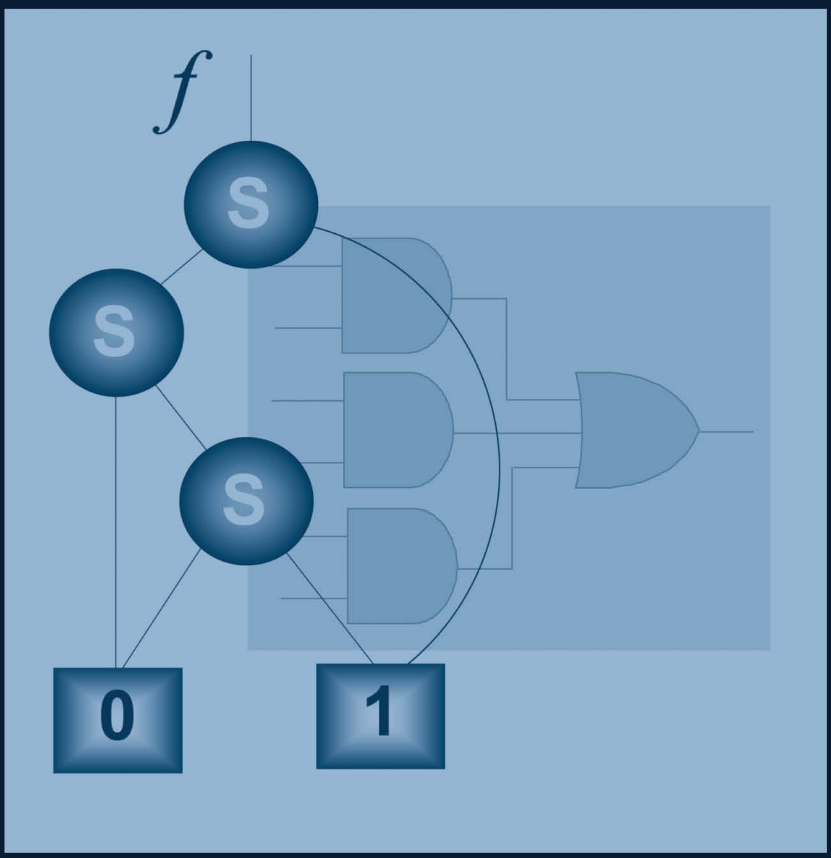
INTRODUCTION TO LOGIC DESIGN



Svetlana N. Yanushkevich
Vlad P. Shmerko

CRC Press
Taylor & Francis Group

INTRODUCTION TO
LOGIC DESIGN

INTRODUCTION TO LOGIC DESIGN

Svetlana N. Yanushkevich
University of Calgary, Alberta, Canada

Vlad P. Shmerko
University of Calgary, Alberta, Canada

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

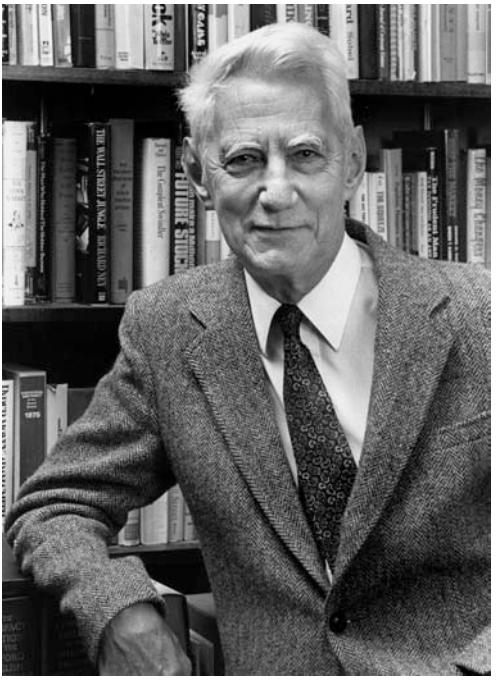
International Standard Book Number-13: 978-1-4200-6095-9 (Ebook-PDF)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

This textbook is dedicated to the memory of Claude Shannon (1916–2001)

Claude Elwood Shannon's inventive genius, probably more than that of any other single person, has altered man's understanding of communication and digital systems.



