diagrams and can relate them to physical events in a digital circuit. For example, assume there is a circuit represented by the block diagram in **Figure 1-8** that detects the "edge" at dawn, waits 10 minutes, and then turns off the streetlamp. Figure 1-8(b) is a timing diagram which shows the input to the circuit as well as the output. From this diagram, we can determine the relationship between the two signals. Notice the curvy arrows. They are used to indicate the cause-and-effect relationship between input and output signals.

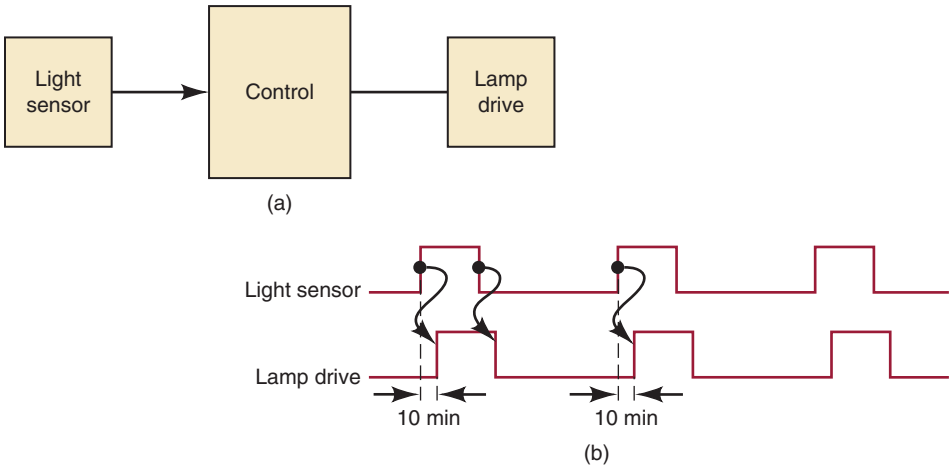



**FIGURE 1-8** Timing diagram with input and output.

## Highs and Lows Over Time

Think about a common digital input to a system that you operate all the time. A microwave oven has a switch in the door that tells the system whether the door is closed or open. This switch could be wired in several ways. Let's assume the switch is open when the door is open and closed when the door is closed. It is wired as shown in **Figure 1-9(a)**. The timing diagram in Figure 1-9(b) depicts the condition of the door over time. We can look at the diagram at any point in time and know the physical condition of the door.

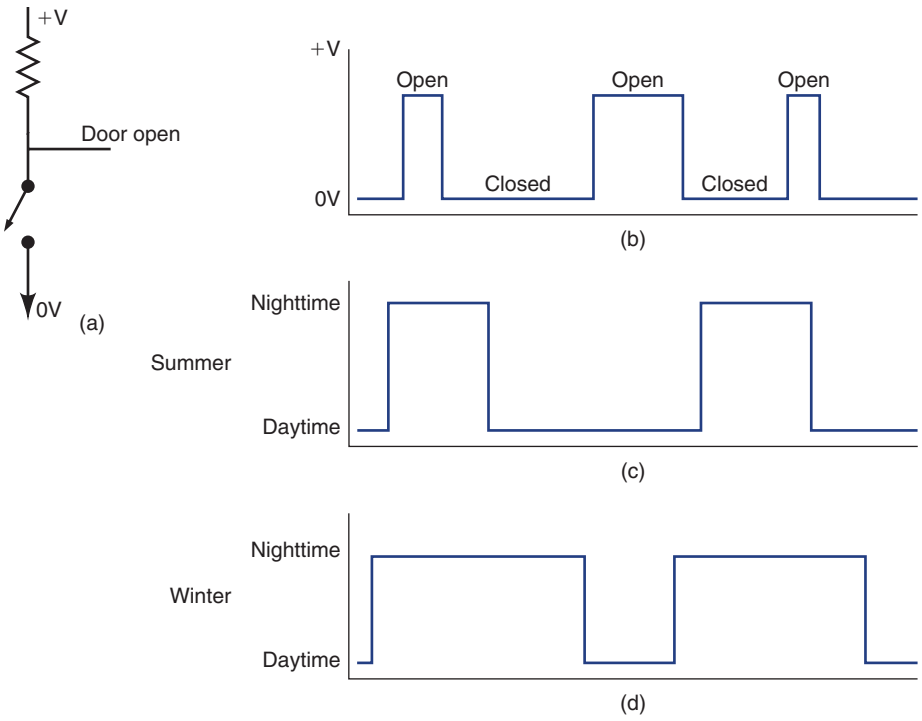



**FIGURE 1-9** A periodic versus periodic signals with duty cycle: (a) microwave door sensor, (b) aperiodic operation of oven door, (c) periodic day/night signal summer short nights, (d) periodic day/night signal winter short days.

## Periodic/Aperiodic

Opening and closing a microwave oven door is something that happens at completely irregular intervals. If we tried to measure the length of time the door stands open, each measurement would be different. There is no regularity to the cycle of opening and closing a door. There would be no fixed period of time between events. Therefore, it is referred to as an **aperiodic** signal. Let's contrast this digital signal with a sensor that turns on and off the streetlights. To make our point in this analogy, we will disregard the effects of weather and assume cloudless days. We also assume the sensor makes one clean transition at dawn and another clean transition at dusk. The sensor tells us whether it is day or night. A timing diagram of this sensor would look like **Figure 1-9(c)** in June (central United States). In December the sensor timing diagram would look more like Figure 1-9(d).

## Period/Frequency

Notice the similarities and differences in the timing waveforms of Figures 1-9(c) and (d). The length of daylight time is different between June and December, but the time it takes for an entire day is always the same. The earth always takes 24 hours for one rotation or one complete cycle. When you measure from dawn to dawn it is always the same, regardless of the season. Likewise, notice

that the amount of time from dusk to dusk is always the same as well. When a system operates such that the time for one complete cycle is always constant, it is called a ***periodic*** system. Certainly, the rotation of the earth is periodic and its period is always 24 hours. The period of any wave can be defined as the amount of time per cycle (seconds/cycle). The frequency of a periodic wave is defined as the number of cycles per unit time (cycles/second). In other words, frequency ($F$) and period ($T$) are reciprocals.

$$F = 1/T \quad T = 1/F$$

## Duty Cycle

The length of daylight time and nighttime varies with the seasons but the period remains the same. If we want to measure how much of the time a digital signal is in its "active" state, then we must think about the purpose of the digital signal. In our example of a sensor whose duty is to turn the streetlights ON, we would say that this sensor is on-duty during the night when (in this example) the sensor is HIGH. The duty cycle of the street light would be the percentage of time it is dark over the course of an entire day.

$$\text{Duty Cycle} = \text{Active pulse Width/Period} = t_w/T$$

## Transitions

Just as you realize that the night does not change instantly into day, it is true that no digital signal can truly change instantly from LOW to HIGH. There is a time of transition. It is common to declare the transition as happening when the signal is half way between the two states. Measurements are taken from the 50% point of waveform. For example, the width of the HIGH pulse is measured as shown in **Figure 1-10**. The period $T$ is also measured from 50% points as shown. Chapter 5 will have more to say about measurements of these transition times and the period of a digital waveform.

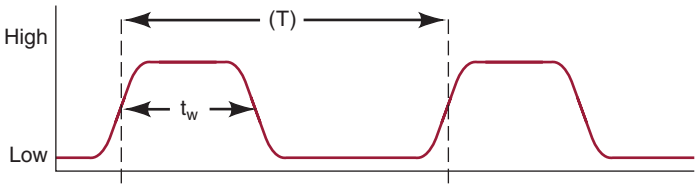



**FIGURE 1-10** Measuring pulse width and period.

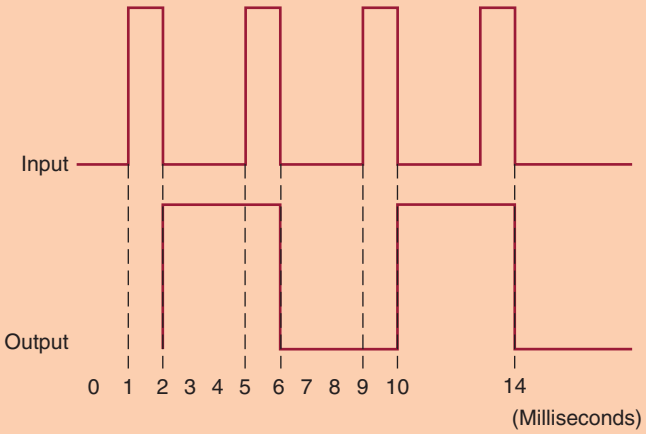
## Edges/Events

Whenever you have a system with only two states, the only thing that can be considered an "**event**" is when the system changes states. A transition from LOW to HIGH or HIGH to LOW is considered an "event" in digital systems. On timing diagrams, these transitions appear as sharp "edges." Some events are rising edges and some are falling edges. We will learn in Chapter 3 that there are circuits that respond to HIGH levels (active HIGH) and circuits that respond to LOW levels (active LOW). Circuits that respond to a particular level are often considered to be ***level triggered***. Other types of digital circuits respond to either rising edges or falling edges. These are called ***edge triggered*** circuits. Chapter 5 will introduce edge triggered devices.

**OUTCOME ASSESSMENT QUESTIONS**

1. Draw a timing diagram showing when a person is "on duty" over an entire week. Begin on Monday morning. The diagram will have one input representing the day/night cycle (assume equinox where length of day = length of night dawn 6:00 am, dusk 6:00 pm) and one output which goes HIGH representing when a person in "on duty." Assume they work a typical 8–5 job Monday–Friday with Saturday and Sunday off.
2. Is the "on_duty" waveform in the diagram from the previous question periodic or aperiodic?
3. Refer to **Figure 1-11**
   (a) Is the input waveform periodic?
   (b) What is the period of the input waveform in sec?
   (c) What is the active-HIGH duty cycle of the input waveform?
   (d) What is the frequency of the waveform in Hz?
   (e) What type of event on the input causes a change on the output?
   (f) What is the period of the output waveform in sec?
   (g) What is the frequency of the output waveform in Hz?

**FIGURE 1-11** Outcome assessment question.




## 1-3 LOGIC CIRCUITS AND EVOLVING TECHNOLOGY

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Identify acceptable digital logic levels for a given technology.
- Recognize terms describing the currently prevalent and legacy technologies for digital circuits.

### Logic Circuits

The manner in which a digital circuit responds to an input is referred to as the circuit's *logic*. Each type of digital circuit obeys a certain set of logic rules. For this reason, digital circuits are also called **logic circuits.** We will use both terms interchangeably throughout the text. In Chapter 3, we will see more clearly what is meant by a circuit's "logic."

We will be studying all the types of logic circuits that are currently used in digital systems. Initially, our attention will be focused only on the logical operation that these circuits perform—that is, the relationship between

the circuit inputs and outputs. We will defer any discussion of the internal circuit operation of these logic circuits until after we have developed an understanding of their logical operation.

## Digital Integrated Circuits

Digital circuits of today's technology are primarily implemented using very sophisticated integrated circuits (ICs) that are electronically configured or tailor-made for their application. Many technologies of the past are completely obsolete. For example, the vacuum tube logic circuits would never be used today for a number of reasons such as too big, too much power, and the vacuum tubes are very hard to find. Occasionally, it makes sense to use a mature technology where it is economical and the parts will be available over the life of a product. For example, most of the ICs that made up digital systems in the 1970s are no longer being manufactured but are still available on the market from large left-over inventories. These devices will, on rare occasion, be used in a new product and they are still used for laboratory instruction for digital courses in high school and college. Throughout this text, we will try to provide enough information about the range of technologies to allow you to learn using simple devices from the past and yet introduce you to the fundamentals necessary to use the tools of the future.

Today the most common technology used to implement digital circuits (including the vast majority of computer hardware) is **CMOS,** which stands for **Complementary Metal-Oxide Semiconductor**. Other technologies have been relegated to much smaller niches in the marketplace. Prior to the advancement of CMOS technology, bipolar transistor technology was king and had a profound influence on digital systems. The major logic family that sprung from bipolar technology is referred to as **TTL (Transistor/Transistor Logic).** You will learn about the various IC technologies, their characteristics, and relative advantages and disadvantages in Chapter 8.

**OUTCOME ASSESSMENT QUESTIONS**

1. *True or false*: The exact value of an input voltage is critical for a digital circuit.
2. Can a digital circuit produce the same output voltage for different input voltage and current values?
3. A digital circuit is also referred to as a _____ circuit.
4. The most prevalent technology used for digital circuits today is abbreviated _____.
5. This acronym stands for _____ _____ _____ _____.
6. Legacy systems of the past used a technology abbreviated as _____
7. The type of transistor used in legacy systems was the _____ transistor.

# 1-4 NUMERICAL REPRESENTATIONS

## OUTCOMES

*Upon completion of this section, you will be able to:*

- Discriminate between digital and analog representations.
- Identify examples of each type of representation.

In science, technology, business, and, in fact, most other fields of endeavor, we are constantly dealing with *quantities.* Quantities are measured, monitored, recorded, manipulated arithmetically, observed, or in some other way utilized in most physical systems. It is important when dealing with various quantities that we be able to represent their values efficiently and accurately. There are basically two ways of representing the numerical value of quantities: *analog* and *digital.*

## Analog Representations

In **analog representation** a quantity is represented by a continuously variable, proportional indicator. An example is an automobile speedometer from the classic muscle cars of the 1960s and 1970s. The deflection of the needle is proportional to the speed of the car and follows any changes that occur as the vehicle speeds up or slows down. On older cars, a flexible mechanical shaft connected the transmission to the speedometer on the dashboard. It is interesting to note that on newer cars, the analog representation is usually preferred even though speed is now measured digitally.

Thermometers before the digital revolution used analog representation to measure temperature, and many are still in use today. Mercury thermometers use a column of mercury whose height is proportional to temperature. These devices are being phased out of the market because of environmental concerns, but nonetheless they are an excellent example of analog representation. Another example is an outdoor thermometer on which the position of the pointer rotates around a dial as a metal coil expands and contracts with temperature changes. The position of the pointer is proportional to the temperature. Regardless of how small the change in temperature, there will be a proportional change in the indication.

In these two examples the physical quantities (speed and temperature) are being coupled to an indicator by purely mechanical means. In electrical analog systems, the physical quantity that is being measured or processed is converted to a proportional voltage or current (electrical signal). This voltage or current is then used by the system for display, processing, or control purposes.

No matter how they are represented, analog quantities have an important characteristic: *they can vary over a continuous range of values.* The automobile speed can have *any* value between zero and, say, 100 mph. Similarly, the temperature indicated by an analog thermometer can have any value from −20° to 120° Fahrenheit.

## Digital Representations

In **digital representation** the quantities are represented not by continuously variable indicators but by symbols called *digits.* As an example, consider a digital indoor/outdoor thermometer. It has four digits and can measure changes of 0.1°. The actual temperature gradually increases from, say, 72.0 to 72.1 but the digital representation changes suddenly from 72.0 to 72.1. In other words, this digital representation of outdoor temperature changes in *discrete* steps, as compared with the analog representation of temperature provided by a fluid column or metal coil thermometer, where the reading changes continuously.

The major difference between analog and digital quantities, then, can be simply stated as follows:

$$\text{Analog} \equiv \text{continuous}$$
$$\text{Digital} \equiv \text{discrete (step by step)}$$

Because of the discrete nature of digital representations, there is no ambiguity when reading the value of a digital quantity, whereas the value of an analog quantity is often open to interpretation. In practice, when we take a measurement of an analog quantity, we always "round" to a convenient level of precision. In other words, we digitize the quantity. The digital representation is the result of assigning a number of limited precision to a continuously variable quantity.

The world around us is full of physical variables that are constantly changing. If we can measure these variables and represent them as a digital quantity, then we can record, arithmetically manipulate, or in some other way use these quantities to control things.

**EXAMPLE 1-1**

Which of the following involve analog quantities and which involve digital quantities?

(a) Elevation using a ladder
(b) Elevation using a ramp
(c) Current flowing from an electrical outlet through a motor
(d) Height of a child measured by a yard stick ruler
(e) Height of a child measured by putting a mark on the wall
(f) Amount of rocks in a bucket
(g) Amount of sand in a bucket
(h) Volume of water in a bucket

**Solution**

(a) Digital
(b) Analog
(c) Analog
(d) Digital: measured to nearest $\frac{1}{8}$ inch
(e) Analog
(f) Digital: can only increase/decrease by one rock
(g) Digital: can only increase/decrease by discrete grains of sand
(h) Analog: (unless you want to get to the nanotechnology level!)

**OUTCOME ASSESSMENT QUESTIONS**

1. Which method of representing quantities involves discrete steps?
2. Which method of representing quantities is continuously variable?
3. Identify each as digital or analog representation:
   (a) Time of day using a sundial
   (b) Time of day using your cell phone
   (c) Volume level of your flat-screen television
   (d) Volume level of vacuum tube radio
   (e) Measuring the circumference of a basketball in millimeters
   (f) Measuring the circumference of a basketball by wrapping a string around it and cutting the string to length

# 1-5 DIGITAL AND ANALOG SYSTEMS

## OUTCOMES

*Upon completion of this section, you will be able to:*

- Identify advantages of digital techniques.
- Identify limitations of digital techniques.

A **digital system** is a combination of devices designed to manipulate logical information or physical quantities that are represented in digital form; that is, the quantities can take on only discrete values. These devices are most often electronic, but they can also be mechanical, magnetic, or pneumatic. Some of the more familiar digital systems include digital computers and calculators, digital audio and video equipment, and the telecommunications system.

An **analog system** contains devices that manipulate physical quantities that are represented in analog form. In an analog system, the quantities can vary over a continuous range of values. For example, the amplitude of the output signal to the speaker in a radio receiver can have any value between zero and its maximum limit.

## Advantages of Digital Techniques

An increasing majority of applications in electronics, as well as in most other technologies, use digital techniques to perform operations that were once performed using analog methods. The chief reasons for the shift to digital technology are:

1. *Digital systems are generally easier to design.* The circuits used in digital systems are *switching circuits*, where *exact* values of voltage or current are not important, only the range (HIGH or LOW) in which they fall.
2. *Information storage is easy.* This is accomplished by special devices and circuits that can latch onto digital information and hold it for as long as necessary, and mass storage techniques that can store billions of bits of information in a relatively small physical space. Analog storage capabilities are, by contrast, extremely limited.
3. *Accuracy and precision are easier to maintain throughout the system.* Once a signal is digitized, the degree to which it deteriorates is predictable and more easily contained within acceptable limits. In analog systems, the voltage and current signals tend to be distorted by the effects of temperature, humidity, and component tolerance variations in the circuits that process the signal.
4. *Operations can be programmed.* It is fairly easy to design digital systems whose operation is controlled by a set of stored instructions called a *program*. Analog systems can also be *programmed*, but the variety and the complexity of the available operations are severely limited.
5. *Digital circuits are less affected by noise.* Spurious fluctuations in voltage (noise) are not as critical in digital systems because the exact value of a voltage is not important, as long as the noise is not large enough to prevent us from distinguishing a HIGH from a LOW.
6. *More digital circuitry can be fabricated on IC chips.* It is true that analog circuitry has also benefited from the tremendous development of IC technology, but its relative complexity and its use of devices that cannot be economically integrated (high-value capacitors, precision resistors, inductors, transformers) have prevented analog systems from achieving the same high degree of integration.

## Limitations of Digital Techniques

There are really very few drawbacks when using digital techniques. The two biggest problems are:

**The real world is analog and digitizing always introduces some error.**
**Processing digitized signals takes time.**

Most physical quantities are analog in nature, and these quantities are often the inputs and outputs that are being monitored, operated on, and controlled by a system. Some examples are temperature, pressure, position, velocity, liquid level, flow rate, and so on. We are in the habit of expressing these quantities *digitally*, such as when we say that the temperature is 64° (63.8° when we want to be more precise), but we are really making a digital approximation to an inherently analog quantity.

To take advantage of digital techniques when dealing with analog inputs and outputs, four steps must be followed:

1. Convert the physical variable to an electrical signal (analog).
2. Convert the electrical (analog) signal into digital form.
3. Process (operate on) the digital information.
4. Convert the digital outputs back to real-world analog form.

An entire book could be written about step 1 alone. There are many kinds of devices that convert various physical variables into electrical analog signals (sensors). These are used to measure things that are found in our "real" analog world. On your car alone, there are sensors for fluid level (gas tank), temperature (climate control and engine), velocity (speedometer), acceleration (airbag collision detection), pressure (oil, manifold), and flow rate (fuel), to name just a few. Chapter 11 will cover the devices that convert analog to digital.