Contents

Preface

The Introduction to Logic Design Course in the Electrical Engineering and Computer Science Undergraduate Curriculum

An introductory course on the logic design of discrete devices is fundamental to the study of the many fields that constitute the ever-expanding discipline of computer engineering, computer science, and electrical engineering. Logic design of discrete devices serves as the prerequisite for additional coursework in the study of communications, signal processing, digital system design, and neural networks.

An introductory course on logic design is frequently encountered in the second year of undergraduate programs. Additional courses are then provided that expand on the one-semester course by including a more detailed treatment of digital system design, focusing, in particular, on simulation using hardware description languages.

With respect to the audience, that is, electrical and computer engineering and computer science, the introductory course on logic design can vary significantly. The course on logic design is usually offered at the sophomore or junior level and assumes no background on the part of the reader.

How This Book Satisfies the Essential Needs of This Course

Given the introductory nature of the computing device design course and the diversity of its applications, an appropriate textbook must be easy to read, accurate, and contain an abundance of insightful examples, problems, and computer experiments to expedite learning the fundamentals of logic design of discrete devices and systems. The textbook must reflect the fundamentals of state-of-the-art logic design and must be useful to people from other fields who are interested in the basics of logic network design. This book has been written with all of these objectives in mind.

We have verified in our academic and research work that the material presented in this book is useful for not only computer/electrical engineering and computer science students, but also students from physics and the

chemical sciences who are working on applications of physical and molecular phenomena at nanoscale for computing logic functions. This textbook attempts to fill this need with a balanced presentation of classic topics and topics of advanced design, such decision trees and diagrams, focusing on the manipulation of various data structures rather than optimization.

It is the primary purpose of this book to present problems relevance for advanced as well as future technologies that will be educationally informative for students and practicing engineers and researchers. The emphasis will be on the manipulation of various data structures in logic network design. Of course, knowledge of classical minimization techniques is also useful and necessary as background. Such techniques are also included in complete detail.

This book builds on the authors' lectures in various universities, providing a balanced and integrated treatment of the fundamental theory of logic functions and applications in the design of digital devices and systems. This approach has the pedagogical advantage of helping the student see the fundamental similarities and differences between various theoretical approaches to the representation, manipulation, and optimization of Boolean functions, and reflects the integrated nature of the advanced concepts in modern engineering practice.

The text has been written with the aim of offering maximum teaching flexibility in both coverage and order of presentation. A one-semester course sequence would likely cover most, if not all, of the topics in the book.

Structure Designed to Facilitate and Reinforce Learning

We designed this textbook to be up-to-date, comprehensive, and pragmatic in its approach. It aims to provide balanced coverage of concepts, techniques, and practices, and to do so in a way that is, at the same time, both scholarly and supportive of the needs of the typical student reader. We wanted to give students:

- A clear picture of fundamental concepts;
- Effective problem-solving techniques; and
- Appropriate exposure to modern technologies, design techniques, and applications.

Through the mutual support of textual material, worked examples, end-of-chapter problems, and other pedagogical features, we have attempted to show how theoretical ideas, physical devices, and design methodologies can all come together to form a successful design system.

Toward Predictable Technologies

With the advent of new technologies, a major shift has occurred in the field of the design and application of digital systems. As a result, many engineers

and scientists have found it necessary to understand the basic operation of digital systems and how these systems can be designed if they carry out particular information-processing tasks associated with their work. This trend has produced a need for an introductory undergraduate course in logic design to provide a unified overview of the interrelationship between digital system design, computer organization, micro- and nano-electronics, and numerical methods.

In this textbook, no specific prerequisites are assumed, nor is any knowledge of electrical circuits or electronics required. In this way, this textbook satisfies the requirements of interdisciplinary interest in logic design.