To illustrate a typical system that uses this approach Figure 1-12 describes a precision temperature regulation system. A user pushes up or down buttons to set the desired temperature in 0.1° increments (digital representation). A temperature sensor in the heated space converts the measured temperature to a proportional voltage. This analog voltage is

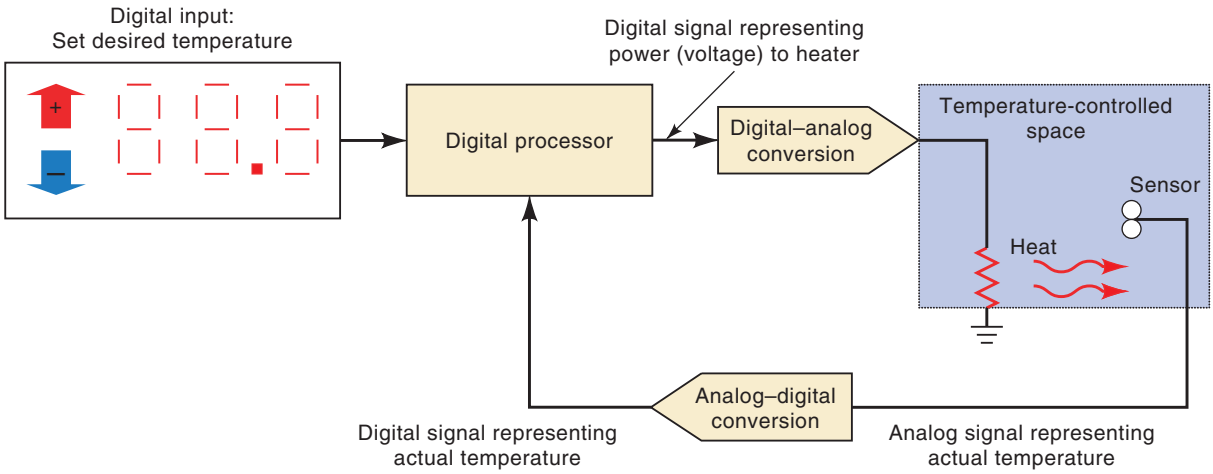



**FIGURE 1-12** Diagram of a precision digital temperature control system.

converted to a digital quantity by an **analog-to-digital converter (ADC).** This value is then compared to the desired value and used to determine a digital value of how much heat is needed. The digital value is converted to an analog quantity (voltage) by a **digital-to-analog converter (DAC)**. This voltage is applied to a heating element, which will produce heat that is related to the voltage applied and will affect the temperature of the space.

**OUTCOME ASSESSMENT QUESTIONS**

1. List three advantages of digital techniques.
2. List the two primary limitations of digital techniques.

## 1-6 DIGITAL NUMBER SYSTEMS

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Identify the weight of each binary digit.
- Determine the range of binary values given the number of binary digits.
- Interpret binary numbers into decimal.
- Count in binary.

Many number systems are in use in digital technology. The most common are the decimal, binary, and hexadecimal systems. Humans operate using decimal numbers, digital systems operate using binary numbers, and **hexadecimal** is a number system that makes it easier for humans to deal with binary numbers. All three of these number systems are defined and function in the exact same way. Let's start by examining the decimal system. Because it is so familiar, we rarely stop to think about how this number system actually works. Examining its characteristics fully will help you to understand the other systems better.

### Decimal System

The **decimal system** is composed of *10* numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; using these symbols as *digits* of a number, we can express any quantity. The decimal system, also called the *base-10* system because it has 10 digits, has evolved naturally as a result of the fact that people have 10 fingers. In fact, the word *digit* is derived from the Latin word for "finger."

The decimal system is a *positional-value system* in which the value of a digit depends on its position. For example, consider the decimal number 453. We know that the digit 4 actually represents 4 *hundreds*, the 5 represents 5 *tens*, and the 3 represents 3 *units.* In essence, the 4 carries the most weight of the three digits; it is referred to as the *most significant digit (MSD)*. The 3 carries the least weight and is called the *least significant digit (LSD)*.

Consider another example, 27.35. This number is actually equal to 2 tens plus 7 units plus 3 tenths plus 5 hundredths, or $2 \times 10 + 7 \times 1 + 3 \times 0.1 + 5 \times 0.01$. The decimal point is used to separate the integer and fractional parts of the number.

More rigorously, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in

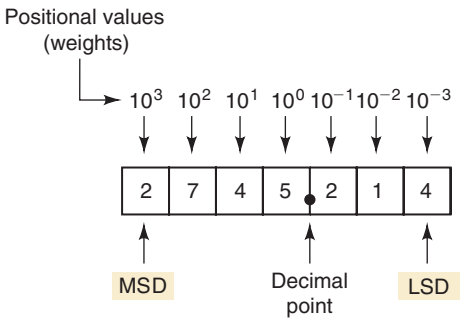



**FIGURE 1-13** Decimal position values as powers of 10.

Figure 1-13, where the number 2745.214 is represented. The decimal point separates the **positive** powers of 10 from the negative powers. The number 2745.214 is thus equal to

$$(2 \times 10^{+3}) + (7 \times 10^{+2}) + (4 \times 10^{1}) + (5 \times 10^{0}) + (2 \times 10^{-1}) + (1 \times 10^{-2}) + (4 \times 10^{-3})$$

In general, any number is simply the sum of the products of each digit value and its positional value.

## Decimal Counting

When counting in the decimal system, we start with 0 in the units position and take each symbol (digit) in progression until we reach 9. Then we add a 1 to the next higher position and start over with 0 in the first position (see Figure 1-14). This process continues until the count of 99 is reached. Then we add a 1 to the third position and start over with 0s in the first two positions. The same pattern is followed continuously as high as we wish to count.

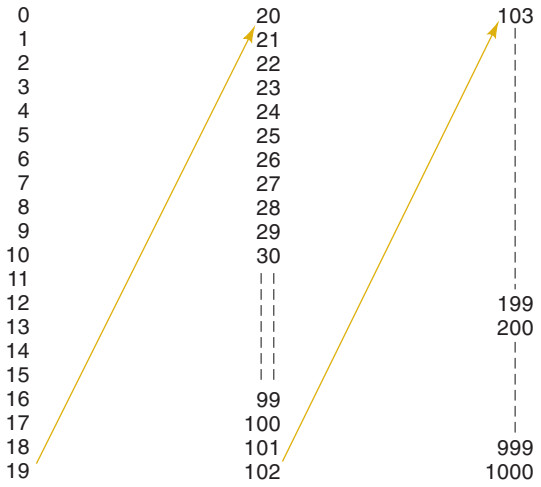



**FIGURE 1-14** Decimal counting.

It is important to note that in decimal counting, the units position (LSD) changes upward with each step in the count, the tens position changes upward every 10 steps in the count, the hundreds position changes upward every 100 steps in the count, and so on.

Another characteristic of the decimal system is that using only two decimal places, we can count through $10^2 = 100$ different numbers (0 to 99).* With three places we can count through 1000 numbers (0 to 999), and so on. In general, with $N$ places or digits, we can count through $10^N$ different numbers, starting with and including zero. The largest number will always be $10^N - 1$.

## Binary System

Unfortunately, the decimal number system does not lend itself to convenient implementation in digital systems. For example, it is very difficult to design electronic equipment so that it can work with 10 different voltage levels (each one representing one decimal character, 0 through 9). On the other hand, it is very easy to design simple, accurate electronic circuits that operate with only two voltage levels. For this reason, almost every digital system uses the binary (base-2) number system as the basic number system of its operations. Other number systems are often used to interpret or represent binary quantities for the convenience of the people who work with and use these digital systems.

In the binary system there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems. In general though, it will take a greater number of binary digits to express a given quantity.

All of the statements made earlier concerning the decimal system are equally applicable to the binary system. The binary system is also a positional-value system, wherein each binary digit has its own value or weight expressed as a power of 2. This is illustrated in Figure 1-15. Here, places to the left of the *binary point* (counterpart of the decimal point) are positive powers of 2 and places to the right are negative powers of 2. The number 1011.101 is shown represented in the figure. To find its equivalent in the decimal system, we simply take the sum of the products of each digit value (0 or 1) and its positional value:

$$\begin{aligned} 1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\ &= 11.625_{10} \end{aligned}$$

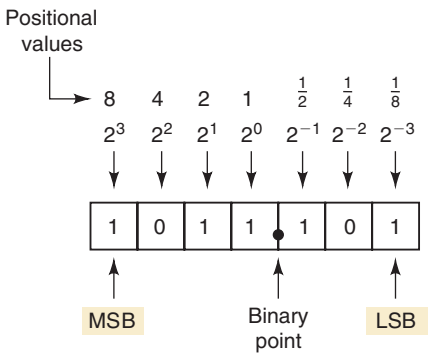



**FIGURE 1-15** Binary position values as powers of 2.

* Zero is counted as a number.

Notice in the preceding operation that subscripts (2 and 10) were used to indicate the base in which the particular number is expressed. This convention is used to avoid confusion whenever more than one number system is being employed.

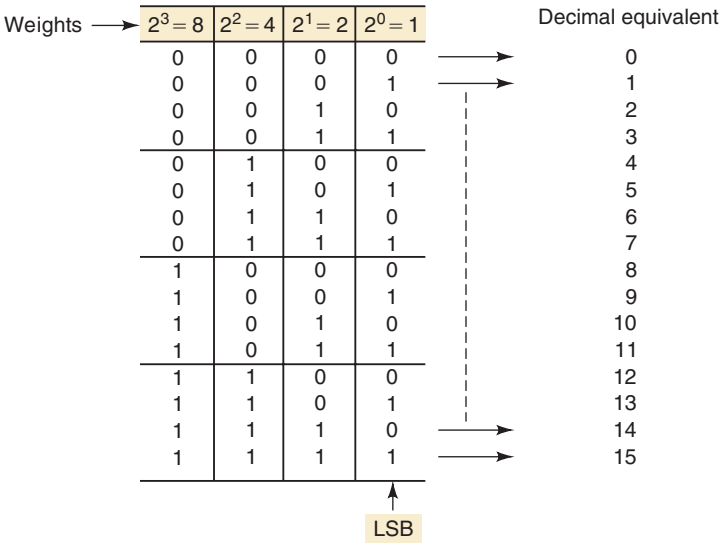
In the binary system, the term *binary digit* is often abbreviated to the term *bit*, which we will use from now on. Thus, in the number expressed in Figure 1-15 there are four bits to the left of the binary point, representing the integer part of the number, and three bits to the right of the binary point, representing the fractional part. The most significant bit (MSB) is the leftmost bit (largest weight). The least significant bit (LSB) is the rightmost bit (smallest weight). These are indicated in Figure 1-15. Here, the MSB has a weight of $2^3$; the LSB has a weight of $2^{-3}$. The weights of each digit increase by a factor of 2 as the position moves from right to left.

## Binary Counting

When we deal with binary numbers, we will usually be restricted to a specific number of bits. This restriction is based on the circuitry used to represent these binary numbers. Let's use four-bit binary numbers to illustrate the method for counting in binary.

The sequence (shown in Figure 1-16) begins with all bits at 0; this is called the *zero count.* For each successive count, the units ($2^0$) position *toggles*; that is, it changes from one binary value to the other. Each time the units bit changes from a 1 to a 0, the twos ($2^1$) position will toggle (change states). Each time the twos position changes from 1 to 0, the fours ($2^2$) position will toggle (change states). Likewise, each time the fours position goes from 1 to 0, the eights ($2^3$) position toggles. This same process would be continued for the higher order bit positions if the binary number had more than four bits.

**FIGURE 1-16** Binary counting sequence.




The binary counting sequence has an important characteristic, as shown in Figure 1-16. The units bit (LSB) changes either from 0 to 1 or 1 to 0 with *each* count. The second bit (twos position) stays at 0 for two counts, then at 1 for two counts, then at 0 for two counts, and so on. The third bit (fours position) stays at 0 for four counts, then at 1 for four counts, and so on. The fourth bit (eights position) stays at 0 for eight counts, then at 1 for eight counts. If we wanted to count further, we would add more places, and this

pattern would continue with 0s and 1s alternating in groups of $2^{N-1}$. For example, using a fifth binary place, the fifth bit would alternate sixteen 0s, then sixteen 1s, and so on.

As we saw for the decimal system, it is also true for the binary system that by using $N$ bits or places, we can go through $2^N$ counts. For example, with two bits we can go through $2^2 = 4$ counts ($00_2$ through $11_2$); with four bits we can go through $2^4 = 16$ counts ($0000_2$ through $1111_2$); and so on. The last count will always be all 1s and is equal to $2^N - 1$ in the decimal system. For example, using four bits, the last count is $1111_2 = 2^4 - 1 = 15_{10}$.

**EXAMPLE 1-2**

What is the largest number that can be represented using eight bits?

**Solution**

$2^N - 1 = 2^8 - 1 = 255_{10} = 11111111_2$.

This has been a brief introduction of the binary number system and its relation to the decimal system. We will spend much more time on these two systems and several others in the next chapter.

**OUTCOME ASSESSMENT QUESTIONS**

1. What is the decimal equivalent of $1101011_2$?
2. What is the next binary number following $10111_2$ in the counting sequence?
3. What is the largest decimal value that can be represented using 12 bits?

## 1-7 REPRESENTING SIGNALS WITH NUMERIC QUANTITIES

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Represent a continuous signal with a sequence of measurements.
- Evaluate the relative effects of precision in measurement.
- Evaluate the relative effects of frequency of measurement.
- Identify the acceptable range of voltages that represent 1s and 0s in a given digital system.

We will now look at the concept of representing an analog signal as a sequence of digital numbers. From the previous section, we know that any quantity can be represented by a binary number, just as easily as it can be represented by a decimal number. Suppose you are doing a science experiment that requires you to keep a record of temperature changes over a long period of time. You know that the temperature of the air is an analog quantity: continuously variable. However, you also know that temperature usually changes rather slowly, so instead of trying to continuously plot the current temperature, you decide to take a measurement at the top of every hour.

Your temperature measuring device is rather crude and can only measure in 10 degree increments. For example, any temperature between 60

and 69 (inclusive) will read 60 degrees. Any temperature between 70 and 79 (inclusive) will read 70 degrees. **Figure 1-17** shows a continuous plot of the temperature on an early summer day which starts out cool but warms up rapidly around 1:00 pm. After 1:00 pm, a thunderstorm comes through and the temperature drops very suddenly and drastically. **Figure 1-18(a)** shows the data taken from this day's temperature readings at the top of every hour. If we plot the numbers from our table, you can see in **Figure 1-18(b)** that the general shape is close to the analog signal even though some of the detailed slow changes are undetected.

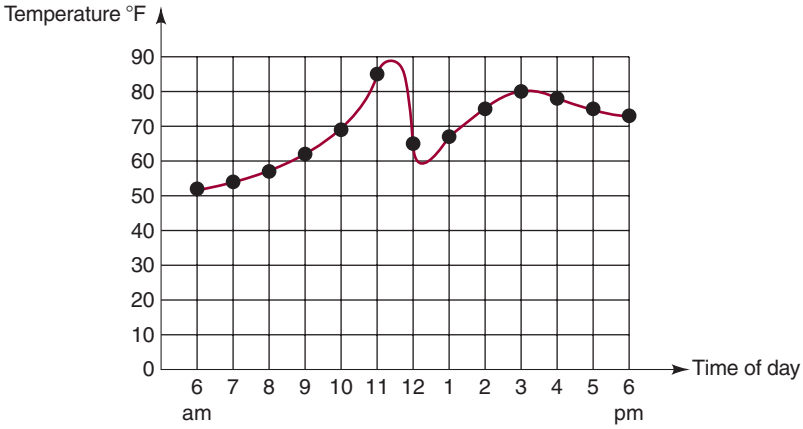



**FIGURE 1-17**
Temperature data sampled hourly. Red line indicates analog.

Next let's assume that we arbitrarily decided to measure the temperature every 2 hours starting at 6:00 am. These measurements are also shown in Figure 1-18a. The plot of the measurements taken every two hours is shown in **Figure 1-18c.** Notice it does not look at all like the actual analog signal. Based on these measurements, the temperature for this day appears to be very stable and quite mild. This example is intended to point out several things about representing analog signals as digital quantities.

1. The major event of the day (rapid increase in heat followed by a sudden drop in temperature) all happened in between two samples, so as far as this digital signal is concerned, the event did not happen.
2. The signal is represented as a list of measurements taken at regular intervals. These are called samples.
3. The measurements do not exactly represent the actual value at the time it is sampled due to the limitations of the measuring device (ten degree steps). This is called quantization error.
4. How often a sample is recorded has a huge impact on the accuracy of the reproduction.
5. The more frequently samples are taken, the more accurately the signal is represented.

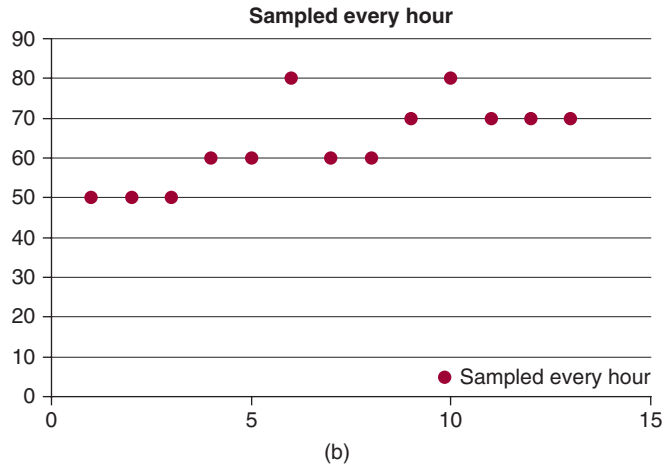
Chapter 11 will have more to say about digital signals and how they are processed.

**OUTCOME ASSESSMENT QUESTION**

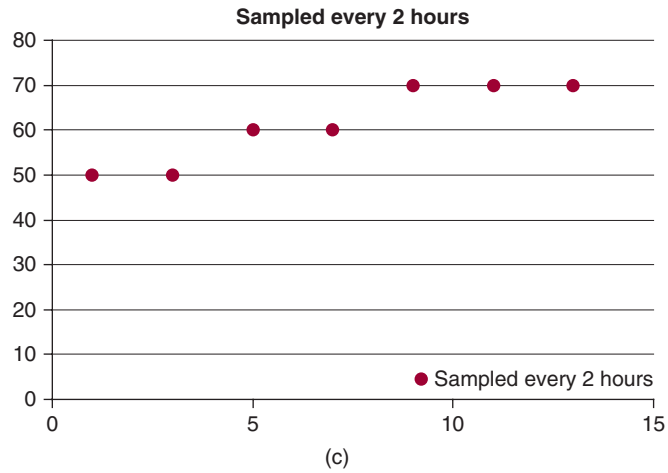
1. How is an analog signal represented in a digital system?

| Sample number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time of day | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 | 2 | 3 | 4 | 5 | 6 |
| Sampled every hour | 50 | 50 | 50 | 60 | 60 | 80 | 60 | 60 | 70 | 80 | 70 | 70 | 70 |
| Sampled every 2 hours | 50 | | 50 | | 60 | | 60 | | 70 | | 70 | | 70 |
| Binary value stored | 101 | 101 | 101 | 110 | 110 | 1000 | 110 | 110 | 111 | 1000 | 111 | 111 | 111 |

(a)




(b)




(c)

**FIGURE 1-18** Temperature measurements: (a) table of data; (b) graph of hourly samples; (c) graph of samples every 2 hours.

## 1-8 PARALLEL AND SERIAL TRANSMISSION

### OUTCOME

*Upon completion of this section, you will be able to:*

- Discriminate between parallel and serial transfer.

One of the most common operations that occur in any digital system is the transmission of information from one place to another. The information can be transmitted over a distance as small as a fraction of an inch on the same circuit board, or over a distance of many miles when two people are texting each other on different continents. The information that is transmitted is in binary form and is generally represented as voltages at the outputs of a sending circuit that are connected to the inputs of a receiving circuit.

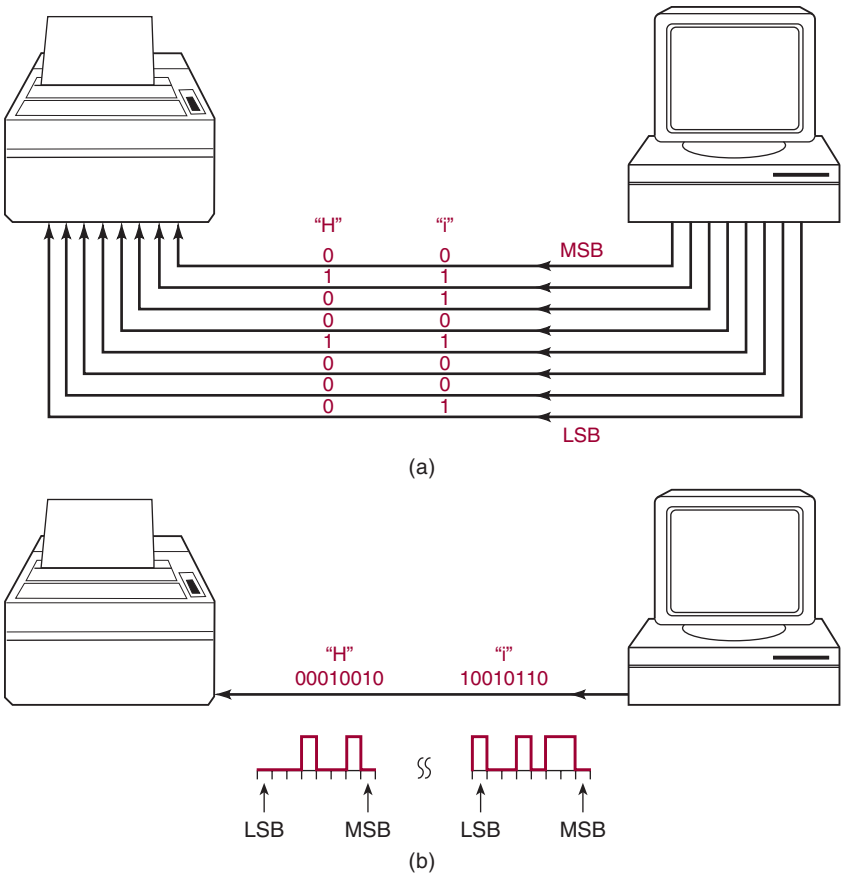



**FIGURE 1-19** (a) Parallel transmission uses one connecting line per bit, and all bits are transmitted simultaneously; (b) serial transmission uses only one signal line, and the individual bits are transmitted serially (one at a time).

Figure 1-19 illustrates the two basic methods for digital information transmission: **parallel** and **serial**.