New Concepts in an Introductory Course

The approach taken in this book is a traditional one. That is, the emphasis is on the presentation of basic principles of logic design and the illustration of each of these principles. However, the following key features distinguish this textbook:

- This textbook presents a comprehensive catalog of logic network design techniques, including the most popular decision diagram techniques. Standard textbooks on logic circuit design generally do not discuss decision diagram techniques, either in their entirely or in a tutorial manner.
- A central role is reserved for data structures, as a key to applications.
- Recent key trends in the theory and practice of logic network design techniques are highlighted.
- Novel techniques of advanced logic design are presented with respect to new possibilities in predictable technologies.

In summary, this is a textbook of a new generation of texts for undergraduate students. Although classical in content, this book is different from other introductory books on logic design in emphasizing topics, such as design and data structures, and design and technological requirements. The relationships between various data structures and their manipulation through design represent the most important aspect of contemporary logic design.

This textbook is dedicated to the memory of Claude Shannon. Claude Elwood Shannon, an electrical engineer and mathematician, was a graduate of the University of Michigan, class of 1936. His legendary Master's thesis dealt with the application of Boolean logic to electronic relays, and ushered in the era of the deployment of Boolean logic to ever-evolving electrical circuits. Known for his mathematically rigorous approach, Shannon remarkably contributed to scientific knowledge in the fields of communication and logic design of discrete devices. He is widely regarded as the father of information theory, and his work helped lay the foundation for the electronic age. It is likely that techniques based on Shannon's information theory will

become some of the main design tools for nanosystems – computing systems for the age of nanotechnology.

We would like to acknowledge several people for their useful suggestions and discussions:

Acknowledgments

Dr. N. Bartley, University of Calgary, Canada
Dr. D. Bochmann, University of Chemnitz, Germany
Dr. J. T. Butler, Naval Postgraduate School, Monterey, CA, U.S.A.
Dr. G. Dueck, University of New Brunswick, Canada
Dr. D. Green, University of Manchester, UK
Dr. M. Kameyama, Tohoku University, Japan
Dr. T. Luba, Warsaw University of Technology, Poland
Dr. S. E. Lyshevski, Rochester Institute of Technology, Rochester, NY, USA
Dr. D. M. Miller, University of Victoria, Canada
Dr. C. Moraga, Dortmund University, Germany
Dr. J. Muzio, University of Victoria, Canada
Dr. M. Perkowski, Portland State University, OR, U.S.A.
Dr. S. Rudeanu, University of Bucharest, Romania
Dr. T. Sasao, Kyushu Institute of Technology, Japan
Dr. R. Stanković, University of Nis, Serbia
Dr. B. Steinbach, Freiberg University of Mining and Technology, Germany
Dr. A. Stoica, California Institute of Technology, U.S.A.
Dr. H. Watanabe, Soka University, Tokyo, Japan

Also, we would like to acknowledge the efforts of the staff at CRC Press, in particular, Nora Konopka and Prudence Board. We would like to thank our undergraduate and graduate students, and also Ian Pollock, for their valuable suggestions and assistance, which were helpful to us in ensuring the coherence of topics and material delivery "synergy."

Svetlana N. Yanushkevich

Vlad P. Shmerko

Calgary, Canada

1

Design Process and Technology

Design process

- Design levels
- Design hierarchy
- Top-down design methodology
- Bottom-up design methodology

Implementation technologies

- Very large scale integration
- Deep submicron integration

Data structures

- Algebraic
- Graphical

Design methodologies

- Application-specific integrated circuits
- System-on-chip

Predictable technologies

- Energy-efficient design
- Nanoelectronics
- Molecular devices

1.1 Introduction

Digital logic networks are used in all devices that process information in digital form. *Information* – recorded or communicated facts, or *data* – takes a variety of physical forms when being stored, communicated, or manipulated. Information on the nature of a physical phenomenon is conveyed by signals that assume a finite number of discrete values, that is, it is expressed as a finite sequence of symbols. A *signal* is defined as a function of one or more variables, and a *system* is defined as an entity that manipulates one or more signals to accomplish a function, thereby yielding new signals.