Figure 1-19(a) demonstrates parallel transmission of data from a computer to a printer. Parallel printer interfaces were standard in personal computers before the USB (Universal Serial Bus). In this scenario, assume we are trying to print the word "Hi" on the printer. The binary code for "H" is 01001000 and the binary code for "i" is 01101001. Each character (the "H" and the "i") are made up of eight bits. Using parallel transmission, all eight bits are sent simultaneously over eight wires. The "H" is sent first, followed by the "i."

Figure 1-19(b) demonstrates serial transmission such as is employed on your computer when using a USB port to send data to a printer. Although the details of the data format are much more complicated for a USB port than we show here, the point is that the data are sent one bit at a time over a single wire. The bits are shown in the diagram as though they were actually moving down the wire in the order shown. The least significant bit of "H" is sent first and the most significant bit of "i" is sent last. Of course, in reality, only one bit can be on the wire at any point in time and time is usually drawn on a graph starting at the left and advancing to the right. This produces a graph of logic bits versus time of the serial transmission called a timing diagram. Notice that in this presentation, the least significant bit is shown on the left because it was sent first.

The principal trade-off between parallel and serial representations is one of speed versus circuit simplicity. The transmission of binary data from one part of a digital system to another can be done more quickly using parallel representation because all the bits are transmitted simultaneously,

while serial representation transmits one bit at a time. On the other hand, parallel requires more signal lines connected between the sender and the receiver of the binary data than does serial. In other words, parallel is faster, and serial requires fewer signal lines. This comparison between parallel and serial methods for representing binary information will be encountered many times in discussions throughout the text.

**OUTCOME ASSESSMENT QUESTION**

1. Describe the relative advantages of parallel and serial transmission of binary data.

## 1-9 MEMORY

### OUTCOME

*Upon completion of this section, you will be able to:*

- Articulate the difference between nonmemory and memory circuits.

When an input signal is applied to most devices or circuits, the output somehow changes in response to the input, and when the input signal is removed, the output returns to its original state. These circuits do not exhibit the property of *memory* because their outputs revert back to normal. In digital circuitry certain types of devices and circuits do have memory. When an input is applied to such a circuit, the output will change its state, but it will remain in the new state even after the input is removed. This property of retaining its response to a momentary input is called **memory**. Figure 1-20 illustrates nonmemory and memory operations.

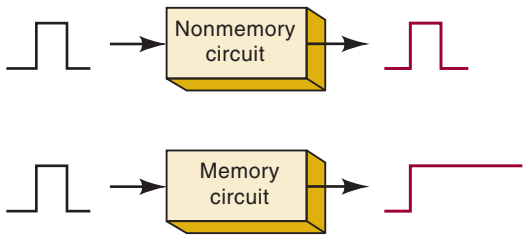



**FIGURE 1-20** Comparison of nonmemory and memory operation.

Memory devices and circuits play an important role in digital systems because they provide a means for storing binary numbers either temporarily or permanently, with the ability to change the stored information at any time. As we shall see, the various memory elements include magnetic and optical types and those that utilize electronic latching circuits (called *latches* and *flip-flops*).

**OUTCOME ASSESSMENT QUESTIONS**

1. A nonmemory circuit output always depends on the _____ input (past present, or future).
2. A memory circuit output depends on _____.

## 1-10 DIGITAL COMPUTERS

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Name the functional blocks of any computer.
- Name the two blocks that make up a central processing unit.
- Explain the primary strategy of improving performance/capability of computers.

Digital techniques have found their way into innumerable areas of technology, but the area of automatic **digital computers** is by far the most notable and most extensive. In simplest terms, *a computer is a system of hardware that performs arithmetic operations, manipulates data (usually in binary form), and makes decisions.*

For the most part, human beings can do whatever computers can do, but computers can do it with much greater speed and accuracy, in spite of the fact that computers perform all their calculations and operations one step at a time. For example, a person walking across the room does not think about which muscle to contract and which muscle to relax, or which direction to go, or even notice the obstacles. Our brains are processing this information all the time in parallel. A computer-controlled robot would need to take in data from sensor 1, then process this information, then output a command to an actuator that moves in a certain way. Then it would have to repeat this process for many other sensors and actuators. Of course, the fact that the computer requires only a few nanoseconds per step makes up for this apparent inefficiency.

A computer is faster and more accurate than people are, but unlike most of us, it must be given a complete set of instructions that tell it *exactly* what to do at each step of its operation. This set of instructions, called a **program**, is prepared by one or more persons for each job the computer is to do. Programs are placed in the computer's memory unit in binary-coded form, with each instruction having a unique code. The computer takes these instruction codes from memory *one at a time* and performs the operation called for by the code.

### Major Parts of a Computer

There are several types of computer systems, but each can be broken down into the same functional units. Each unit performs specific functions, and all units function together to carry out the instructions given in the program. Figure 1-21 shows the five major functional parts of a digital computer and their interaction. The solid lines with arrows represent the flow of data and information. The dashed lines with arrows represent the flow of timing and control signals.

The major functions of each unit are as follows:

1. **Input unit.** Through this unit, a complete set of instructions and data is fed into the computer system and into the memory unit, to be stored until needed. The information typically enters the input unit from a keyboard, a disk, or various sensors (in the case of a process control computer).
2. **Memory unit.** The memory stores the instructions and data received from the input unit. It stores the results of arithmetic operations received from the arithmetic unit. It also supplies information to the output unit.

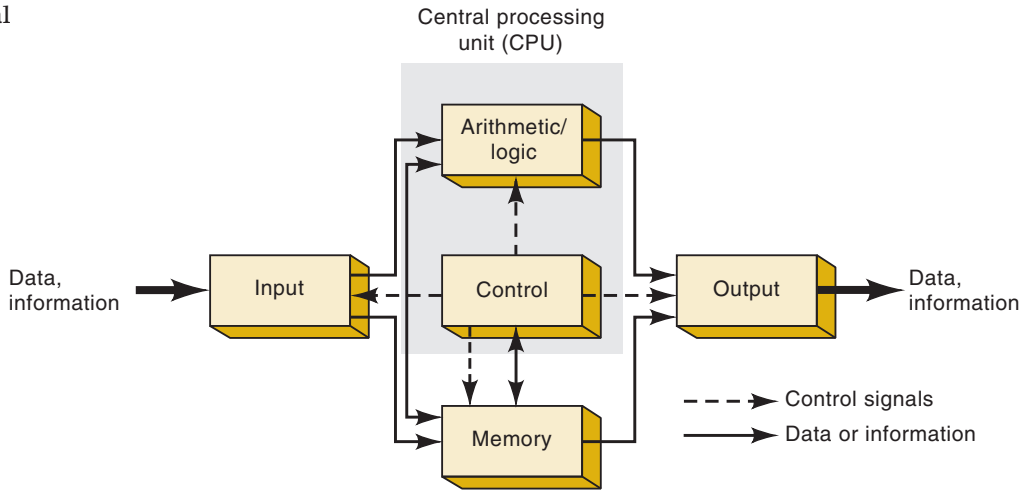



**FIGURE 1-21** Functional diagram of a digital computer.

3. **Control unit.** This unit takes instructions from the memory unit one at a time and interprets them. It then sends appropriate signals to all the other units to cause the specific instruction to be executed.
4. **Arithmetic/logic unit.** All arithmetic calculations and logical decisions are performed in this unit, which can then send results to the memory unit to be stored.
5. **Output unit.** This unit takes data from the memory unit and prints out, displays, or otherwise presents the information to the operator (or process, in the case of a process control computer).

As the diagram in Figure 1-21 shows, the control and arithmetic/logic units are often considered as one unit, called the **central processing unit (CPU)**. The CPU contains all of the circuitry for fetching and interpreting instructions and for controlling and performing the various operations called for by the instructions.

## Types of Computers

There are many ways to categorize computers and many names for different types of computers in each category have evolved. Some old terms are often improperly used to describe new configurations and some new terms are little more than another name for older configurations. We will describe three categories of computers and offer some names of types of computers in each category. The important thing to remember is that all of these systems in all of these categories can be broken down into the five major parts that we have described. These parts are simply arranged differently to optimize the system for a particular purpose.

The central processing unit of the first generation of computers was made up of lots of digital circuits distributed over many circuit boards. That is why in the 1970s when all of the parts of a central processing unit were "integrated" into one fairly small chip, that they were given the name "microprocessor." These microprocessors were combined with memory, input and output circuits to produce "microcomputers." With advances in integrated circuit technology, more and more digital circuits could fit into smaller and smaller packages and manufacturers started to offer microcomputers surrounded by specialized support hardware to make it easy to control things—all in a single integrated circuit. These came to be known as microcontrollers.

**HIGH-END COMPUTERS** The high end of computers are the very powerful systems that can handle lots of tasks and produce results very quickly. Some of the names that are often given to these computer systems are *supercomputers*, *clusters*, *servers*, and *mainframes*. Today all of these systems obtain their high speed and large throughput by using a simple strategy of dividing up the work. The overall computer is made up of many powerful microprocessors with local resources (memory, input, output) plus resources of memory and the inputs and outputs that are shared. In other words, they are made up of many computers working together to produce the intended results.

**PERSONAL COMPUTERS** The way we will define personal computer is a general-purpose computer that can run many different applications and is intended for use by individual people. Workstations, desktops, laptops, notebooks, tablets, and even cell phones fall into this category. They have a single processor, though it may be made up of multiple "cores." Each core is really a CPU or processor that shares the instruction que called the cache. The cores work together to grab (fetch) the next instruction and execute it as efficiently as possible.

**EMBEDDED COMPUTERS** In spite of all the high-end and personal computers that you see everywhere, the category that claims the most computer applications is the embedded computer market. These are the single chip computers which also contain built-in digital hardware to help it efficiently control things and communicate with other devices. These other devices include analog-to-digital and digital-to-analog converters, pulse width modulators, timer/counters, and serial interface circuits. These computers are embedded in so many commercial products that it is almost easier to name the products that do not contain a microcontroller. You have probably never seen one on a mousetrap but if someone ever builds a better mousetrap, it will probably include a microcontroller.

The unique thing about embedded microcontrollers is that they are an integral part of the inner workings of a system. They are not seen, though they are usually at the center of every function in the system. The designer gives an embedded controller one program of instructions and it is expected to run that program for the rest of their life.

## Memory

The fundamental purpose of all memory devices is to store a group of 1s and 0s. This fact begs two questions: How many 1s and 0s are in a group? How many groups can be stored in the memory device? Chapter 12 will explain all the different configurations of memory devices and help you to decide which size/shape will meet your needs. The goal for all memory devices (and most other electronic components) is to make it smaller, lower power, faster, and less expensive. Using different technologies, we can optimize some of these features, but no single technology is best at all of them. Consequently, computer systems use a combination of memory technologies. For example, many computers still use mechanical hard drives for long-term storage. This technology stores 1s and 0s (data) as magnetic fields on a rotating disk. The newer solid-state hard drives store data using Flash technology on special transistors. The data on these devices must be remembered, even if power is removed from the storage device. The working memory of your computer where the apps are accessed when they are active is made from dynamic RAM technology which stores data on capacitors. The working memory must store a very large number of bits. The video memory must be very

fast. However, it is not a problem if working memory and video memory lose their data whenever power is turned off.

To understand memory terms, think of how a typical dormitory or hotel is laid out. Each floor has the same number of rooms and each room has a number on the door. This number (usually referred to as the address) is used to locate one particular room out of all the rooms in the dormitory. The lower order digits of the room number identify where it is located within a given floor and the higher order digit identifies the floor. The number of people, characteristics of people, and assortment of stuff is unique in each room. In like manner, memory systems have an array of places to store information. These are called memory locations and they are identified with an address. The address will have upper digits that specify a general area in the device along with lower digits that identify a particular location in that area. The contents of any given memory location will be a binary number referred to as data. The number of digits that can be stored will be the same in each location, but the value of the data stored there can be unique. Chapter 12 will have much more to say about the many types of memory devices.

## Digital Progress Today and Tomorrow

Why should the contents of this book and the subject of digital systems matter to you? To answer that question, just think about the inventions that have changed how we do things since the year 2000. If you cannot think of ten examples, use the internet to look for inventions of the twenty-first century. As you look at lists generated by lots of people, ask yourself one simple question: Is any part of this invention a digital system? In almost every instance, the answer is YES! If you want to be a part of great innovations of the next 50 years, knowledge of digital systems will help you. The building blocks of digital systems have been known and understood for decades. As technology makes those building blocks faster, smaller, lower power, and less expensive, you will be able to find new ways to use them to solve the world's problems.

**OUTCOME ASSESSMENT QUESTIONS**

1. Name the five major functional units of a computer.
2. Which two units make up the CPU?
3. What is the primary strategy of improving the capabilities, speed, and throughput of computer systems?
4. What category boasts the largest number of computer applications?
5. If you discover the greatest invention of the twenty-first century, what technology will almost certainly be involved?

## SUMMARY

1. The two basic ways of representing the numerical value of physical quantities are analog (continuous) and digital (discrete).
2. Most quantities in the real world are analog, but digital techniques are generally superior to analog techniques, and most of the predicted advances will be in the digital realm.
3. The binary number system (0 and 1) is the basic system used in digital technology.
4. Digital or logic circuits operate on voltages that fall in prescribed ranges that represent either a binary 0 or a binary 1.

5. The two basic ways to transfer digital information are parallel—all bits simultaneously—and serial—one bit at a time.
6. The main parts of all computers are the input, control, memory, arithmetic/logic, and output units.
7. The combination of the arithmetic/logic unit and the control unit makes up the CPU (central processing unit).
8. General-purpose computer systems today are made up of many CPU cores (microprocessors) that work together.
9. A microcontroller is a microcomputer especially designed for dedicated (not general-purpose) control applications.

## IMPORTANT TERMS†

binary digit
bit
logic states
logic levels
timing diagram
edge
negative edge
falling edge
aperiodic
periodic
frequency (*F*)
period (*T*)
duty cycle
event
level triggered
edge triggered
logic circuits
CMOS
TTL
analog representation
digital representation
digital system
analog system
analog-to-digital converter (ADC)
digital-to-analog converter (DAC)
hexadecimal system
decimal system
binary system
parallel transmission
serial transmission
memory
digital computer
input unit
memory unit
control unit
arithmetic/logic unit
output unit
central processing unit (CPU)
microprocessor
microcomputer
microcontroller

## PROBLEMS

### SECTION 1-1

1-1. In Figure 1-22a, what is the logic level that must be output to turn on the LED?

1-2. In Figure 1-22b, what is the logic level that must be output to turn on the LED?

1-3. In Figure 1-22a, which switch must be closed to turn on the LED?

1-4. In Figure 1-22b, which switch must be closed to turn on the LED?

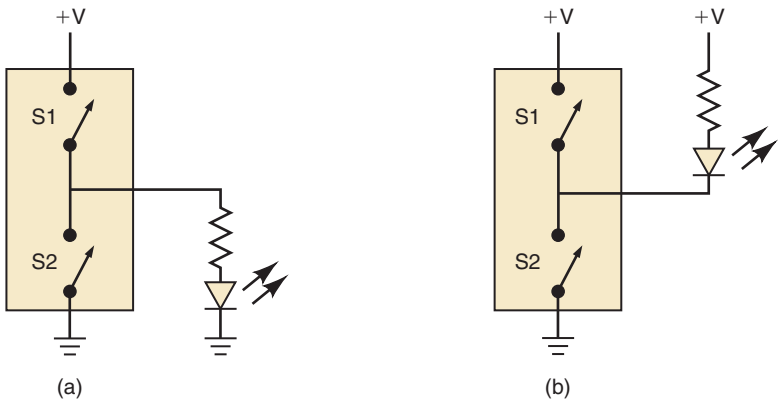



**FIGURE 1-22** Problem 1-1.

†These terms can be found in **boldface** type in the chapter and are defined in the Glossary at the end of the book. This applies to all chapters.

## SECTIONS 1-2 AND 1-3

1-5. Create a good label (name) for each signal described below:

(a)*A sensor outputs a LOW when the elevator door is closed

(b) A streetlight sensor outputs a HIGH when it detects daylight

(c)*A passenger seat sensor goes LOW when the seat is empty

(d) A temperature sensor goes HIGH when the radiator fluid is critically hot

1-6.* Draw two cycles of the timing diagram for a digital signal that continuously alternates between 0.2 V (binary 0) for 2 ms and 4.4 V (binary 1) for 4 ms.

1-7.* Measure the period, frequency, and duty cycle of the waveform in 1-6. Assume the active part of the waveform is HIGH.

1-8. An input signal alternates between LOW for twos and HIGH for threes. The output signal begins with LOW and changes state at the rising edge of the input signal. Draw their timing diagrams.

1-9. With reference to the waveforms in 1-8, measure the active-HIGH duty cycle of the input signal and the active-LOW duty cycle of the output signal.

1-10. Label each waveform in Figure 1-23: periodic/aperiodic. For those that are periodic, measure *T*, *F*, and duty cycle (assume active HIGH).

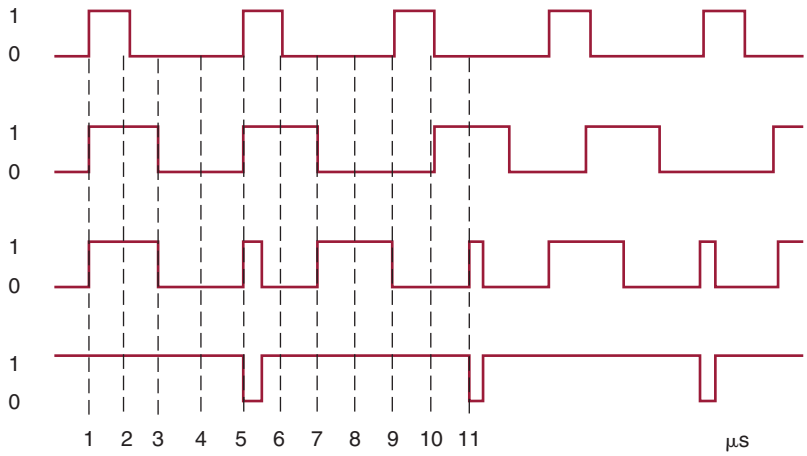



**FIGURE 1-23** Problem 1-10.

## SECTIONS 1-4 AND 1-5

1-11.* Which of the following are analog quantities, and which are digital?

(a) Number of atoms in a sample of material

(b) Altitude of an aircraft

(c) Pressure in a bicycle tire

(d) Current through a speaker

(e) Your age measured in years

* Answers to problems marked with an asterisk can be found in the back of the text.

1-12. Which of the following are analog quantities, and which are digital?
   (a) Atmospheric pressure
   (b) The number of pens in a box
   (c) The amount of milk in a glass
   (d) The volume level of a bird's chirp in the woods
   (e) The volume level of a song played on your smartphone

### SECTION 1-6

1-13.* Convert the following binary numbers to their equivalent decimal values.
   (a) $11001_2$
   (b) $1001.1001_2$
   (c) $10011011001.10110_2$

1-14. Convert the following binary numbers to decimal.
   (a) $10110_2$
   (b) $1011.0011_2$
   (c) $11001100101.10011_2$

1-15.* Using three bits, show the binary counting sequence from 000 to 111.

1-16. How many bits will be required to count the binary equivalent of 0 to 31? Show the binary counting sequence.

1-17.* What is the maximum number that we can count up to using 10 bits?

1-18. What is the maximum number that we can count up to using 12 bits?

1-19.* How many bits are needed to count up to a maximum of 511?

1-20. How many bits are needed to count up to a maximum of 255?

### SECTION 1-7

1-21. With respect to the sampled temperatures in Figure 1-18 (b) and (c), provide the following:
   (a) Sampling intervals and sampling rates from both figures
   (b) The sampled temperature that has fewer quantization errors
   (c) A method to increase sampling accuracy while maintaining the same sampling rate

### SECTION 1-8

1-22.* Suppose that the decimal integer values from 0 to 15 are to be transmitted in binary.
   (a) How many lines will be needed if parallel representation is used?
   (b) How many lines will be needed if serial representation is used?

## ANSWERS TO OUTCOME ASSESSMENT QUESTIONS

### SECTION 1-1

1. 0 and 1  2. LOW and HIGH  3. Bit  4. 0  5. It depends on the technology of the system.  6. HIGH  7. LOW  8. 2.0 V  9. 0.8 V  10. Invalid

## SECTION 1-2

1. See Figure 1-24 2. Aperiodic 3. (a) Yes (b) 0.004 sec. (c) 25% (d) 250 Hz (e) Falling edge (f) 0.008 sec. (g) 125 Hz.

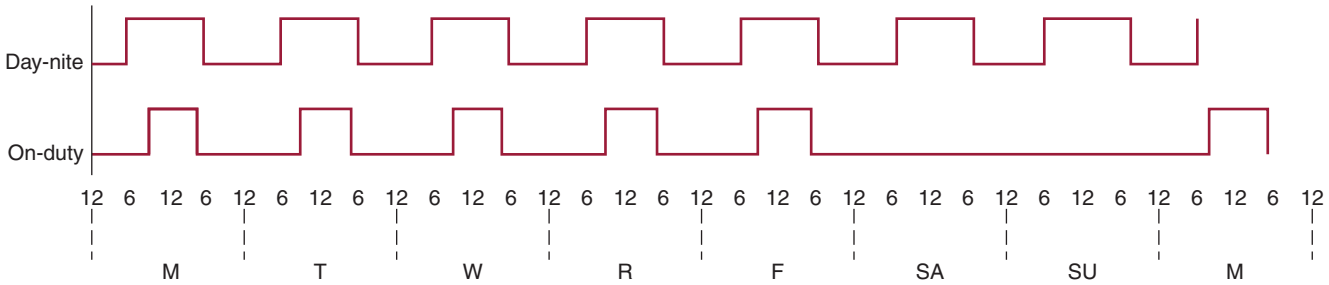



**FIGURE 1-24** Section 1-2, outcome assessment 1-1.

## SECTION 1-3

1. False 2. YES 3. Logic 4. CMOS 5. Complementary Metal-Oxide Semiconductor 6. TTL 7. Bipolar

## SECTION 1-4

1. Digital 2. Analog 3. (a) Analog, (b) Digital, (c) Digital, (d) Analog, (e) Digital, (f) Analog.

## SECTION 1-5

1. Easier to design; easier to store information; greater accuracy and precision; programmability; less affected by noise; higher degree of integration 2. Real-world physical quantities are analog. Digital processing takes time.

## SECTION 1-6

1. $107_{10}$ 2. $11000_2$ 3. $4095_{10}$

## SECTION 1-7

1. A sequence of binary numbers, representing the signal's value measured at regular intervals

## SECTION 1-8

1. Parallel is faster; serial requires only one signal line.

## SECTION 1-9

1. present 2. past outputs and present inputs.

## SECTION 1-10

1. Input, output, memory, arithmetic/logic, control 2. Control and arithmetic/logic

CHAPTER 2

# NUMBER SYSTEMS AND CODES

## OUTLINE

## CHAPTER OUTCOMES

*Upon completion of this chapter, you will be able to:*

- Convert a number from one number system (decimal, binary, hexadecimal) to its equivalent in one of the other number systems.
- Cite the advantages of the hexadecimal number system.
- Count in hexadecimal.
- Represent decimal numbers using the BCD code; cite the pros and cons of using BCD.
- Explain the difference between BCD and straight binary.
- Explain the purpose of alphanumeric codes such as the ASCII code.
- Explain the parity method for error detection.
- Determine the parity bit to be attached to a digital data string.

## INTRODUCTION

The binary number system is the most important one in digital systems, but several others are also important. The decimal system is important because it is universally used to represent quantities outside a digital system. This means that there will be situations where decimal values must be converted to binary values before they are entered into the digital system. For example, when you punch a decimal number into your hand calculator (or computer), the circuitry inside the machine converts the decimal number to a binary value.

Likewise, there will be situations where the binary values at the outputs of a digital system must be converted to decimal values for presentation to the outside world. For example, your calculator (or computer) uses binary numbers to calculate answers to a problem and then converts the answers to decimal digits before displaying them.

As you will see, it is not easy to simply look at a large binary number and convert it to its equivalent decimal value. It is very tedious to enter a long sequence of 1s and 0s on a keypad, or to write large binary numbers on a piece of paper. It is especially difficult to try to convey a binary quantity while speaking to someone. The hexadecimal (base-16) number system has become a standard way of communicating numeric values in digital systems. The great advantage is that hexadecimal numbers can be converted easily to and from binary. You will find that many advanced computer tools, which are designed to help software developers troubleshoot or debug their programs, use the hexadecimal number system to enter numbers that are stored in the computer as binary and display them again as hexadecimal.

Other methods of representing decimal quantities with binary-encoded digits have been devised that are not truly number systems but offer the ease of conversion between the binary code and the decimal number system. This is referred to as binary-coded decimal. Quantities and patterns of bits might be represented by any of these methods in any given system

and throughout the written material that supports the system, so it is very important that you are able to interpret values in any system and convert between any of these numeric representations. Other codes that use 1s and 0s to represent things such as alphanumeric characters will be covered because they are so common in digital systems.

## 2-1 BINARY-TO-DECIMAL CONVERSIONS

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Convert binary numbers to decimal.
- Identify the weight of each bit in a binary number.

As explained in Chapter 1, the binary number system is a positional system where each binary digit (bit) carries a certain weight based on its position relative to the LSB. Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number that contain a 1. To illustrate, let's change $11011_2$ to its decimal equivalent.

$$\begin{array}{l} 1 \quad 1 \quad 0 \quad 1 \quad 1_2 \\ 2^4 + 2^3 + 0 + 2^1 + 2^0 = 16 + 8 + 2 + 1 \\ \qquad\qquad\qquad\quad = 27_{10} \end{array}$$

Let's try another example with a greater number of bits:

$$\begin{array}{l} 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1_2 = \\ 2^7 + 0 + 2^5 + 2^4 + 0 + 2^2 + 0 + 2^0 = 181_{10} \end{array}$$

Note that the procedure is to find the weights (i.e., powers of 2) for each bit position that contains a 1, and then to add them up. Also note that the MSB has a weight of $2^7$ even though it is the eighth bit; this is because the LSB is the first bit and has a weight of $2^0$.

Another method of binary-to-decimal conversion that avoids the addition of large numbers and keeping track of column weights is called the double-dabble method. The procedure is as follows:

1. Write down the left-most 1 in the binary number.
2. Double it and add the next bit to the right.
3. Write down the result under the next bit.
4. Continue with steps 2 and 3 until finished with the binary number.

Let's use the same binary numbers to verify this method.

Given: 1 1 0 1 $1_2$

Results:

$$\begin{array}{l} 1 \times 2 = 2 \\ \quad +1 \\ \hline 3 \times 2 = 6 \\ \quad +0 \\ \hline 6 \times 2 = 12 \\ \quad +1 \\ \hline 13 \times 2 = 26 \\ \quad +1 \\ \hline 27_{10} \end{array}$$

| Given: | 1 | 0 | 1 | 1 | 0 | 1 | 0 | $1_2$ |
|---|---|---|---|---|---|---|---|---|
| Results: | 1 → | 2 → | 5 → | 11 → | 22 → | 45 → | 90 → | $\mathbf{181}_{10}$ |

**OUTCOME ASSESSMENT QUESTIONS**

1. Convert $100011011011_2$ to its decimal equivalent by adding the products of digits and weights.
2. What is the weight of the MSB of a 16-bit number?
3. Repeat the conversion in Question 1 using the double-dabble method.

## 2-2 DECIMAL-TO-BINARY CONVERSIONS

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Convert decimal numbers to binary.
- Identify the number of bits needed for a given range of values.
- Identify the range of values given the number of bits.

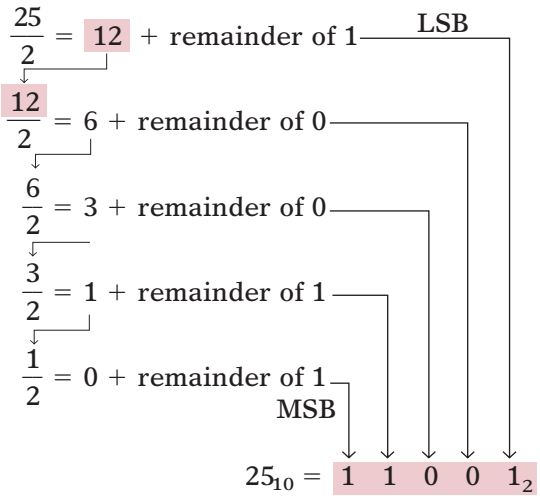
There are two ways to convert a decimal *whole* number to its equivalent binary-system representation. The first method is the reverse of the first process described in Section 2-1. The decimal number is simply expressed as a sum of powers of 2, and then 1s and 0s are written in the appropriate bit positions. To illustrate:

$$\begin{aligned} 45_{10} = 32 + 8 + 4 + 1 &= 2^5 + 0 + 2^3 + 2^2 + 0 + 2^0 \\ &= 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1_2 \end{aligned}$$

Note that a 0 is placed in the $2^1$ and $2^4$ positions, since all positions must be accounted for. Another example is the following:

$$\begin{aligned} 76_{10} = 64 + 8 + 4 &= 2^6 + 0 + 0 + 2^3 + 2^2 + 0 + 0 \\ &= 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0_2 \end{aligned}$$

Another method for converting decimal integers uses repeated division by 2. The conversion, illustrated below for $25_{10}$, requires repeatedly dividing the decimal number by 2 and writing down the remainder after each division until a quotient of 0 is obtained. Note that the binary result is obtained by writing the first remainder as the LSB and the last remainder as the MSB. This process, diagrammed in the flowchart of Figure 2-1, can also be used to convert from decimal to any other number system, as we shall see.



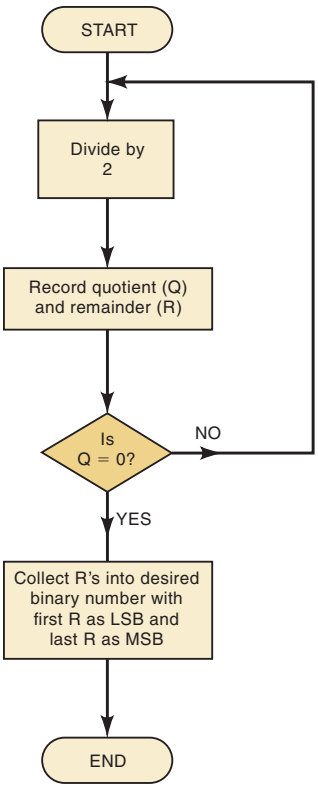



**FIGURE 2-1** Flowchart for repeated-division method of decimal-to-binary conversion of integers. The same process can be used to convert a decimal integer to any other number system.

## CALCULATOR HINT:

If you use a calculator to perform the divisions by 2, you can tell whether the remainder is 0 or 1 by whether or not the result has a fractional part. For instance, $\frac{25}{2}$ would produce 12.5. Since there is a fractional part (the .5), the remainder is a 1. If there were no fractional part, such as $\frac{12}{2} = 6$, then the remainder would be 0. Example 2-1 illustrates this.

**EXAMPLE 2-1**

Convert $37_{10}$ to binary. Try to do it on your own before you look at the solution.

**Solution**

$$\frac{37}{2} = 18.5 \longrightarrow \text{remainder of 1 (LSB)}$$

$$\frac{18}{2} = 9.0 \longrightarrow 0$$

$$\frac{9}{2} = 4.5 \longrightarrow 1$$

$$\frac{4}{2} = 2.0 \longrightarrow 0$$

$$\frac{2}{2} = 1.0 \longrightarrow 0$$

$$\frac{1}{2} = 0.5 \longrightarrow 1 \text{ (MSB)}$$

Thus, $37^{10} = \mathbf{100101_2}$.

## Counting Range

Recall that using $N$ bits, we can count through $2^N$ different decimal numbers ranging from 0 to $2^N - 1$. For example, for $N = 4$, we can count from $0000_2$ to $1111_2$, which is $0_{10}$ to $15_{10}$, for a total of 16 different numbers. Here, the largest decimal value is $2^4 - 1 = 15$, and there are $2^4$ different numbers.

In general, then, we can state:

**Using $N$ bits, we can represent decimal numbers ranging from 0 to $2^N - 1$, a total of $2^N$ different numbers.**

**EXAMPLE 2-2**

(a) What is the total range of decimal values that can be represented in eight bits?

(b) How many bits are needed to represent decimal values ranging from 0 to 12,500?

**Solution**

(a) Here we have $N = 8$. Thus, we can represent decimal numbers from 0 to $2^8 - 1 = 255$. We can verify this by checking to see that $11111111_2$ converts to $255_{10}$.

(b) With 13 bits, we can count from decimal 0 to $2^{13} - 1 = 8191$. With 14 bits, we can count from 0 to $2^{14} - 1 = 16{,}383$. Clearly, 13 bits aren't enough, but 14 bits will get us up beyond 12,500. Thus, the required number of bits is **14**.

**OUTCOME ASSESSMENT QUESTIONS**

1. Convert $83_{10}$ to binary using both methods.
2. Convert $729_{10}$ to binary using both methods. Check your answer by converting back to decimal.
3. How many bits are required to count up to decimal 1 million?

# 2-3 HEXADECIMAL NUMBER SYSTEM

## OUTCOMES

*Upon completion of this section, you will be able to:*

- Identify the weight of each hexadecimal digit.
- Convert between any of the following number systems: binary, decimal, hexadecimal.
- Count in hexadecimal.
- Identify range of numbers (in all systems) for a given number of digits.
- Identify number of digits needed for a given range of values.
- Memorize the value of each hexadecimal digit in binary and decimal.
- Cite advantages of the hexadecimal number system.

The **hexadecimal number system** uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F

as the 16 digit symbols. The digit positions are weighted as powers of 16 as shown below, rather than as powers of 10 as in the decimal system.

| $16^4$ | $16^3$ | $16^2$ | $16^1$ | $16^0$ | $16^{-1}$ | $16^{-2}$ | $16^{-3}$ | $16^{-4}$ |
|---|---|---|---|---|---|---|---|---|

Hexadecimal point

Table 2-1 shows the relationships among hexadecimal, decimal, and binary. Note that each hexadecimal digit represents a group of four binary digits. It is important to remember that hex (abbreviation for "hexadecimal") digits A through F are equivalent to the decimal values 10 through 15.

**TABLE 2-1**

| Hexadecimal | Decimal | Binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

## Hex-to-Decimal Conversion

A hex number can be converted to its decimal equivalent by using the fact that each hex digit position has a weight that is a power of 16. The LSD has a weight of $16^0 = 1$; the next higher digit position has a weight of $16^1 = 16$; the next has a weight of $16^2 = 256$; and so on. The conversion process is demonstrated in the examples below.

**CALCULATOR HINT:**

You can use the $y^x$ calculator function to evaluate the powers of 16.

$$\begin{aligned} 356_{16} &= 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0 \\ &= 768 + 80 + 6 \\ &= 854_{10} \end{aligned}$$

$$\begin{aligned} 2AF_{16} &= 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 \\ &= 512 + 160 + 15 \\ &= 687_{10} \end{aligned}$$

Note that in the second example, the value 10 was substituted for A and the value 15 for F in the conversion to decimal.

For practice, verify that $1BC2_{16}$ is equal to $7106_{10}$.

## Decimal-to-Hex Conversion

Recall that we did decimal-to-binary conversion using repeated division by 2. Likewise, decimal-to-hex conversion can be done using repeated division by 16 (Figure 2-1). The following example contains two illustrations of this conversion.

**EXAMPLE 2-3**

(a) Convert $423_{10}$ to hex.

**Solution**

$$\frac{423}{16} = 26 + \text{remainder of } 7$$

$$\frac{26}{16} = 1 + \text{remainder of } 10$$

$$\frac{1}{16} = 0 + \text{remainder of } 1$$

$$423_{10} = 1A7_{16}$$

(b) Convert $214_{10}$ to hex.

**Solution**

$$\frac{214}{16} = 13 + \text{remainder of } 6$$

$$\frac{13}{16} = 0 + \text{remainder of } 13$$

$$214_{10} = D6_{16}$$

Again note that the remainders of the division processes form the digits of the hex number. Also note that any remainders that are greater than 9 are represented by the letters A through F.

**CALCULATOR HINT:**

If a calculator is used to perform the divisions in the conversion process, the results will include a decimal fraction instead of a remainder. The remainder can be obtained by multiplying the fraction by 16. To illustrate, in Example 2-3(b), the calculator would have produced

$$\frac{214}{16} = 13.375$$

The remainder becomes $(0.375) \times 16 = 6$.

## Hex-to-Binary Conversion

The hexadecimal number system is used primarily as a "shorthand" method for representing binary numbers. It is a relatively simple matter to convert

a hex number to binary. *Each* hex digit is converted to its four-bit binary equivalent (Table 2-1). This is illustrated below for $9F2_{16}$.

$$\begin{aligned} 9F2_{16} &= \quad 9 \qquad\qquad F \qquad\qquad 2 \\ &\qquad\quad \downarrow \qquad\quad \downarrow \qquad\quad \downarrow \\ &= 1\ 0\ 0\ 1 \quad 1\ 1\ 1\ 1 \quad 0\ 0\ 1\ 0 \\ &= 100111110010_2 \end{aligned}$$

For practice, verify that $BA6_{16} = 101110100110_2$.

## Binary-to-Hex Conversion

Conversion from binary to hex is just the reverse of the process above. The binary number is grouped into groups of *four* bits, and each group is converted to its equivalent hex digit. Zeros (shown shaded) are added, as needed, to complete a four-bit group.

$$\begin{aligned} 1\,1\,1\,0\,1\,0\,0\,1\,1\,0_2 &= \underbrace{0\ 0\ 1\ 1}_{3}\ \underbrace{1\ 0\ 1\ 0}_{A}\ \underbrace{0\ 1\ 1\ 0}_{6} \\ &= 3A6_{16} \end{aligned}$$

To perform these conversions between hex and binary, it is necessary to know the four-bit binary numbers (0000 through 1111) and their equivalent hex digits. Once these are mastered, the conversions can be performed quickly without the need for any calculations. This is why hex is so useful in representing large binary numbers.

For practice, verify that $101011111_2 = 15F_{16}$.

## Counting in Hexadecimal

When counting in hex, each digit position can be incremented (increased by 1) from 0 to F. Once a digit position reaches the value F, it is reset to 0, and the next digit position is incremented. This is illustrated in the following hex counting sequences:

(a) 38, 39, 3A, 3B, 3C, 3D, 3E, 3F, 40, 41, 42
(b) 6F8, 6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700

Note that when there is a 9 in a digit position, it becomes an A when it is incremented.

With $N$ hex digit positions, we can count from decimal 0 to $16^N - 1$, for a total of $16^N$ different values. For example, with three hex digits, we can count from $000_{16}$ to $FFF_{16}$, which is $0_{10}$ to $4095_{10}$, for a total of $4096 = 16^3$ different values.

## Usefulness of Hex

Hex is often used in a digital system as sort of a "shorthand" way to represent strings of bits. In computer work, strings as long as 64 bits are not uncommon. These binary strings do not always represent a numerical value, but—as you will find out—can be some type of code that conveys nonnumerical information. When dealing with a large number of bits, it is more convenient and less error-prone to write the binary numbers in hex and, as we have seen, it is relatively easy to convert back and forth between binary and hex. To illustrate the advantage of hex representation of a

binary string, suppose you had in front of you a printout of the contents of 50 memory locations, each of which was a 16-bit number, and you were checking it against a list. Would you rather check 50 numbers like this one: 0110111001100111, or 50 numbers like this one: 6E67? And which one would you be more apt to read incorrectly? It is important to keep in mind, though, that digital circuits all work in binary. Hex is simply used as a convenience for the humans involved. You should memorize the four-bit binary pattern for each hexadecimal digit. Only then will you realize the usefulness of this tool in digital systems.

**EXAMPLE 2-4**

Convert decimal 378 to a 16-bit binary number by first converting to hexadecimal.

**Solution**

$$\frac{378}{16} = 23 + \text{remainder of } 10_{10} = A_{16}$$

$$\frac{23}{16} = 1 + \text{remainder of } 7$$

$$\frac{1}{16} = 0 + \text{remainder of } 1$$

Thus, $378_{10} = 17A_{16}$. This hex value can be converted easily to binary 000101111010. Finally, we can express $378_{10}$ as a 16-bit number by adding four leading 0s:

$$378_{10} = 0000 \quad 0001 \quad 0111 \quad 1010_2$$

**EXAMPLE 2-5**

Convert $B2F_{16}$ to decimal.

**Solution**

$$\begin{aligned} B2F_{16} &= B \times 16^2 + 2 \times 16^1 + F \times 16^0 \\ &= 11 \times 256 + 2 \times 16 + 15 \\ &= 2863_{10} \end{aligned}$$

## Summary of Conversions

Right now, your head is probably spinning as you try to keep straight all of these different conversions from one number system to another. You probably realize that many of these conversions can be done *automatically* on your calculator just by pressing a key, but it is important for you to master these conversions so that you understand the process. Besides, what happens if your calculator battery dies at a crucial time and you have no handy replacement? The following summary should help you, but nothing beats practice, practice, practice!

1. When converting from binary [or hex] to decimal, use the method of taking the weighted sum of each digit position or follow the double-dabble procedure.
2. When converting from decimal to binary [or hex], use the method of repeatedly dividing by 2 [or 16] and collecting remainders (Figure 2-1).

3. When converting from binary to hex, group the bits in groups of four, and convert each group into the correct hex digit.
4. When converting from hex to binary, convert each digit into its four-bit equivalent.

**OUTCOME ASSESSMENT QUESTIONS**

1. Convert $24CE_{16}$ to decimal.
2. Convert $3117_{10}$ to hex, then from hex to binary.
3. Convert $1001011110110101_2$ to hex.
4. Write the next four numbers in this hex counting sequence: E9A, E9B, E9C, E9D, _____, _____, _____, _____.
5. Convert $3527_{16}$ to binary.
6. What range of decimal values can be represented by a four-digit hex number?

## 2-4 BCD CODE

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Convert decimal numbers to BCD code.
- Convert BCD code to decimal.
- Cite the pros and cons of using BCD.
- Cite advantages/disadvantages of BCD versus binary in digital systems.

When numbers, letters, or words are represented by a special group of symbols, we say that they are being encoded, and the group of symbols is called a *code.* Probably one of the most familiar codes is the Morse code, where a series of dots and dashes represents letters of the alphabet.

We have seen that any decimal number can be represented by an equivalent binary number. The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number. When a decimal number is represented by its equivalent binary number, we call it **straight binary coding.**

Digital systems all use some form of binary numbers for their internal operation, but the external world is decimal in nature. This means that conversions between the decimal and binary systems are being performed often. We have seen that the conversions between decimal and binary can become long and complicated for large numbers. For this reason, a means of encoding decimal numbers that combines some features of both the decimal and the binary systems is used in certain situations.

### Binary-Coded-Decimal Code

If *each* digit of a decimal number is represented by its binary equivalent, the result is a code called **binary-coded decimal** (hereafter abbreviated BCD). Since a decimal digit can be as large as 9, four bits are required to code each digit (the binary code for 9 is 1001).