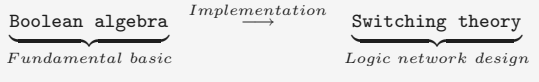
In this book, each signal is assumed to have only one of two values, denoted by the symbols 0 and 1. If the signals are constrained to only two values, the system is *binary*.

There are many methodologies for discrete system design. A *discrete system* is a combination of logic networks and discrete devices that is assembled to accomplish a desired result, such as the computing and transferring of data. Discrete system are classified using various criteria; in particular, with respect to applications, power consumption, requirements for reliability, performance, technological criteria, etc.

1.2 Theory of logic design

In 1854, George Boole introduced a systematic treatment of logic in the work *An Investigation of the Laws of Thought* and developed for this purpose an algebraic system, now called *Algebra*. Boole's goal was the development of a formalism to compute the truth or falsehood of complex compound statements from the truth values of their component statements.

Boolean algebra was applied by C. Shannon in 1938 to relay-contact networks, the first switching circuits. This theory is called *switching theory* and has been used ever since in the design of digital *logic networks* or *logic circuits*. Since then the technology has gone from relay-contacts through diode gates, transistor gates, integrated circuits, to future nanotechnologies, and still Boolean algebra is its fundamental and unchanging basis:



Modern logic design includes methods and techniques from various fields, in particular:

Methods and techniques of modern logic design

- *Digital signal processing* is adopted in logic design for efficient manipulation of data.
- *Communication theory* to solve communication problems between computing components in logic networks.
- *Artificial intelligence methods* and techniques for optimization at logic and physical levels of logic network design.

As far as predictable technologies are concerned, non-traditional computing paradigms that are based on various physical and chemical phenomena are studied in logic design. The assumed stochastic nature of many processes at nanoscale implies that *random* signals must be used instead of *deterministic* signals. Random signals take on random values at any given moment in time and must be modeled probabilistically, while deterministic signals can be modeled as completely specified functions of time. The theoretical base of probabilistic logic signals is called *probabilistic logic*. Schematically, advanced logic design can be viewed as follows:

1.3 Analysis and synthesis

The *specification* of a system is defined as a description of its function. The *implementation* of a system refers to the construction of a system (Figure 1.1). The *analysis* of a system has as its objective to determine its specification from the implementation:

$$\texttt{Logic network} \overset{Analysis}{\longrightarrow} \texttt{Specification}$$

Synthesis, or *design*, consists of obtaining an implementation that satisfies the specification of the system:

$$\texttt{Specification} \overset{Synthesis}{\longrightarrow} \texttt{Logic network}$$

The central task in logic synthesis is to optimize the representation of a logic function with respect to various criteria. In Figure 1.1, analysis and synthesis are shown as the relationship between the *specification* and *implementation* of a system.

A system can be examined at various levels of abstraction. Each such level is called a *design level* (Figure 1.2a). The following design levels are distinguished:

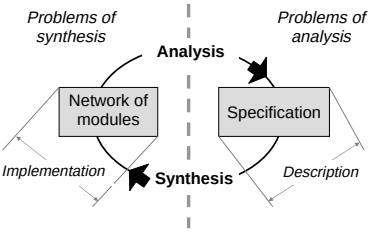


FIGURE 1.1
Design of discrete systems.

- The top design level is called the *architectural* or *system* level.
- The intermediate design level is called the *logic* level; this level is the subject of the present book.
- The bottom design level is called the *physical* level; this level is concerned with the details needed to manufacture or assemble the system.

At the physical level, a system is implemented by a complex interconnection of elements such as transistors, resistors, etc. Because of this complexity, it is impractical to perform design and optimization at this level, motivating a move to the intermediate level of design. At the intermediate level, a modular structure provides a reasonable simplification of design. *Libraries* of standard modules significantly simplify the design of different systems. Assembling modules of increasing complexity into higher hierarchical blocks is achieved at the system level.