To illustrate the BCD code, take a decimal number such as 874. Each *digit* is changed to its binary equivalent as follows:

$$\begin{array}{cccc} 8 & 7 & 4 & \text{(decimal)} \\ \downarrow & \downarrow & \downarrow & \\ 1000 & 0111 & 0100 & \text{(BCD)} \end{array}$$

As another example, let us change 943 to its BCD-code representation:

$$\begin{array}{cccc} 9 & 4 & 3 & \text{(decimal)} \\ \downarrow & \downarrow & \downarrow & \\ 1001 & 0100 & 0011 & \text{(BCD)} \end{array}$$

Once again, each decimal digit is changed to its straight binary equivalent. Note that four bits are *always* used for each digit.

The BCD code, then, represents each digit of the decimal number by a four-bit binary number. Clearly only the four-bit binary numbers from 0000 through 1001 are used. The BCD code does not use the numbers 1010, 1011, 1100, 1101, 1110, and 1111. In other words, only 10 of the 16 possible four-bit binary code groups are used. If any of the "forbidden" four-bit numbers ever occurs in a machine using the BCD code, it is usually an indication that an error has occurred.

**EXAMPLE 2-6**

Convert 0110100000111001 (BCD) to its decimal equivalent.

**Solution**

Divide the BCD number into four-bit groups and convert each to decimal.

$$\begin{array}{cccc} \underbrace{0110} & \underbrace{1000} & \underbrace{0011} & \underbrace{1001} \\ 6 & 8 & 3 & 9 \end{array}$$

**EXAMPLE 2-7**

Convert the BCD number 011111000001 to its decimal equivalent.

**Solution**

$$\begin{array}{ccc} \underbrace{0111} & 1100 & \underbrace{0001} \\ 7 & \downarrow & 1 \end{array}$$

The forbidden code group indicates an error in the BCD number.

## Comparison of BCD and Binary

It is important to realize that BCD is not another number system like binary, decimal, and hexadecimal. In fact, it is the decimal system with each digit encoded in its binary equivalent. It is also important to understand that a BCD number is *not* the same as a straight binary number. A straight binary number takes the *complete* decimal number and represents it in binary; the BCD code converts *each* decimal *digit* to binary individually. To illustrate, take the number 137 and compare its straight binary and BCD codes:

$$137_{10} = 10001001_2 \quad \text{(binary)}$$
$$137_{10} = 0001\ 0011\ 0111 \quad \text{(BCD)}$$

The BCD code requires 12 bits, while the straight binary code requires only eight bits to represent 137. BCD requires more bits than straight binary to represent decimal numbers of more than one digit because BCD does not use all possible four-bit groups, as pointed out earlier, and is therefore somewhat inefficient.

The main advantage of the BCD code is the relative ease of converting to and from decimal. Only the four-bit code groups for the decimal digits 0 through 9 need to be remembered. This ease of conversion is especially important from a hardware standpoint because in a digital system, it is the logic circuits that perform the conversions to and from decimal.

**EXAMPLE 2-8**

An automatic teller machine (ATM) at the bank allows you to enter the amount of cash you wish to withdraw in decimal by pressing decimal digit symbol keys. Does the computer convert this decimal number into straight binary or BCD? Explain.

**Solution**

The number that represents your balance (all the money you have in the bank) is stored as a straight binary number. When the withdrawn amount is entered, it must be subtracted from the balance. Since arithmetic must be performed on the numbers, both values (the balance and the withdrawn cash) must be in straight binary. It converts the decimal entry to straight binary.

**EXAMPLE 2-9**

Your cell phone allows you to enter/store a 10-decimal-digit phone number. Does the cell phone store the phone number in straight binary or BCD? Explain.

**Solution**

A phone number is a combination of many decimal digits. It is not necessary to mathematically combine the digits (i.e., you never add two phone numbers together). The phone just needs to store them in the sequence they were entered and retrieve them one at a time when you press *send*. Therefore, they will be stored as BCD digits in the memory of the computer in your cell phone.

**OUTCOME ASSESSMENT QUESTIONS**

1. Represent the decimal value 178 by its straight binary equivalent. Then encode the same decimal number using BCD.
2. How many bits are required to represent an eight-digit decimal number in BCD?
3. What is an advantage of encoding a decimal number in BCD rather than in straight binary? What is a disadvantage?

## 2-5 THE GRAY CODE

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Cite advantage of Gray code versus binary.
- Convert between Gray code and binary values.
- Generate the Gray code sequence.

**TABLE 2-2** Three-bit binary and Gray code equivalents.

| $B_2$ | $B_1$ | $B_0$ | $G_2$ | $G_1$ | $G_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Digital systems operate at very fast speeds and respond to changes that occur in the digital inputs. Just as in life, when multiple input conditions are changing at the same time, the situation can be misinterpreted and cause an erroneous reaction. When you look at the bits in a binary count sequence, it is clear that there are often several bits that must change states at the same time. For example, consider when the three-bit binary number for 3 changes to 4: all three bits must change state.

In order to reduce the likelihood of a digital circuit misinterpreting a changing input, the **Gray code** has been developed as a way to represent a sequence of numbers. The unique aspect of the Gray code is that only one bit ever changes between two successive numbers in the sequence. Table 2-2 shows the translation between three-bit binary and Gray code values. To convert binary to Gray, simply start on the most significant bit and use it as the Gray MSB as shown in Figure 2-2(a). Now compare the MSB binary with the next binary bit (B1). If they are the same, then G1 = 0. If they are different, then G1 = 1. G0 can be found by comparing B1 with B0.

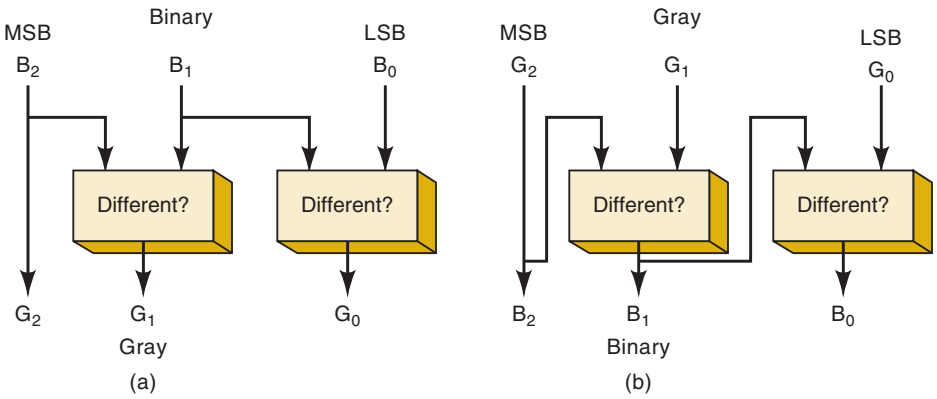



**FIGURE 2-2** Converting (a) binary to Gray and (b) Gray to binary.

Conversion from Gray code back into binary is shown in Figure 2-2(b). Note that the MSB in Gray is always the same as the MSB in binary. The next binary bit is found by comparing the *binary* bit to the left with the *corresponding Gray code bit*. Similar bits produce a 0 and differing bits produce a 1. The most common application of the Gray code is in shaft position encoders as shown in Figure 2-3. These devices produce a binary value that represents the position of a rotating mechanical shaft. A practical shaft encoder would use many more bits than just three and divide the rotation into many more segments than eight, so that it could detect much smaller increments of rotation.

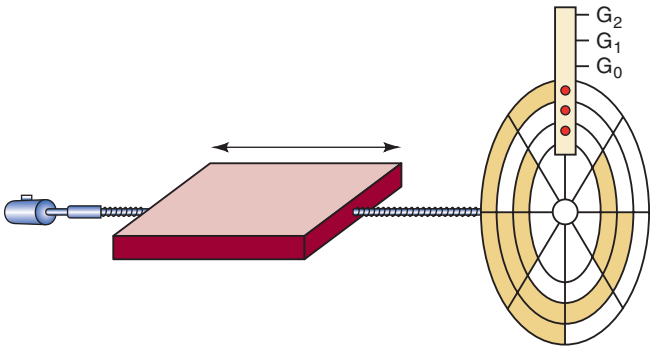



**FIGURE 2-3** An eight-position, three-bit shaft encoder.

## Quadrature Encoders

The most common application of the Gray code is the quadrature shaft encoder. As the shaft rotates, this device produces a two-bit Gray code sequence on its outputs. Clockwise rotation produces the sequence shown in Table 2-3(a) and counterclockwise rotation produces the sequence shown in Table 2-3(b). Converting these Gray code values to binary shows that they are counting up or counting down depending on the direction of rotation. The sensitivity or number of degrees of rotation represented by each state of the Gray sequence will vary between the many models of shaft encoders available. The important feature of a shaft encoder is that the *sequence* of states can be used to determine which direction the shaft is rotating.

**TABLE 2-3** The two-bit Gray code from a quadrature shaft encoder.

| Clockwise | | | | Counter Clockwise | | | |
|---|---|---|---|---|---|---|---|
| A | B | Binary | Decimal | A | B | Binary | Decimal |
| 0 | 0 | 00 | 0 | 0 | 0 | 00 | 0 |
| 0 | 1 | 01 | 1 | 1 | 0 | 11 | 3 |
| 1 | 1 | 10 | 2 | 1 | 1 | 10 | 2 |
| 1 | 0 | 11 | 3 | 0 | 1 | 01 | 1 |
| | | (a) | | | | (b) | |

Figure 2-4 shows an inexpensive shaft encoder that can be used as a control knob on consumer electronics. This knob could be a volume control, or a tuning control on a radio receiver, for example. There are three terminals on this encoder. One terminal connects to the spoked wheel. The conducting spokes on the wheel rub against two spring metal contact arms as the shaft

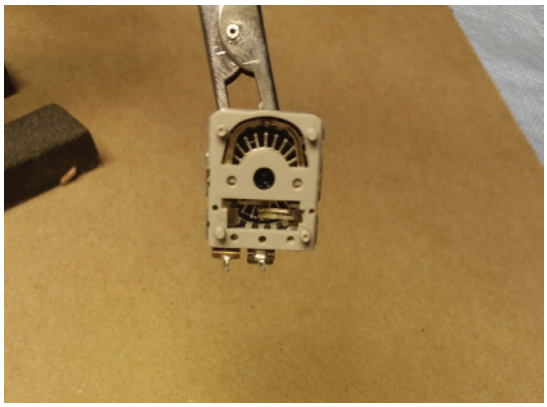


**FIGURE 2-4** A mechanical contact quadrature encoder.

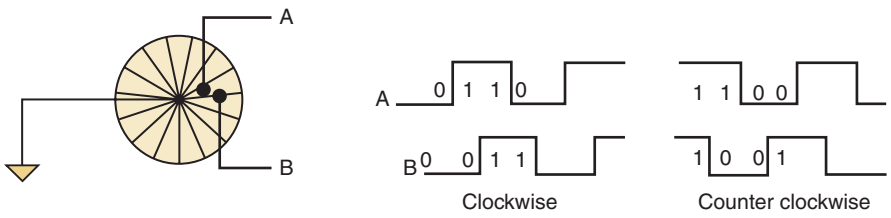



**FIGURE 2-5** Operation of a quadrature encoder.

rotates. The other two terminals are wired to the two spring metal contacts. The spring metal contacts are positioned such that one will always make contact with the spoke slightly before the other one as the shaft is rotated. With this encoder connected as shown in Figure 2-5, rotating the shaft clockwise and counter clockwise produces the waveforms shown. Notice that the states in these timing diagrams follow the two-bit Gray code sequence.

Quadrature encoders are referred to as incremental shaft encoders, while encoders that put out enough Gray code bits to uniquely identify any shaft position as shown in Figure 2-3 are referred to as absolute shaft encoders. Chapter 5 will demonstrate how to combine an incremental (quadrature) shaft encoder with a digital counter circuit to keep track of absolute shaft position.

**OUTCOME ASSESSMENT QUESTIONS**

1. Convert the number 0101 (binary) to its Gray code equivalent.
2. Convert 0101 (Gray code) to its binary number equivalent.

## 2-6 PUTTING IT ALL TOGETHER

Table 2-4 gives the representation of the decimal numbers 1 through 15 in the binary and hex number systems and also in the BCD and Gray codes. Examine it carefully and make sure you understand how it was obtained. Especially note how the BCD representation always uses four bits for each decimal digit.

**TABLE 2-4** Number system/code equivalents.

| Decimal | Binary | Hexadecimal | BCD | Gray |
|---|---|---|---|---|
| 0 | 0 | 0 | 0000 | 0000 |
| 1 | 1 | 1 | 0001 | 0001 |
| 2 | 10 | 2 | 0010 | 0011 |
| 3 | 11 | 3 | 0011 | 0010 |
| 4 | 100 | 4 | 0100 | 0110 |
| 5 | 101 | 5 | 0101 | 0111 |
| 6 | 110 | 6 | 0110 | 0101 |
| 7 | 111 | 7 | 0111 | 0100 |
| 8 | 1000 | 8 | 1000 | 1100 |
| 9 | 1001 | 9 | 1001 | 1101 |
| 10 | 1010 | A | 0001 0000 | 1111 |
| 11 | 1011 | B | 0001 0001 | 1110 |
| 12 | 1100 | C | 0001 0010 | 1010 |
| 13 | 1101 | D | 0001 0011 | 1011 |
| 14 | 1110 | E | 0001 0100 | 1001 |
| 15 | 1111 | F | 0001 0101 | 1000 |

## 2-7 THE BYTE, NIBBLE, AND WORD

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Define common terms: *byte*, *nibble*, and *word*.
- Use these words in context.
- Interpret these words in the context used.

### Bytes

Most microcomputers handle and store binary data and information in groups of eight bits, so a special name is given to a string of eight bits: it is called a **byte.** A byte always consists of eight bits, and it can represent any of numerous types of data or information. The following examples will illustrate.

**EXAMPLE 2-10**

How many bytes are in a 32-bit string (a string of 32 bits)?

**Solution**

$\frac{32}{8} = 4$, so there are **four** bytes in a 32-bit string.

**EXAMPLE 2-11**

What is the largest decimal value that can be represented in binary using two bytes?

**Solution**

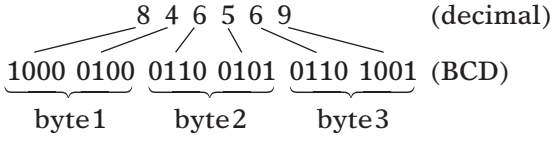
Two bytes is 16 bits, so the largest binary value will be equivalent to decimal $2^{16} - 1 = 65{,}535$.

**EXAMPLE 2-12**

How many bytes are needed to represent the decimal value 846,569 in BCD?

**Solution**

Each decimal digit converts to a four-bit BCD code. Thus, a six-digit decimal number requires 24 bits. These 24 bits are equal to **three** bytes. This is diagrammed below.




### Nibbles

Binary numbers are often broken down into groups of four bits, as we have seen with BCD codes and hexadecimal number conversions. In the early days of digital systems, a term caught on to describe a group of four bits. Because it is half as big as a byte, it was named a **nibble.** The following examples illustrate the use of this term.

**EXAMPLE 2-13**

How many nibbles are in a byte?

**Solution**

2

**EXAMPLE 2-14**

What is the hex value of the least significant nibble of the binary number 10010101?

**Solution**

1001 0101

The least significant nibble is 0101 = 5.

## Words

*Bits*, *nibbles*, and *bytes* are terms that represent a fixed number of binary digits. As systems have grown over the years, their capacity (appetite?) for handling binary data has also grown. A **word** is a group of bits that represents a certain unit of information. The size of the word depends on the size of the data pathway in the system that uses the information. The **word size** can be defined as the number of bits in the binary word that a digital system operates on. For example, the computer in your microwave oven can probably handle only one byte at a time. It has a word size of eight bits. On the other hand, the personal computer on your desk can handle eight bytes at a time, so it has a word size of 64 bits.

**OUTCOME ASSESSMENT QUESTIONS**

1. How many bytes are needed to represent $235_{10}$ in binary?
2. What is the largest decimal value that can be represented in BCD using two bytes?
3. How many hex digits can a nibble represent?
4. How many nibbles are in one BCD digit?

# 2-8 ALPHANUMERIC CODES

## OUTCOMES

*Upon completion of this section, you will be able to:*

- Use a table to translate between ASCII codes and characters.
- Explain the purpose of alphanumeric codes such as ASCII.

In addition to numerical data, a computer must be able to handle nonnumerical information. In other words, a computer should recognize codes that represent letters of the alphabet, punctuation marks, and other special characters as well as numbers. These codes are called **alphanumeric codes.** A complete alphanumeric code would include the 26 lowercase letters, 26 uppercase letters, 10 numeric digits, 7 punctuation marks, and anywhere from 20 to 40 other characters, as +, /, #, %, *, and so on. We can say that an alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.

## ASCII Code

The most widely used alphanumeric code is the **American Standard Code for Information Interchange (ASCII).** The ASCII (pronounced "askee") code is a seven-bit code, and so it has $2^7 = 128$ possible code groups. This is more than enough to represent all of the standard keyboard characters as well as the control functions such as the (RETURN) and (LINEFEED) functions. Table 2-5 shows a listing of the standard seven-bit ASCII code. The table gives the hexadecimal and decimal equivalents. The seven-bit binary code for each character can be obtained by converting the hex value to binary.

**TABLE 2-5** Standard ASCII codes.

| Character | Hex | Decimal | Character | Hex | Decimal | Character | Hex | Decimal | Character | Hex | Decimal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NUL (null) | 0 | 0 | Space | 20 | 32 | @ | 40 | 64 | . | 60 | 96 |
| Start Heading | 1 | 1 | ! | 21 | 33 | A | 41 | 65 | a | 61 | 97 |
| Start Text | 2 | 2 | " | 22 | 34 | B | 42 | 66 | b | 62 | 98 |
| End Text | 3 | 3 | # | 23 | 35 | C | 43 | 67 | c | 63 | 99 |
| End Transmit. | 4 | 4 | $ | 24 | 36 | D | 44 | 68 | d | 64 | 100 |
| Enquiry | 5 | 5 | % | 25 | 37 | E | 45 | 69 | e | 65 | 101 |
| Acknowlege | 6 | 6 | & | 26 | 38 | F | 46 | 70 | f | 66 | 102 |
| Bell | 7 | 7 | ` | 27 | 39 | G | 47 | 71 | g | 67 | 103 |
| Backspace | 8 | 8 | ( | 28 | 40 | H | 48 | 72 | h | 68 | 104 |
| Horiz. Tab | 9 | 9 | ) | 29 | 41 | I | 49 | 73 | i | 69 | 105 |
| Line Feed | A | 10 | * | 2A | 42 | J | 4A | 74 | j | 6A | 106 |
| Vert. Tab | B | 11 | + | 2B | 43 | K | 4B | 75 | k | 6B | 107 |
| Form Feed | C | 12 | , | 2C | 44 | L | 4C | 76 | l | 6C | 108 |
| Carriage Return | D | 13 | - | 2D | 45 | M | 4D | 77 | m | 6D | 109 |
| Shift Out | E | 14 | . | 2E | 46 | N | 4E | 78 | n | 6E | 110 |
| Shift In | F | 15 | / | 2F | 47 | O | 4F | 79 | o | 6F | 111 |
| Data Link Esc | 10 | 16 | 0 | 30 | 48 | P | 50 | 80 | p | 70 | 112 |
| Direct Control 1 | 11 | 17 | 1 | 31 | 49 | Q | 51 | 81 | q | 71 | 113 |
| Direct Control 2 | 12 | 18 | 2 | 32 | 50 | R | 52 | 82 | r | 72 | 114 |
| Direct Control 3 | 13 | 19 | 3 | 33 | 51 | S | 53 | 83 | s | 73 | 115 |
| Direct Control 4 | 14 | 20 | 4 | 34 | 52 | T | 54 | 84 | t | 74 | 116 |
| Negative ACK | 15 | 21 | 5 | 35 | 53 | U | 55 | 85 | u | 75 | 117 |
| Synch Idle | 16 | 22 | 6 | 36 | 54 | V | 56 | 86 | v | 76 | 118 |
| End Trans Block | 17 | 23 | 7 | 37 | 55 | W | 57 | 87 | w | 77 | 119 |
| Cancel | 18 | 24 | 8 | 38 | 56 | X | 58 | 88 | x | 78 | 120 |
| End of Medium | 19 | 25 | 9 | 39 | 57 | Y | 59 | 89 | y | 79 | 121 |
| Substitue | 1A | 26 | : | 3A | 58 | Z | 5A | 90 | z | 7A | 122 |
| Escape | 1B | 27 | ; | 3B | 59 | [ | 5B | 91 | { | 7B | 123 |
| Form Separator | 1C | 28 | < | 3C | 60 | \ | 5C | 92 | \| | 7C | 124 |
| Group Separator | 1D | 29 | = | 3D | 61 | ] | 5D | 93 | } | 7D | 125 |
| Record Separator | 1E | 30 | > | 3E | 62 | ^ | 5E | 94 | ~ | 7E | 126 |
| Unit Separator | 1F | 31 | ? | 3F | 63 | _ | 5F | 95 | Delete | 7F | 127 |

**EXAMPLE 2-15**

Use Table 2-5 to find the seven-bit ASCII code for the backslash character (\).

**Solution**

The hex value given in Table 2-5 is 5C. Translating each hex digit into four-bit binary produces 0101 1100. The lower seven bits represent the ASCII code for \, or 1011100.