1.3.1 Design hierarchy

A *hierarchical* approach to digital system design aims at

- Reducing the cost of the design of a system, and
- Improving the quality of the solutions obtained.

The hierarchical approach to design makes a large system more manageable by reducing complexity and introducing a rational *partitioning* of the design processes. They can be designed, tested, and manufactured separately. This is the basis for *standardization*. The specified components can be mass produced at relatively low cost. In the design process, these components can be composed in standard *libraries* and reused with minor modifications. This facilitates the reduction of overall design time and cost. The *robustness* of the hierarchical approach provides many possibilities for avoiding design errors, design corrections, and repairs after manufacture.

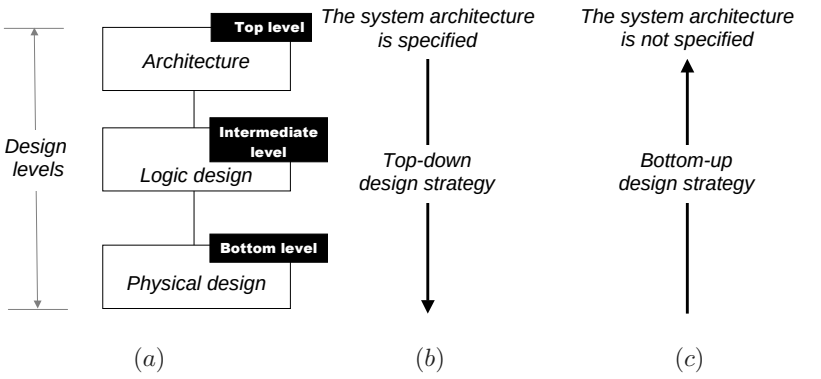


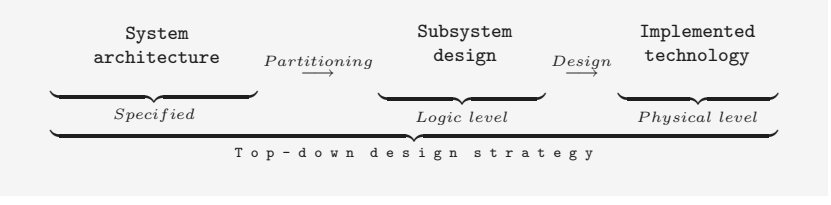
FIGURE 1.2
Design hierarchy (a), top-down (b), and bottom-up (c) design strategies.

Any design process includes a *design loop* that provides the possibility to carry out a *redesign* if errors are detected in simulation. This loop is repeated until the simulation indicates a successful design.

1.3.2 Design methodology

Top-down design methodology

A design that evolves from a generalized or abstract point of view and proceeds in steps to specific components is referred to as a *top-down design methodology* (Figure 1.2b):



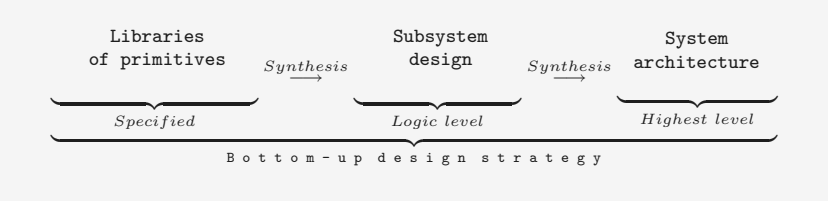
In this approach, the hierarchy tree is traversed from top to bottom. The system architecture is specified at the highest level first. The disadvantage of this approach is that no systematic procedure exists for optimization of the final implementation; that is, optimization at one particular level does not guarantee an optimal final solution. The success of the approach depends mainly on the experience and professional skills of the designer.

An example of an advanced top-down methodology is so-called *platform-based design*. A platform is defined as a family of the designs and not a

single design. In this top-down process, the constraints that accompany the specification are mapped into constraints on the components of the platform.

Bottom-up design methodology

An alternative process to the top-down approach is a *bottom-up design methodology* (Figure 1.2c):



This is the reverse of the top-down design process. One starts with specific components in mind and proceeds by interconnecting these components into a generalized system. In a bottom-up design approach, the components at or near the lowest design level of the hierarchy tree are designed first. The architecture of the entire system is not specified until the top of the tree is reached. Unfortunately, in general, there is no systematic technique that results in correct system specification.

1.3.3 Design styles

In a general sense, the following design styles are distinguished:

Digital device design styles

Full-custom design; this style provides freedom to the designer and is characterized by great flexibility; however, this style is not acceptable for the design of large systems.

Semi-custom design provides more possibilities for automation using, in particular, standardization; such as a *library* of standard cells.

Mixed design styles often provide an acceptable reduction to the flexibility of full-custom style, while opening up possibilities for the automation and optimization of semi-custom style.

Gate-array design meets the requirements of fabrication and simplifies the optimization problem; this style results in regular structures within the chip, that is, connected cells that are placed on a chip in a regular way.

1.3.4 Simulation

The use of software simulation is an important part of any modern design process. The primary uses of a simulator are to check a design for functional correctness and to evaluate its performance. Simulators are key tools in determining whether design goals have been met and whether redesign is necessary. More details on sample simulation tools are given in Chapter 19.

1.4 Implementation technologies

The scaling of microelectronics down to nanoelectronics is the inevitable result of technological evolution (Figure 1.3). The most general classification of the trends in technology is based on grouping computers into *generations*. Using this criterion, five generations of computers are distinguished (Table 1.1). Each computer generation is 8 to 10 years in length.

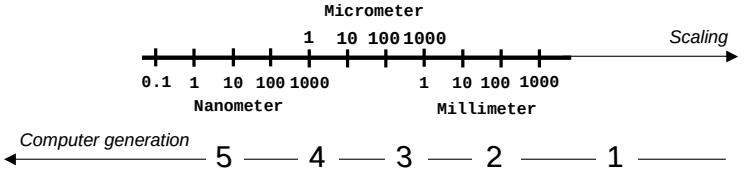


FIGURE 1.3
Progress from micro- to nanosize in computing devices.

TABLE 1.1
Computer generations are determined by the change in the dominant technology.

Generation	Dates	Technology
1	1950–1964	Vacuum tubes (zero-scale integration)
2	1965–1969	Transistors (small-scale integration)
3	1970–1979	Integrated circuits (medium-scale integration)
4	1980–2004	Large, very large, ultra large-scale integration
5	2005–	Nanotechnology (giga-scale integration)

The following can be compared against this scale:

The scaling of microelectronics down to nanoelectronics

- The size of an atom is approximately 10^{-10} m. Atoms are composed of subatomic particles; e.g., protons, neutrons and electrons. Protons and neutrons form the nucleus, with a diameter of approximately 10^{-15} m.
- 2-dimensional molecular assembly (1 nm).
- 3-dimensional functional nanoICs topology with doped carbon molecules ($2 \times 2 \times 2$ nm).
- 3-dimensional nanobioICs (10 nm).
- *E.coli* bacteria (2 mm) and ants (5 mm) have complex and high-performance integrated nanobiocircuitry.
- 1.5×1.5 cm 478-pin Intel® Pentium® processor with millions of transistors, and Intel 4004 Microprocessor (made in 1971 with 2,250 transistors).

Binary states are encountered in many different physical forms in various technologies. The standard approach is to use the digit symbols 0 and 1 to represent the two possible values of binary quantity. These symbols are referred to as *bits*.

Binary arithmetic has the following advantages:

(*a*) It can be implemented using on-off switches, the simplest binary devices.