The ASCII code is used for the transfer of alphanumeric information between a computer and the external devices such as a printer or another computer. A computer also uses ASCII internally to store the information that an operator types in at the computer's keyboard. The following example illustrates this.

**EXAMPLE 2-16**

An operator is typing in a C language program at the keyboard of a certain microcomputer. The computer converts each keystroke into its ASCII code and stores the code as a byte in memory. Determine the binary strings that will be entered into memory when the operator types in the following C statement:

```
if (x>3)
```

**Solution**

Locate each character (including the space) in Table 2-5 and record its ASCII code.

| | | | |
|---|---|---|---|
| i | 69 | 0110 | 1001 |
| f | 66 | 0110 | 0110 |
| space | 20 | 0010 | 0000 |
| ( | 28 | 0010 | 1000 |
| x | 78 | 0111 | 1000 |
| > | 3E | 0011 | 1110 |
| 3 | 33 | 0011 | 0011 |
| ) | 29 | 0010 | 1001 |

Note that a 0 was added to the leftmost bit of each ASCII code because the codes must be stored as bytes (eight bits). This adding of an extra bit is called *padding with 0s.*

**OUTCOME ASSESSMENT QUESTIONS**

1. Encode the following message in ASCII code using the hex representation: "COST = $72."
2. The following padded ASCII-coded message is stored in successive memory locations in a computer:

   01010011 01010100 01001111 01010000

What is the message?

## 2-9 PARITY METHOD FOR ERROR DETECTION

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Use even or odd parity schemes.
- Add the proper parity bit for either scheme.
- Explain the parity method for error detection.
- Determine if an error has occurred using either scheme.

The movement of binary data and codes from one location to another is the most frequent operation performed in digital systems. Here are just a few examples:

- The transmission of digitized voice over a microwave link
- The storage of data in and retrieval of data from external memory devices such as magnetic and optical disk
- The transmission of digital data from a computer to a remote computer over telephone lines (i.e., using a modem). This is one of the major ways of sending and receiving information on the Internet.

Whenever information is transmitted from one device (the transmitter) to another device (the receiver), there is a possibility that errors can occur such that the receiver does not receive the identical information that was sent by the transmitter. The major cause of any transmission errors is *electrical noise*, which consists of spurious fluctuations in voltage or current that are present in all electronic systems to varying degrees. Figure 2-6 is a simple illustration of a type of transmission error.

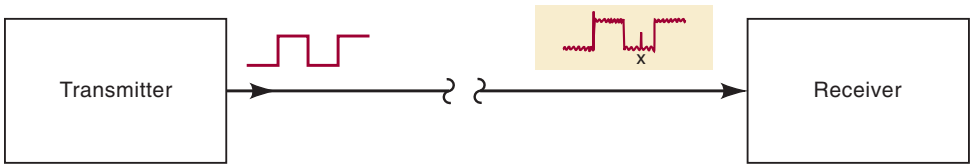



**FIGURE 2-6** Example of noise causing an error in the transmission of digital data.

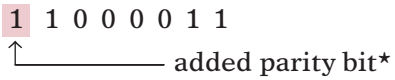
The transmitter sends a relatively noise-free serial digital signal over a signal line to a receiver. However, by the time the signal reaches the receiver, it contains a certain degree of noise superimposed on the original signal. Occasionally, the noise is large enough in amplitude that it will alter the logic level of the signal, as it does at point *x*. When this occurs, the receiver may incorrectly interpret that bit as a logic 1, which is not what the transmitter has sent.

Most modern digital equipment is designed to be relatively error-free, and the probability of errors such as the one shown in Figure 2-6 is very low. However, we must realize that digital systems often transmit thousands, even millions, of bits per second, so that even a very low rate of occurrence of errors can produce an occasional error that might prove to be bothersome, if not disastrous. For this reason, many digital systems employ some method for detection (and sometimes correction) of errors. One of the simplest and most widely used schemes for error detection is the **parity method.**

## Parity Bit

A **parity bit** is an extra bit that is attached to a code group that is being transferred from one location to another. The parity bit is made either 0 or 1, depending on the number of 1s that are contained in the code group. Two different methods are used.

In the *even-parity* method, the value of the parity bit is chosen so that the total number of 1s in the code group (including the parity bit) is an *even* number. For example, suppose that the group is 1000011. This is the ASCII character "C." The code group has *three* 1s. Therefore, we will add a parity bit of 1 to make the total number of 1s an even number. The *new* code group, *including the parity bit*, thus becomes




If the code group contains an even number of 1s to begin with, the parity bit is given a value of 0. For example, if the code group were 1000001 (the ASCII code for "A"), the assigned parity bit would be 0, so that the new code, *including the parity bit*, would be 01000001.

The *odd-parity* method is used in exactly the same way except that the parity bit is chosen so the total number of 1s (including the parity bit) is an *odd* number. For example, for the code group 1000001, the assigned parity bit would be a 1. For the code group 1000011, the parity bit would be a 0.

Regardless of whether even parity or odd parity is used, the parity bit becomes an actual part of the code word. For example, adding a parity bit to the seven-bit ASCII code produces an eight-bit code. Thus, the parity bit is treated just like any other bit in the code.

The parity bit is issued to detect any *single-bit* errors that occur during the transmission of a code from one location to another. For example, suppose that the character "A" is being transmitted and *odd* parity is being used. The transmitted code would be

1 1 0 0 0 0 0 1

When the receiver circuit receives this code, it will check that the code contains an odd number of 1s (including the parity bit). If so, the receiver will assume that the code has been correctly received. Now, suppose that because of some noise or malfunction the receiver actually receives the following code:

1 1 0 0 0 0 0 0

The receiver will find that this code has an *even* number of 1s. This tells the receiver that there must be an error in the code because presumably the transmitter and receiver have agreed to use odd parity. There is no way, however, that the receiver can tell which bit is in error because it does not know what the code is supposed to be.

It should be apparent that this parity method would not work if *two* bits were in error, because two errors would not change the "oddness" or "evenness" of the number of 1s in the code. In practice, the parity method is used only in situations where the probability of a single error is very low and the probability of double errors is essentially zero.

When the parity method is being used, the transmitter and the receiver must have agreement, in advance, as to whether odd or even parity is being

*The parity bit can be placed at either end of the code group, but it is usually placed to the left of the MSB.

used. There is no advantage of one over the other, although even parity seems to be used more often. The transmitter must attach an appropriate parity bit to each unit of information that it transmits. For example, if the transmitter is sending ASCII-coded data, it will attach a parity bit to each seven-bit ASCII code group. When the receiver examines the data that it has received from the transmitter, it checks each code group to see that the total number of 1s (including the parity bit) is consistent with the agreed-upon type of parity. This is often called *checking the parity* of the data. In the event that it detects an error, the receiver may send a message back to the transmitter asking it to retransmit the last set of data. The exact procedure that is followed when an error is detected depends on the particular system.

**EXAMPLE 2-17**

Computers often communicate with other remote computers over telephone lines. For example, this is how dial-up communication over the internet takes place. When one computer is transmitting a message to another, the information is usually encoded in ASCII. What actual bit strings would a computer transmit to send the message HELLO, using ASCII with even parity?

**Solution**

First, look up the ASCII codes for each character in the message. Then for each code, count the number of 1s. If it is an even number, attach a 0 as the MSB. If it is an odd number, attach a 1. Thus, the resulting eight-bit codes (bytes) will all have an even number of 1s (including parity).

| | attached even-parity bits ↓ | |
|---|---|---|
| H = | 0 | 1 0 0 1 0 0 0 |
| E = | 1 | 1 0 0 0 1 0 1 |
| L = | 1 | 1 0 0 1 1 0 0 |
| L = | 1 | 1 0 0 1 1 0 0 |
| O = | 1 | 1 0 0 1 1 1 1 |

## Error Correction

Error detection is beneficial because the system that receives a datum containing an error knows it has received "damaged goods." Wouldn't it be great if somehow the receiver could also know which bit was wrong? If a binary bit is wrong, then the correct value is simply its complement. Several methods have been developed to accomplish this. In each case, it requires that several bits of "error detection/correction codes" be applied to each transmitted packet of information. As the packet is received, a digital circuit can detect if errors have occurred (even multiple errors) and correct them. This technology is used for massive transfer of high speed data in such applications as magnetic disk drives, flash drives, CD, DVD, Blu-ray Disc, digital television, and broadband Internet networks.

**OUTCOME ASSESSMENT QUESTIONS**

1. Attach an odd-parity bit to the ASCII code for the $ symbol, and express the result in hexadecimal.
2. Attach an even-parity bit to the BCD code for decimal 69.
3. Why can't the parity method detect a double error in transmitted data?

## 2-10 APPLICATIONS

Here are several applications that will serve as a review of some of the concepts covered in this chapter. These applications should give a sense of how the various number systems and codes are used in the digital world. More applications are presented in the end-of-chapter problems.

### APPLICATION 2-1

A typical CD-ROM can store 650 megabytes of digital data. Since mega $= 2^{20}$, how many bits of data can a CD-ROM hold?

**Solution**

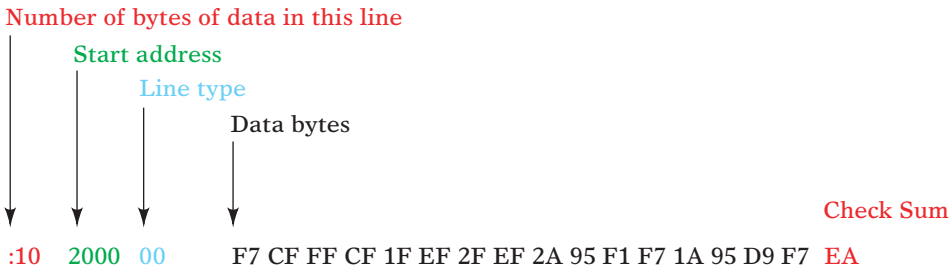
Remember that a byte is eight bits. Therefore, 650 megabytes is $650 \times 2^{20} \times 8 =$ **5,452,595,200 bits.**

### APPLICATION 2-2

In order to program many microcontrollers, the binary instructions are stored in a file on a personal computer in a special way known as Intel Hex Format. The hexadecimal information is encoded into ASCII characters so it can be displayed easily on the PC screen, printed, and easily transmitted one character at a time over a standard PC's serial COM port. One line of an Intel Hex Format file is shown below:

:10200000F7CFFFCF1FEF2FEF2A95F1F71A95D9F7EA

**Intel hex format:**




The first character sent is the ASCII code for a colon, followed by a 1. Each has an even-parity bit appended as the most significant bit. A test instrument captures the binary bit pattern as it goes across the cable to the microcontroller.

(a) What should the binary bit pattern (including parity) look like? (MSB – LSB)

(b) The value 10, following the colon, represents the total hexadecimal number of bytes that are to be loaded into the micro's memory. What is the decimal number of bytes being loaded?

(c) The number 2000 is a four-digit hex value representing the address where the first byte is to be stored. What is the biggest address possible? How many bits would it take to represent this address?

(d) The value of the first data byte is F7. What is the value (in binary) of the least significant nibble of this byte?

**Solution**

(a) ASCII codes are 3A (for:) and 31 (for 1) 00111010 10110001
even-parity bit

(b) 10 hex $= 1 \times 16 + 0 \times 1 = 16$ decimal bytes

(c) FFFF is the biggest possible value. Each hex digit is 4 bits, so we need 16 bits.

FFFF 1111 1111 1111 1111 16 bits

(d) The least significant nibble (4 bits) is represented by hex 7. In binary, this would be 0111.

## APPLICATION 2-3

A small process-control computer uses hexadecimal codes to represent its 16-bit memory addresses.

(a) How many hex digits are required?

(b) What is the range of addresses in hex?

(c) How many memory locations are there?

**Solution**

(a) Since 4 bits convert to a single hex digit, $\frac{16}{4} = 4$ hex digits are needed.

(b) The binary range is $0000000000000000_2$ to $1111111111111111_2$. In hex, this becomes $0000_{16}$ to $FFFF_{16}$.

(c) With 4 hex digits, the total number of addresses is $16^4 = 65{,}536$.

## APPLICATION 2-4

Numbers are entered into a microcontroller-based system in BCD, but stored in straight binary. As a programmer, you must decide whether you need a one-byte or two-byte storage location.

(a) How many bytes do you need if the system takes a two-digit decimal entry?

(b) What if you needed to be able to enter three digits?

**Solution**

(a) With two digits, you can enter values up to 99 ($1001\ 1001_{BCD}$). In binary this value is 01100011, which will fit into an eight-bit memory location. Thus you can use a single byte.

(b) Three digits can represent up to 999 (1001 1001 1001). In binary, this value is 1111100111 (10 bits). Thus you cannot use a single byte; you need two bytes.

## APPLICATION 2-5

When ASCII characters must be transmitted between two independent systems (such as between a computer and a modem), there must be a way of telling the receiver when a new character is coming in. There is often a need to detect errors in the transmission as well. The method of transfer is called asynchronous data communication. The normal resting state of the transmission line is logic 1. When the transmitter sends an ASCII character, it must be "framed" so the receiver knows where the data begins and ends. The first bit must always be a start bit (logic 0). Next the ASCII code is sent

LSB first and MSB last. After the MSB, a parity bit is appended to check for transmission errors. Finally, the transmission is ended by sending a stop bit (logic 1). A typical asynchronous transmission of a seven-bit ASCII code for the pound sign # (23 Hex) with even parity is shown in Figure 2-7.

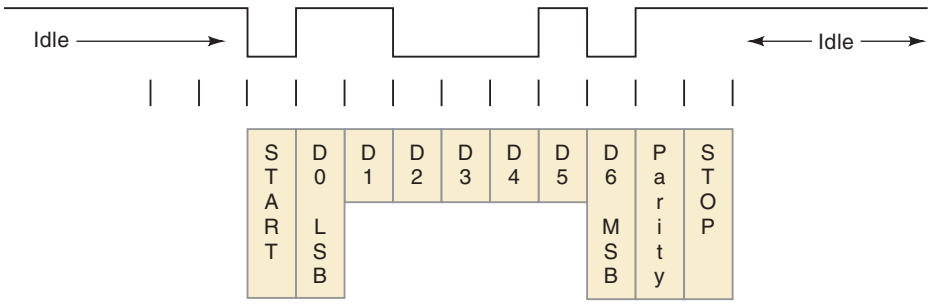



**FIGURE 2-7** Asynchronous serial data with even parity.

**APPLICATION 2-6**

Your PC encounters an error when running an application. The dialog box reports information about the addresses that it could not read or write. What number system is used to report the address area?

**Solution**

These numbers will normally be reported in hexadecimal. Rather than using the subscript 16 as we have done in this text, other methods may be used to indicate hexadecimal (e.g., attaching a 0x prefix to the number).

## SUMMARY

1. The hexadecimal number system is used in digital systems and computers as an efficient way of representing binary quantities.
2. In conversions between hex and binary, each hex digit corresponds to four bits.
3. The repeated-division method is used to convert decimal numbers to binary or hexadecimal.
4. Using an $N$-bit binary number, we can represent decimal values from 0 to $2^N - 1$.
5. The BCD code for a decimal number is formed by converting each digit of the decimal number to its four-bit binary equivalent.
6. The Gray code defines a sequence of bit patterns in which only one bit changes between successive patterns in the sequence.
7. A byte is a string of eight bits. A nibble is four bits. The word size depends on the system.
8. An alphanumeric code is one that uses groups of bits to represent all of the various characters and functions that are part of a typical computer's keyboard. The ASCII code is the most widely used alphanumeric code.
9. The parity method for error detection attaches a special parity bit to each transmitted group of bits.

## IMPORTANT TERMS

hexadecimal number system
straight binary coding
binary-coded-decimal (BCD) code
Gray code
byte
nibble
word
word size
alphanumeric code
American Standard Code for Information Interchange (ASCII)
parity method
parity bit

## PROBLEMS

### SECTIONS 2-1 AND 2-2

2-1. Convert these binary numbers to decimal.

| | | |
|---|---|---|
| (a)*11011 | (e)*1001001 | (i)*1100110011 |
| (b) 1010111 | (f) 11000001 | (j) 11110000 |
| (c)*1100110 | (g)*10000111 | (k)*11111111101 |
| (d) 1110 | (h) 100100010 | (l) 100001111 |

2-2. Convert the following decimal values to binary.

| | | |
|---|---|---|
| (a)*37 | (e)*77 | (i)*511 |
| (b) 13 | (f) 390 | (j) 25 |
| (c)*189 | (g)*205 | (k) 52 |
| (d) 1000 | (h) 2133 | (l) 47 |

2-3. What is the largest decimal value that can be represented by (a)* an eight-bit binary number? (b) A 16-bit number?

### SECTION 2-4

2-4. Convert each hex number to its decimal equivalent.

| | | |
|---|---|---|
| (a)*921 | (e)*2A | (i) A1D |
| (b) CAB | (f) 35 | (j) 120 |
| (c)*1DE | (g)*D0 | (k) 8A |
| (d) D66 | (h) B13 | (l) F00D |

2-5. Convert each of the following decimal numbers to hex.

| | | |
|---|---|---|
| (a)*59 | (e)*771 | (i) 29 |
| (b) 372 | (f) 2313 | (j) 33 |
| (c)*919 | (g)*65,536 | (k) 100 |
| (d) 1024 | (h) 255 | (l) 200 |

2-6. Convert each of the hex values from Problem 2-4 to binary.

2-7. Convert the binary numbers in Problem 2-1 to hex.

2-8. Count the number of hex numbers between $100_{16}$ and $125_{16}$, including both these numbers.

2-9.*When a large decimal number is to be converted to binary, it is sometimes easier to convert it first to hex, and then from hex to binary. Try this procedure for $2133_{10}$ and compare it with the procedure used in Problem 2-2(h).

2-10. How many hex digits are required to represent decimal numbers up to 1,999? How many bits are required?

*Answers to problems marked with an asterisk can be found in the back of the text.

2-11. Convert these hex values to decimal.

| | | |
|---|---|---|
| (a)*B1B | (e)*DEC | (i) F00 |
| (b) C2 | (f) A0 | (j) 98 |
| (c)*BE | (g)*B1D | (k) 1EEE |
| (d) 707 | (h) 234 | (l) 56 |

2-12. Convert these decimal values to hex.

| | | |
|---|---|---|
| (a)*75 | (e)*7245 | (i) 95 |
| (b) 314 | (f) 498 | (j) 89 |
| (c)*2048 | (g)*25,619 | (k) 128 |
| (d) 24 | (h) 4095 | (l) 256 |

2-13. Take each four-bit binary number in the order they are written and write the equivalent hex digit without performing a calculation by hand or by calculator.

| | | | |
|---|---|---|---|
| (a) 1001 | (e) 1111 | (i) 1011 | (m) 0001 |
| (b) 1101 | (f) 0010 | (j) 1100 | (n) 0101 |
| (c) 1000 | (g) 1010 | (k) 0011 | (o) 0111 |
| (d) 0000 | (h) 1001 | (l) 0100 | (p) 0110 |

2-14. Take each hex digit and write its four-bit binary value without performing any calculations by hand or by calculator.

| | | | |
|---|---|---|---|
| (a) 6 | (e) 4 | (i) 9 | (m) 0 |
| (b) 7 | (f) 3 | (j) A | (n) 8 |
| (c) 5 | (g) C | (k) 2 | (o) D |
| (d) 1 | (h) B | (l) F | (p) 9 |

2-15. What is the range of values that can be obtained from four hex digits starting with *A*?

2-16.*Convert the hex values in Problem 2-11 to binary.

2-17.*List the hex numbers in sequence from 280 to 2A0.

2-18. How many hex digits are required to represent a binary number having 21 bits, assuming that the first bit is 1? Repeat the exercise for 128 bits.

## SECTION 2-4

2-19. Encode these decimal numbers in BCD.

| | | |
|---|---|---|
| (a)*47 | (e)*13 | (i)* 72 |
| (b) 962 | (f) 529 | (j) 38 |
| (c)*187 | (g)*89,627 | (k)*61 |
| (d) 6727 | (h) 1024 | (l) 90 |

2-20. If a straight binary number has 10 bits in it, find the maximum number of bits that will be required to represent that number using BCD code.

2-21. The following numbers are in BCD. Convert them to decimal.

| | |
|---|---|
| (a)*1001011101010010 | (f) 010101010101 |
| (b) 000110000100 | (g) 10111 |
| (c)*011010010101 | (h) 010110 |
| (d) 0111011101110101 | (i) 1110101 |
| (e)*010010010010 | |

## SECTION 2-7

2-22.*(a) How many bits are contained in eight bytes?

(b) What is the largest hex number that can be represented in four bytes?

(c) What is the largest BCD-encoded decimal value that can be represented in three bytes?

2-23. (a) Refer to Table 2-5. What is the most significant nibble of the ASCII code for the letter X?

(b) How many nibbles can be stored in a 16-bit word?

(c) How many bytes does it take to make up a 24-bit word?

## SECTIONS 2-8 AND 2-9

2-24. A certain message is encoded using padded ASCII-code and stored in memory as "01001000 01000101 01001100 01001100 01001111." What is the message?

2-25.*What will be the encoded message if the message in Problem 2.24 has an odd-parity bit attached? An even-parity bit? Give your results in hex.

2-26. The following bytes (shown in hex) represent a person's name as it would be stored in a computer's memory. Each byte is a padded ASCII code. Determine the name of each person.

(a) *42 45 4E 20 53 4D 49 54 48

(b) 4A 6F 65 20 47 72 65 65 6E

2-27. Convert the following decimal numbers to BCD code and then attach an *odd*-parity bit.

| | | | |
|---|---|---|---|
| (a)*74 | (c)*8884 | (e)*165 | (g) 11 |
| (b) 38 | (d) 275 | (f) 9201 | (h) 51 |

2-28.*In a certain digital system, the decimal numbers from 000 through 999 are represented in BCD code. An *odd*-parity bit is also included at the end of each code group. Examine each of the code groups below, and assume that each one has just been transferred from one location to another. Some of the groups contain errors. Assume that *no more than* two errors have occurred for each group. Determine which of the code groups have a single error and which of them *definitely* have a double error. (*Hint:* Remember that this is a BCD code.)

(a) 1001010110000


(b) 0100011101100

(c) 0111110000011

(d) 1000011000101

2-29. Suppose that the receiver received the following data from the transmitter of Example 2-17:

01001000
11000101
11001100
11001000
11001100

What errors can the receiver determine in these received data?

## DRILL QUESTIONS

2-30.*Perform each of the following conversions. For some of them, you may want to try several methods to see which one works best for you. For example, a binary-to-decimal conversion may be done directly, or it may be done as a binary-to-hex conversion followed by a hex-to-decimal conversion.

(a) $342_{10}$ = _____ 2
(b) $6789_{10}$ = _____ 2
(c) $1001101_2$ = _____ 10
(d) $1100\ 1110\ 0110\ 0101_2$ = _____ 16
(e) $925_{10}$ = _____ 16
(f) $951_{10}$ = _____ (BCD)
(g) $785_{16}$ = _____ (BCD)
(h) $6393_{10}$ = _____ 16
(i) $BAD_{16}$ = _____ 2
(j) $C41E_{16}$ = _____ 10
(k) $1024_{10}$ = _____ 16
(l) $4132_{10}$ = _____ 16
(m) $885_{10}$ = _____ (BCD)
(n) 1000 1001 0101 (BCD) = _____ 10
(o) $975_{16}$ = _____ 2
(p) $F03_{16}$ = _____ 2
(q) 01100101 (BCD) = _____ 2
(r) $101010_2$ = _____ (BCD)

2-31.*Encode the binary number 10101100 in the following ways.

(a) Decimal
(b) Hexadecimal
(c) BCD
(d) The ASCII value of the equivalent hex (representing each hex digit as character)

2-32.*Fill in the blanks with the correct word or words.

(a) The double-dabble method is a method for _____ conversion without keeping track of column weights.
(b) Using *N* bits, we can count through _____ decimal numbers, ranging from 0 to _____.
(c) Conversion from _____ to binary requires direct conversion of each digit to its four-bit equivalent.
(d) In BCD code, each digit of a decimal number is represented by a four-bit _____ number and does not use the numbers _____ through _____.
(e) A(n) _____ code uses groups of bits to represent all the characters and functions that are part of a typical computer's keyboard.
(f) The word size of a digital system that can handle four bytes at a time is _____.
(g) The quadrature shaft encoder is an application of the concept of _____ code.
(h) A nibble consists of _____ bits and a byte consists of _____ nibbles.

2-33. Write the binary number that results when each of the following numbers is incremented by one.
(a)*10101 (b) 110011 (c) 101111 (d) 11011

2-34. Decrement each binary number by one.
(a)*1000 (b) 100100 (c) 10101010 (d) 11100

2-35. Write the number that results when each of the following is incremented.
(a)*$6599_{16}$ (c)*$234F_{16}$ (e)*1001 1001 (BCD) (g) 1000 0000 (BCD)
(b) $600_{16}$ (d) $89A_{16}$ (f) $FFF_{16}$ (h) 0101 1001 (BCD)

2-36.*Repeat Problem 2-35 for the decrement operation.

## CHALLENGING EXERCISES

2-37.*In a microcomputer, the *addresses* of memory locations are binary numbers that identify each memory circuit where a byte is stored. The number of bits that make up an address depends on how many memory locations there are. Since the number of bits can be very large, the addresses are often specified in hex instead of binary.

(a) If a microcomputer uses a 20-bit address, how many different memory locations are there?

(b) How many hex digits are needed to represent the address of a memory location?

(c) What is the hex address of the 256th memory location? (*Note:* The first address is always 0.)

(d) The computer program is stored in the lowest 2 kbyte block of memory. Give the start and end address of this block.

2-38. In an audio CD, the audio voltage signal is typically sampled about 44,000 times per second, and the value of each sample is recorded on the CD surface as a binary number. In other words, each recorded binary number represents a single voltage point on the audio signal waveform.

(a) If the binary numbers are six bits in length, how many different voltage values can be represented by a single binary number? Repeat for eight bits and ten bits.

(b) If ten-bit numbers are used, how many bits will be recorded on the CD in 1 second?

(c) If a CD can typically store 5 billion bits, how many seconds of audio can be recorded when ten-bit numbers are used?

2-39.*A black-and-white digital camera lays a fine grid over an image and then measures and records a binary number representing the level of gray it sees in each cell of the grid. For example, if four-bit numbers are used, the value of black is set to 0000 and the value of white to 1111, and any level of gray is somewhere between 0000 and 1111. If six-bit numbers are used, black is 000000, white is 111111, and all grays are between the two.

Suppose we wanted to distinguish among 254 different levels of gray within each cell of the grid. How many bits would we need to use to represent these levels?

2-40. A 3-Megapixel digital camera stores an eight-bit number for the brightness of each of the primary colors (red, green, blue) found in each picture element (pixel). If every bit is stored (no data compression),

how many pictures can be stored on a 128-Megabyte memory card? (Note: In digital systems, Mega means $2^{20}$.)

2-41. Construct a table showing the binary, hex, and BCD representations of all decimal numbers from 0 to 15. Compare your table with Table 2-4.

## ANSWERS TO OUTCOME ASSESSMENT QUESTIONS

### SECTION 2-1

1. 2267 2. 32768 3. 2267

### SECTION 2-2

1. 1010011 2. 1011011001 3. 20 bits

### SECTION 2-3

1. 9422 2. C2D; 110000101101 3. 97B5 4. E9E, E9F, EA0, EA1
5. 11010100100111 6. 0 to 65,535

### SECTION 2-4

1. $10110010_2$; 000101111000 (BCD) 2. 32 3. Advantage: ease of conversion. Disadvantage: BCD requires more bits.

### SECTION 2-5

1. 0111 2. 0110

### SECTION 2-7

1. One 2. 9999 3. One 4. One

### SECTION 2-8

1. 43, 4F, 53, 54, 20, 3D, 20, 24, 37, 32 2. STOP

### SECTION 2-9

1. A4 2. 001101001 3. Two errors in the data would not change the oddness or evenness of the number of 1s in the data.

CHAPTER 3

# DESCRIBING LOGIC CIRCUITS

## OUTLINE

## CHAPTER OUTCOMES

*Upon completion of this chapter, you will be able to:*

- Perform the three basic logic operations.
- Describe the operation of and construct the truth tables for the AND, NAND, OR, and NOR gates, and the NOT (INVERTER) circuit.
- Draw timing diagrams for the various logic-circuit gates.
- Write the Boolean expression for the logic gates and combinations of logic gates.
- Implement logic circuits using basic AND, OR, and NOT gates.
- Use Boolean algebra to simplify complex logic circuits.
- Use DeMorgan's theorems to simplify logic expressions.
- Use either of the universal gates (NAND or NOR) to implement a circuit represented by a Boolean expression.
- Explain the advantages of constructing a logic-circuit diagram using the alternate gate symbols versus the standard logic-gate symbols.
- Describe the concept of active-LOW and active-HIGH logic signals.
- Describe and measure propagation delay time.
- Use several methods to describe the operation of logic circuits.
- Interpret simple circuits defined by a hardware description language (HDL).
- Explain the difference between an HDL and a computer programming language.
- Create an HDL file for a simple logic gate.
- Create an HDL file for combinational circuits with intermediate variables.

## INTRODUCTION

Chapters 1 and 2 introduced the concepts of logic levels and logic circuits. In logic, only two possible conditions exist for any input or output: true and false. The binary number system uses only two digits, 1 and 0, so it is perfect for representing logical relationships. Digital logic circuits use predefined voltage ranges to represent these binary states. Using these concepts, we can create circuits made of little more than processed beach sand and wire that make consistent, intelligent, logical decisions. It is vitally important that we have a method to describe the logical decisions made by these circuits. In other words, we must describe how they operate. In this chapter, we will discover many ways to describe their operation. Each description method is important because all these methods commonly

appear in technical literature and system documentation and are used in conjunction with modern design and development tools.

Life is full of examples of circumstances that are in one state or another. For example, a creature is either alive or dead, a light is either on or off, a door is locked or unlocked, and it is either raining or it is not. In 1854, a mathematician named George Boole wrote *An Investigation of the Laws of Thought,* in which he described the way we make logical decisions based on true or false circumstances. The methods he described are referred to today as Boolean logic, and the system of using symbols and operators to describe these decisions is called Boolean algebra. In the same way we use symbols such as $x$ and $y$ to represent unknown numerical values in regular algebra, Boolean algebra uses symbols to represent a logical expression that has one of two possible values: true or false. The logical expression might be *door is closed*, *button is pressed*, or *fuel is low*. Writing these expressions is very tedious, and so we tend to substitute symbols such as *A*, *B*, and *C*.

The main purpose of these logical expressions is to describe the relationship between a logic circuit's output (the decision) and its inputs (the circumstances). In this chapter, we will study the most basic logic circuits—*logic gates*—which are the fundamental building blocks from which all other logic circuits and digital systems are constructed. We will see how the operation of the different logic gates and the more complex circuits formed from combinations of logic gates can be described and analyzed using Boolean algebra. We will also get a glimpse of how Boolean algebra can be used to simplify a circuit's Boolean expression so that the circuit can be rebuilt using fewer logic gates and/or fewer connections. Much more will be done with circuit simplification in Chapter 4.

Boolean algebra is not only used as a tool for analysis and simplification of logic systems. It can also be used as a tool to create a logic circuit that will produce the desired input/output relationship. This process is often called synthesis of logic circuits as opposed to analysis. Other techniques have been used in the analysis, synthesis, and documentation of logic systems and circuits including truth tables, schematic symbols, timing diagrams, and—last but by no means least—language. To categorize these methods, we could say that Boolean algebra is a mathematic tool, truth tables are data organizational tools, schematic symbols are drawing tools, timing diagrams are graphing tools, and language is the universal description tool.

Today, any of these tools can be used to provide input to computers. The computers can be used to simplify and translate between these various forms of description and ultimately provide an output in the form necessary to implement a digital system. To take advantage of the powerful benefits of computer software, we must first fully understand the acceptable ways for describing these systems in terms the computer can understand. This chapter will lay the groundwork for further study of these vital tools for synthesis and analysis of digital systems.

Clearly the tools described here are invaluable tools in describing, analyzing, designing, and implementing digital circuits. The student who expects to work in the digital field must work hard at understanding and becoming comfortable with Boolean algebra (believe us, it's much, much easier than conventional algebra) and all the other tools. Do *all* of the examples, exercises, and problems, even the ones your instructor doesn't assign. When those run out, make up your own. The time you spend will be well worth it because you will see your skills improve and your confidence grow.

# 3-1 BOOLEAN CONSTANTS AND VARIABLES

## OUTCOMES

*Upon completion of this section, you will be able to:*

- Differentiate between Boolean variables and constants.
- State the possible values of a Boolean variable or constant.
- Define Boolean algebra.

Boolean algebra differs in a major way from ordinary algebra because Boolean constants and variables are allowed to have only two possible values, 0 or 1. A Boolean variable is a quantity that may, at different times, be equal to either 0 or 1. Boolean variables are often used to represent the voltage level present on a wire or at the input/output terminals of a circuit. For example, in a certain digital system, the Boolean value of 0 might be assigned to any voltage in the range from 0 to 0.8 V, while the Boolean value of 1 might be assigned to any voltage in the range 2 to 5 V.*

Thus, Boolean 0 and 1 do not represent actual numbers but instead represent the state of a voltage variable, or what is called its **logic level.** A voltage in a digital circuit is said to be at the logic 0 level or the logic 1 level, depending on its actual numerical value. In digital logic, several other terms are used synonymously with 0 and 1. Some of the more common ones are shown in Table 3-1. We will use the 0/1 and LOW/HIGH designations most of the time.

**TABLE 3-1** Common logic terms.

| Logic 0 | Logic 1 |
|---|---|
| False | True |
| Off | On |
| LOW | HIGH |
| No | Yes |
| Open switch | Closed switch |

As we said in the introduction, **Boolean algebra** is a means for expressing the relationship between a logic circuit's inputs and outputs. The inputs are considered logic variables whose logic levels at any time determine the output levels. In all our work to follow, we shall use letter symbols to represent logic variables. For example, the letter $A$ might represent a certain digital circuit input or output, and at any time we must have either $A = 0$ or $A = 1$: if not one, then the other.

Because only two values are possible, Boolean algebra is relatively easy to work with compared with ordinary algebra. In Boolean algebra, there are no fractions, decimals, negative numbers, square roots, cube roots, logarithms, imaginary numbers, and so on. In fact, in Boolean algebra there are only *three* basic operations: *OR*, *AND*, and *NOT.*

These basic operations are called *logic operations.* Digital circuits called *logic gates* can be constructed from diodes, transistors, and resistors connected so that the circuit output is the result of a basic logic operation *(OR*, *AND*, *NOT)* performed on the inputs. We will be using Boolean algebra first

*Voltages between 0.8 and 2 V are undefined (neither 0 nor 1) and should not occur under normal circumstances.

to describe and analyze these basic logic gates, then later to analyze and design combinations of logic gates connected as logic circuits.

A Boolean constant represents a point in the circuit where the logic level never changes. In other words, each bit is "hard-wired" to either a logic 1 or a logic 0. A good name for a constant logic 1 would be VCC or HIGH. VCC is a common name given to the positive voltage supply of digital systems. A good name for a constant logic 0 would be LOW or GND. GND stands for *ground* which is the term used for the negative side of the power supply for digital circuits. Recall from Chapter 1 that these voltage levels are used to represent 1s and 0s.

**OUTCOME ASSESSMENT QUESTIONS**

1. A circuit has more inputs than your application needs. The extra inputs will not affect your application if they are LOW. Should you apply a variable or a constant? What would be a good name for this point in the circuit?
2. The quadrature encoder (two-bit Gray code) described in Chapter 2 has its two channels A and B connected as inputs to a logic circuit. Should these inputs be labeled as variables or constants? What would be a good name for each input?
3. Define Boolean algebra.

## 3-2 TRUTH TABLES

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Construct a truth table.
- For any input, identify the correct output.
- Determine the size of the truth table based on the number of variables.

A **truth table** is a means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs. Figure 3-1(a) illustrates a truth table for one type of two-input logic circuit. The table lists all possible combinations of logic levels present at inputs *A* and *B*, along with the corresponding output level *x*. The first entry in the table shows that when *A* and *B*

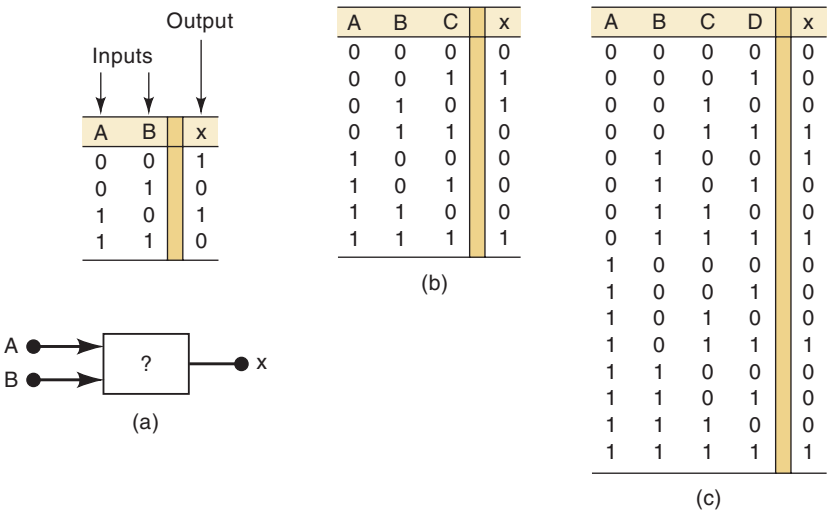



**FIGURE 3-1** Example truth tables for (a) two-input, (b) three-input, and (c) four-input circuits.

are both at the 0 level, the output $x$ is at the 1 level or, equivalently, in the 1 state. The second entry shows that when input $B$ is changed to the 1 state, so that $A = 0$ and $B = 1$, the output $x$ becomes a 0. In a similar way, the table shows what happens to the output state for any set of input conditions.

Figures 3-1(b) and (c) show samples of truth tables for three- and four-input logic circuits. Again, each table lists all possible combinations of input logic levels on the left, with the resultant logic level for output $x$ on the right. Of course, the actual values for $x$ will depend on the type of logic circuit.

Note that there are 4 table entries for the two-input truth table, 8 entries for a three-input truth table, and 16 entries for the four-input truth table. The number of input combinations will equal $2^N$ for an $N$-input truth table. Also note that the list of all possible input combinations follows the binary counting sequence, and so it is an easy matter to write down all of the combinations without missing any.

**OUTCOME ASSESSMENT QUESTIONS**

1. What is the output state of the four-input circuit represented in Figure 3-1(c) when all inputs except $B$ are 1?
2. Repeat question 1 for the following input conditions: $A = 1, B = 0, C = 1, D = 0$.
3. How many table entries are needed for a five-input circuit?

## 3-3 OR OPERATION WITH OR GATES

### OUTCOMES

*Upon completion of this section, you will be able to:*

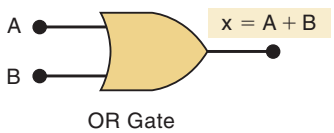
- Define the OR logic function.
- Write Boolean equations using the OR function.
- Draw the logic symbol for the OR function.
- Write a truth table describing the OR function.
- Draw a timing diagram that demonstrates the OR function.
- Use any of the above methods to infer the correct output of a logic circuit based on its input.

The **OR operation** is the first of the three basic Boolean operations to be learned. An example can be found in the kitchen oven. The light inside the oven should turn on if either the *oven light switch is on* OR if the *door is opened*. The letter $A$ could be used to represent the *oven light switch is on* (true or false) and $B$ could represent *door is opened* (true or false). The letter $x$ could represent the *light is on* (true or false). The truth table in Figure 3-2(a) shows

OR

| A | B | x = A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(a)




(b)

**FIGURE 3-2** (a) Truth table defining the OR operation; (b) circuit symbol for a two-input OR gate.

what happens when two logic inputs, $A$ and $B$, are combined using the OR operation to produce the output $x$. The table shows that $x$ is a logic 1 for every combination of input levels where one *or* more inputs are 1. The only case where $x$ is a 0 is when both inputs are 0.

The Boolean expression for the OR operation is

$$x = A + B$$

In this expression, the + sign does not stand for ordinary addition; it stands for the OR operation. The OR operation is similar to ordinary addition except for the case where $A$ and $B$ are both 1; the OR operation produces $1 + 1 = 1$, not $1 + 1 = 2$. In Boolean algebra, 1 is as high as we go, so we can never have a result greater than 1. The same holds true for combining three inputs using the OR operation. Here we have $x = A + B + C$. If we consider the case where all three inputs are 1, we have

$$x = 1 + 1 + 1 = 1$$

The expression $x = A + B$ is read as "$x$ equals $A$ OR $B$," which means that $x$ will be 1 when $A$ or $B$ or both are 1. Likewise, the expression $x = A + B + C$ is read as "$x$ equals $A$ OR $B$ OR $C$," which means that $x$ will be 1 when $A$ or $B$ or $C$ or any combination of them are 1. To describe this circuit in the English language, we could say that *x is true (1) WHEN A is true (1) OR B is true (1) OR C is true (1).*

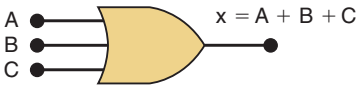
## OR Gate

In digital circuitry, an **OR gate*** is a circuit that has two or more inputs and whose output is equal to the OR combination of the inputs. Figure 3-2(b) is the logic symbol for a two-input OR gate. The inputs $A$ and $B$ are logic voltage levels, and the output $x$ is a logic voltage level whose value is the result of the OR operation on $A$ and $B$; that is, $x = A + B$. In other words, the OR gate operates so that its output is HIGH (logic 1) if either input *A or B or both* are at a logic 1 level. The OR gate output will be LOW (logic 0) only if all its inputs are at logic 0.

This same idea can be extended to more than two inputs. Figure 3-3 shows a three-input OR gate and its truth table. Examination of this truth table shows again that the output will be 1 for every case where one or more inputs are 1. This general principle is the same for OR gates with any number of inputs.

Using the language of Boolean algebra, the output $x$ can be expressed as $x = A + B + C$, where again it must be emphasized that the + represents

**FIGURE 3-3** Symbol and truth table for a three-input OR gate.




| A | B | C | x = A + B + C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

*The term *gate* comes from the inhibit/enable operation discussed in Chapter 4.

the OR operation. The output of any OR gate, then, can be expressed as the OR combination of its various inputs. We will put this to use when we subsequently analyze logic circuits.

## Summary of the OR Operation

The important points to remember concerning the OR operation and OR gates are:

1. The OR operation produces a result (output) of 1 whenever *any* input is a 1. Otherwise the output is 0.
2. An OR gate is a logic circuit that performs an OR operation on the circuit's inputs.
3. The expression $x = A + B$ is read as "$x$ equals $A$ OR $B$."

**EXAMPLE 3-1**

In many industrial control systems, it is required to activate an output function whenever any one of several inputs is activated. For example, in a chemical process it may be desired that an alarm be activated whenever the process temperature exceeds a maximum value *or* whenever the pressure goes above a certain limit. Figure 3-4 is a block diagram of this situation. The temperature transducer circuit produces an output voltage proportional to the process temperature. This voltage, $V_T$, is compared with a temperature reference voltage, $V_{TR}$, in a voltage comparator circuit. The comparator output, $T_H$, is normally a low voltage (logic 0), but it switches to a high voltage (logic 1) when $V_T$ exceeds $V_{TR}$, indicating that the process temperature is too high. A similar arrangement is used for the pressure measurement, so that its associated comparator output, $P_H$, goes from LOW to HIGH when the pressure is too high. What is the purpose of the OR gate?

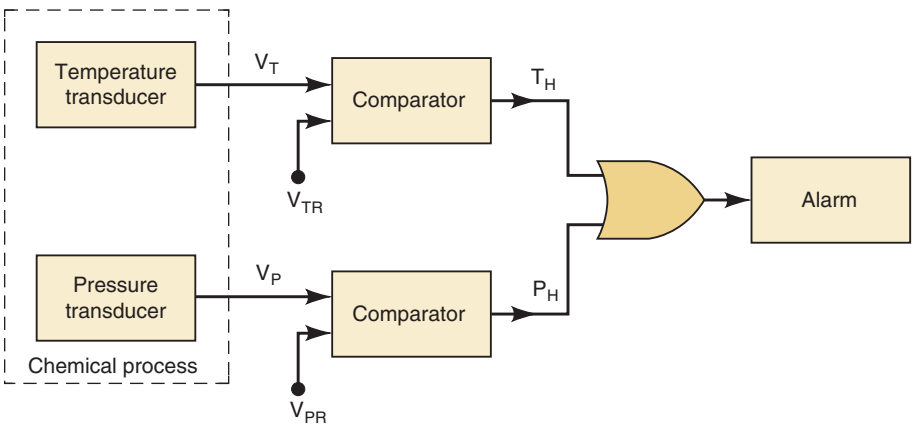



**FIGURE 3-4** Example of the use of an OR gate in an alarm system.

### Solution

Since we want the alarm to be activated when either temperature *or* pressure is too high, it should be apparent that the two comparator outputs can be fed to a two-input OR gate. The OR gate output thus goes HIGH (1) for either alarm condition and will activate the alarm. This same idea can obviously be extended to situations with more than two process variables.

## EXAMPLE 3-2

Determine the OR gate output in Figure 3-5. The OR gate inputs *A* and *B* are varying according to the timing diagrams shown. For example, *A* starts out LOW at time $t_0$, goes HIGH at $t_1$, back to LOW at $t_3$, and so on.

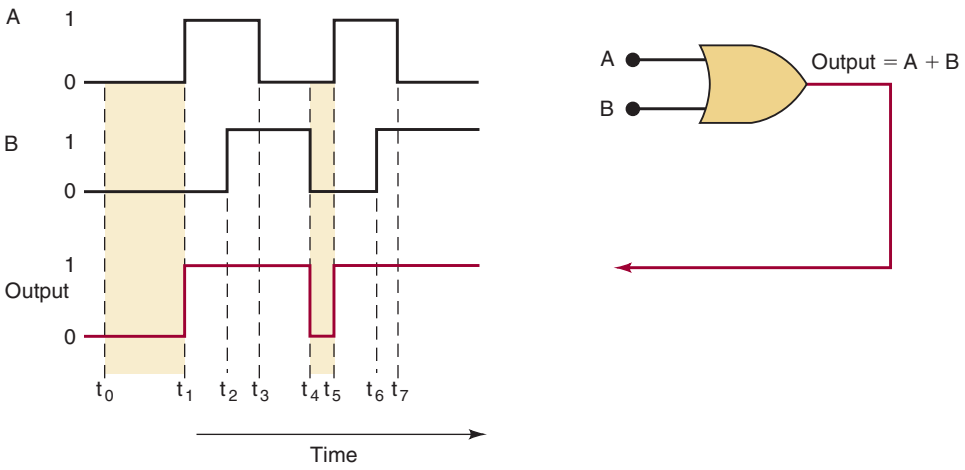



**FIGURE 3-5** Example 3-2.

### Solution

The OR gate output will be HIGH whenever *any* input is HIGH. Between time $t_0$ and $t_1$, both inputs are LOW, so OUTPUT = LOW. At $t_1$, input *A* goes HIGH while *B* remains LOW. This causes OUTPUT to go HIGH at $t_1$ and stay HIGH until $t_4$ because, during this interval, one or both inputs are HIGH. At $t_4$, input *B* goes from 1 to 0 so that now both inputs are LOW, and this drives OUTPUT back to LOW. At $t_5$, *A* goes HIGH, sending OUTPUT back HIGH, where it stays for the rest of the shown time span.

## EXAMPLE 3-3A

For the situation depicted in Figure 3-6, determine the waveform at the OR gate output.

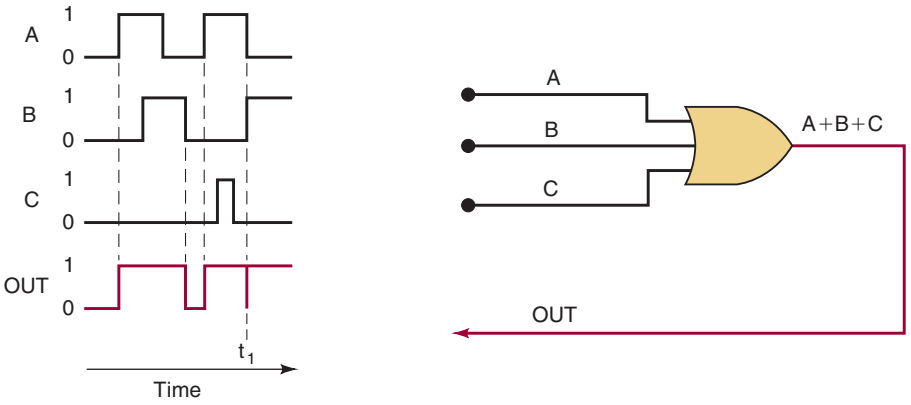



**FIGURE 3-6** Examples 3-3A and B.

### Solution

The three OR gate inputs *A*, *B*, and *C* are varying, as shown by their waveform diagrams. The OR gate output is determined by realizing that it will be HIGH whenever *any* of the three inputs is at a HIGH level. Using this reasoning, the OR output waveform is as shown in the figure. Particular

attention should be paid to what occurs at time $t_1$. The diagram shows that at that instant of time, input *A* is going from HIGH to LOW while input *B* is going from LOW to HIGH. Since these inputs are making their transitions at approximately the same time, and since these transitions take a certain amount of time, there is a short interval when these OR gate inputs are both in the undefined range between 0 and 1. When this occurs, the OR gate output also becomes a value in this range, as evidenced by the glitch or spike on the output waveform at $t_1$. The occurrence of this glitch and its size (amplitude and width) depend on the speed with which the input transitions occur.

**EXAMPLE 3-3B**

What would happen to the glitch in the output in Figure 3-6 if input *C* sat in the HIGH state while *A* and *B* were changing at time $t_1$?

**Solution**

With the *C* input HIGH at $t_1$, the OR gate output will remain HIGH, regardless of what is occurring at the other inputs, because any HIGH input will keep an OR gate output HIGH. Therefore, the glitch will not appear in the output.

**OUTCOME ASSESSMENT QUESTIONS**

1. What is the only set of input conditions that will produce a LOW output for any OR gate?
2. Write the Boolean expression for a six-input OR gate.
3. If the *A* input in Figure 3-6 is permanently kept at the 1 level, what will the resultant output waveform be?

## 3-4 AND OPERATION WITH AND GATES

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Define the AND logic function.
- Write Boolean equations using the AND function.
- Draw the logic symbol for the AND function.
- Write a truth table describing the AND function.
- Draw a timing diagram that demonstrates the AND function.
- Use any of the above methods to infer the correct output of a logic circuit based on its input.

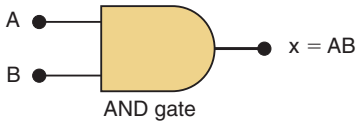
The **AND operation** is the second basic Boolean operation. As an example of the use of AND logic, consider a typical clothes dryer. It is drying clothes (heating, tumbling) only if the *timer is set above zero* AND the *door is closed*. Let's assign *A* to represent *timer is set* (T/F), *B* to represent *door is closed* (T/F), and *x* can represent the *heater and motor are on* (T/F). The truth table in Figure 3-7(a) shows what happens when two logic inputs, *A* and *B*, are combined using the AND operation to produce output *x*. The table shows that *x* is a logic 1 only when both *A* and *B* are at the logic 1 level. For any case where one of the inputs is 0, the output is 0.

**FIGURE 3-7** (a) Truth table for the AND operation; (b) AND gate symbol.

AND

| A | B | x = A·B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(a)




(b)

The Boolean expression for the AND operation is

$$x = A \cdot B$$

In this expression, the · sign stands for the Boolean AND operation and not the multiplication operation. However, the AND operation on Boolean variables operates the same as ordinary multiplication, as examination of the truth table shows, so we can think of them as being the same. This characteristic can be helpful when evaluating logic expressions that contain AND operations.

The expression $x = A \cdot B$ is read as "$x$ equals $A$ AND $B$," which means that $x$ will be 1 only when $A$ and $B$ are both 1. The · sign is usually omitted so that the expression simply becomes $x = AB$. For the case when three inputs are ANDed, we have $x = A \cdot B \cdot C = ABC$. This is read as "$x$ equals $A$ AND $B$ AND $C$," which means that $x$ will be 1 only when $A$ and $B$ and $C$ are all 1.

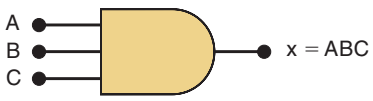
## AND Gate

The logic symbol for a two-input **AND gate** is shown in Figure 3-7(b). The AND gate output is equal to the AND product of the logic inputs; that is, $x = AB$. In other words, the AND gate is a circuit that operates so that its output is HIGH only when all its inputs are HIGH. For all other cases, the AND gate output is LOW.

This same operation is characteristic of AND gates with more than two inputs. For example, a three-input AND gate and its accompanying truth table are shown in Figure 3-8. Once again, note that the gate output is 1 only for the case where $A = B = C = 1$. The expression for the output is $x = ABC$. For a four-input AND gate, the output is $x = ABCD$, and so on.

**FIGURE 3-8** Truth table and symbol for a three-input AND gate.

| A | B | C | x = ABC |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




Note the difference between the symbols for the AND gate and the OR gate. Whenever you see the AND symbol on a logic-circuit diagram, it tells you that the output will go HIGH *only* when *all* inputs are HIGH. Whenever you see the OR symbol, it means that the output will go HIGH when *any* input is HIGH.

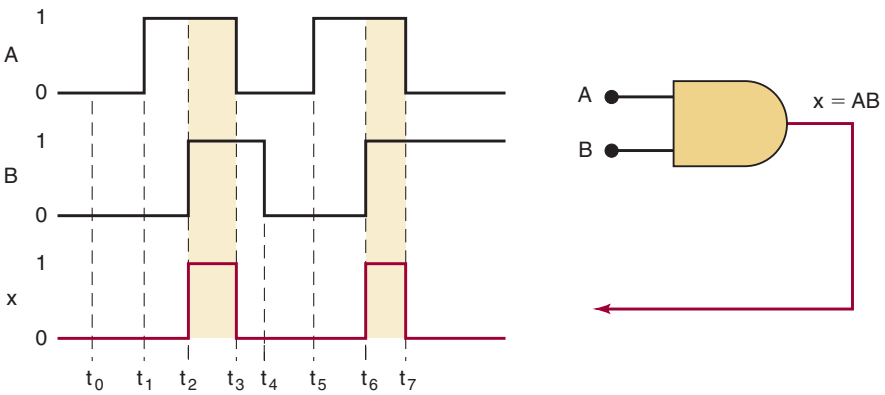
## Summary of the AND Operation

1. The AND operation is performed the same as ordinary multiplication of 1s and 0s.
2. An AND gate is a logic circuit that performs the AND operation on the circuit's inputs.
3. An AND gate output will be 1 *only* for the case when *all* inputs are 1; for all other cases, the output will be 0.
4. The expression $x = AB$ is read as "$x$ equals $A$ AND $B$."

**EXAMPLE 3-4**

Determine the output $x$ from the AND gate in Figure 3-9 for the given input waveforms.

**FIGURE 3-9** Example 3-4.



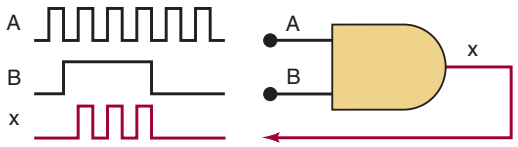

### Solution

The output of an AND gate is determined by realizing that it will be HIGH only when all inputs are HIGH at the same time. For the input waveforms given, this condition is met only during intervals $t_2 - t_3$ and $t_6 - t_7$. At all other times, one or more of the inputs are 0, thereby producing a LOW output. Note that input level changes that occur while the other input is LOW have no effect on the output.

**EXAMPLE 3-5A**

Determine the output waveform for the AND gate shown in Figure 3-10.

**FIGURE 3-10** Examples 3-5A and B.




### Solution

The output $x$ will be at 1 only when $A$ and $B$ are both HIGH at the same time. Using this fact, we can determine the $x$ waveform as shown in the figure.

Notice that the $x$ waveform is 0 whenever $B$ is 0, regardless of the signal at $A$. Also notice that whenever $B$ is 1, the $x$ waveform is the same as $A$. Thus, we can think of the $B$ input as a *control* input whose logic level determines whether or not the $A$ waveform gets through to the $x$ output. In this situation, the AND gate is used as an *inhibit circuit*. We can say that $B = 0$ is the inhibit condition producing a 0 output. Conversely, $B = 1$ is the *enable* condition, which enables $A$ to reach the output. This inhibit operation is an important application of AND gates, which will be encountered later.

**EXAMPLE 3-5B**

What will happen to the $x$ output waveform in Figure 3-10 if the $B$ input is kept at the 0 level?

**Solution**

With $B$ kept LOW, the $x$ output will also stay LOW. This can be reasoned in two different ways. First, with $B = 0$ we have $x = A \cdot B = A \cdot 0 = 0$ because anything multiplied (ANDed) by 0 will be 0. Another way to look at it is that an AND gate requires that all inputs be HIGH for the output to be HIGH, and this cannot happen if $B$ is kept LOW.

**OUTCOME ASSESSMENT QUESTIONS**

1. What is the only input combination that will produce a HIGH at the output of a five-input AND gate?
2. What logic level should be applied to the second input of a two-input AND gate if the logic signal at the first input is to be inhibited (prevented) from reaching the output?
3. *True or false:* An AND gate output will always differ from an OR gate output for the same input conditions.

## 3-5 NOT OPERATION

### OUTCOMES

*Upon completion of this section, you will be able to:*

- Define the NOT logic function.
- Write Boolean equations using the NOT function.
- Draw the logic symbol for the NOT function.
- Write a truth table describing the NOT function.
- Draw a timing diagram that demonstrates the NOT function.

The **NOT operation** is unlike the OR and AND operations because it can be performed on a single input variable. For example, if the variable $A$ is subjected to the NOT operation, the result $x$ can be expressed as

$$x = \overline{A}$$

where the overbar represents the NOT operation. This expression is read as "$x$ equals NOT $A$" or "$x$ equals the *inverse* of $A$" or "$x$ equals the *complement* of $A$." Each of these is in common usage, and all indicate that the logic value of $x = \overline{A}$ *is opposite to* the logic value of $A$. The truth table in Figure 3-11(a) clarifies this for the two cases $A = 0$ and $A = 1$. That is,

$$0 = \overline{1} \quad \text{because 0 is not 1}$$

and

$$1 = \overline{0} \quad \text{because 1 is not 0}$$

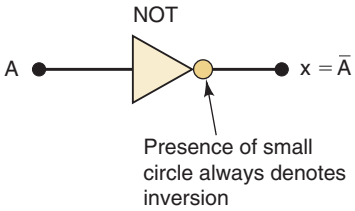
The NOT operation is also referred to as **inversion** or **complementation**, and these terms will be used interchangeably throughout the book. Although we will always use the overbar indicator to represent inversion, it is important

**FIGURE 3-11** (a) Truth table; (b) symbol for the INVERTER (NOT circuit); (c) sample waveforms.

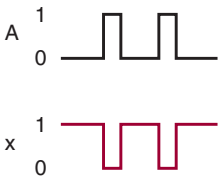
NOT

| A | x = $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

(a)




(b)




(c)

to mention that another indicator for inversion is the prime symbol ($'$). That is,

$$A' = \overline{A}$$

Both should be recognized as indicating the inversion operation.

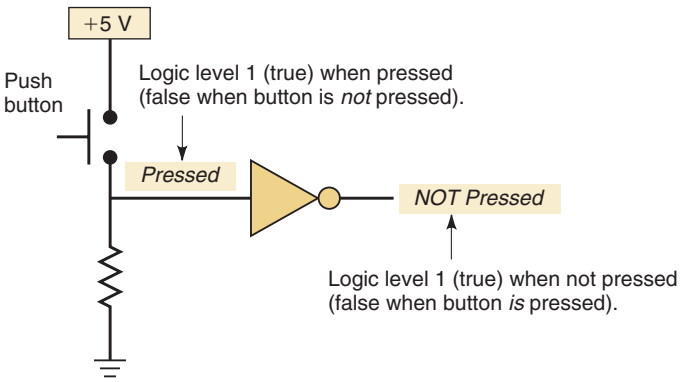
## NOT Circuit (INVERTER)

Figure 3-11(b) shows the symbol for a **NOT circuit,** which is more commonly called an **INVERTER**. This circuit *always* has only a single input, and its output logic level is always opposite to the logic level of this input. Figure 3-11(c) shows how the INVERTER affects an input signal. It inverts (complements) the input signal at all points on the waveform so that whenever the input = 0, output = 1, and vice versa.

**APPLICATION 3-1**

Figure 3-12 shows a typical application of the NOT gate. The push button is wired to produce a logic 1 (true) when it is pressed. Sometimes we want to know if the push button is not being pressed, and so this circuit provides an expression that is true when the button is not pressed.

**FIGURE 3-12** A NOT gate indicating a button is *not* pressed when its output is true.




## Summary of Boolean Operations

The rules for the OR, AND, and NOT operations may be summarized as follows:

| *OR* | *AND* | *NOT* |
|---|---|---|
| 0 + 0 = 0 | 0 · 0 = 0 | $\overline{0} = 1$ |
| 0 + 1 = 1 | 0 · 1 = 0 | $\overline{1} = 0$ |
| 1 + 0 = 1 | 1 · 0 = 0 | |
| 1 + 1 = 1 | 1 · 1 = 1 | |

**OUTCOME ASSESSMENT QUESTIONS**

1. The output of the INVERTER of Figure 3-11 is connected to the input of a second INVERTER. Determine the output level of the second INVERTER for each level of input *A*.
2. The output of the AND gate in Figure 3-7 is connected to the input of an INVERTER. Write the truth table showing the INVERTER output, *y*, for each combination of inputs *A* and *B*.

## 3-6 DESCRIBING LOGIC CIRCUITS ALGEBRAICALLY

### OUTCOME

*Upon completion of this section, you will be able to:*

- Translate logic diagrams into Boolean algebraic expressions.

Any logic circuit, no matter how complex, can be described completely using the three basic Boolean operations because the OR gate, AND gate, and NOT circuit are the basic building blocks of digital systems. For example, consider the circuit in Figure 3-13(a). This circuit has three inputs, *A*, *B*, and *C*, and a single output, *x*. Utilizing the Boolean expression for each gate, we can easily determine the expression for the output.

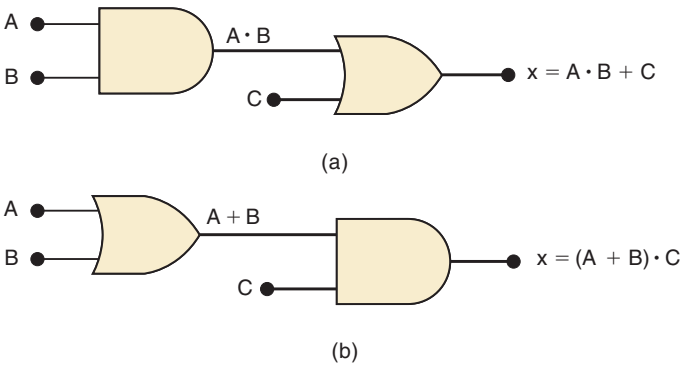



**FIGURE 3-13** (a) Logic circuit with its Boolean expression; (b) logic circuit whose expression requires parentheses.

The expression for the AND gate output is written $A \cdot B$. This AND output is connected as an input to the OR gate along with *C*, another input. The OR gate operates on its inputs so that its output is the OR sum of the inputs. Thus, we can express the OR output as $x = A \cdot B + C$. (This final expression could also be written as $x = C + A \cdot B$ because it does not matter which term of the OR sum is written first.)

### Operator Precedence

Occasionally, there may be confusion about which operation in an expression is performed first. The expression $A \cdot B + C$ can be interpreted in two different ways: (1) $A \cdot B$ is ORed with *C*, or (2) *A* is ANDed with the term $B + C$. To avoid this confusion, it will be understood that if an expression contains both AND and OR operations, the AND operations are performed first, unless there are *parentheses* in the expression, in which case the operation inside the parentheses is to be performed first. This is the same rule that is used in ordinary algebra to determine the order of operations.

To illustrate further, consider the circuit in Figure 3-13(b). The expression for the OR gate output is simply $A + B$. This output serves as an input