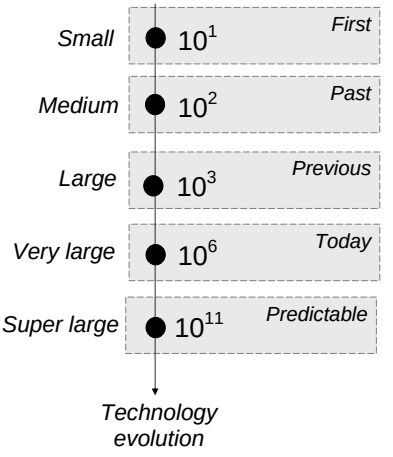
(*b*) It provides for the simplest decision-making such as YES (1) and NO (0).

(*c*) Binary signals are more reliable than those formed by more than two quantization levels.

Significant evolutionary progress has been achieved in microelectronics. This progress (miniaturization, optimal design and technology enhancement) has been achieved by scaling down microdevices, approaching 45 nm sizing features for structures, while increasing the integration level (Figure 1.4). Complementary metal-oxide semiconductor (CMOS) technology is being enhanced, as nanolithography, advanced etching, enhanced deposition, novel materials, and modified processes are all currently used to fabricate ICs. The channel length of metal-oxide-semiconductor field effect transistors (MOSFETs) has decreased from

- 50 μm in 1960, to
- 1 μm in 1990, and to
- 130 nm in 2004, 65 nm in 2006, and 45 nm in 2007.

This progress in miniaturization and integration can be observed, for example, on Intel processors:



Small-scale integration (SSI):

- *1960s, dozens of gates in a package*

Medium-scale integration (MSI):

- *1970s, hundreds of gates in a package*

Large-scale integration (LSI):

- *1980s, thousands of gates in a package*

Very large-scale integration (VLSI):

- *1980s* and *1990s, hundreds of thousands of gates in a package*

Giga-scale integration:

- *Today, millions of gates in a package*

Tera-scale integration:

- *Expected, hundred millions of gates in a package*

FIGURE 1.4
Evolution of technologies: levels of integration of chips.

1971–1982: From Intel 4004 (1971, 2,250 transistors), to Intel 286 (1982, 120,000 transistors),

1993–2007: From Pentium (1993, 3,100,000 transistors), to Pentium 4 (2000, 42,000,000 transistors), to Itanium™ 2 Processor (2002), Pentium® M Processor (2003) with hundreds of millions of transistors, and Pentium Dual-Core in 2007.

Typical signal integrity effects include interconnect delay, crosstalk (in closely coupled lines the phenomenon of crosstalk can be observed), power supply integrity, and noise-on-delay effects. In the early days of very large-scale integration (VLSI) design, these effects were negligible because of relatively slow chip speed and low integration density. However, with the introduction of technological generations working at about 0.25 μm scale and below, there have been many significant changes in wiring and electrical performance.

As the number of computational and functional units on a single chip increases, the need for communication between those units also increases. Interconnection has started to become a dominant factor in chip performance. As chip speed continually increases, the increasing inductance of interconnections affects the signal parameters.

Noise is a deviation of a signal from its intended or ideal value. Most noise in VLSI circuits is created by the system itself. Electromagnetic interference is an external noise source between subsystems. In deep submicron circuits,

the noise created by parasitic components with digital switching exhibits the strongest effect on system performance.

Programmable versus ASIC methodology

Application-specific integrated circuits (ASIC) design methodology has been very successful in a wide range of applications. The system-on-chip (SoC) is where the integration of a complete electronic system including all its periphery occurs. The reduction in size and cost that would come with this high level of integration opens the door to truly ubiquitous electronics.

1.5 Predictable technologies

The key to the predictable technologies of the future is changing the *computing paradigm*, that is, the model of computation and the physical effects for the realization of this model:

$$\underbrace{\texttt{Computing model}}_{Design} \overset{Implementation}{\longrightarrow} \underbrace{\texttt{Physical effects}}_{Technology}$$

There are fundamental technological differences among

- Nanoelectronic devices vs. microelectronic ones (which can even be nanometers in size),
- Nanoelectronics vs. microelectronics; e.g., nano integrated circuits vs. integrated circuits.

These enormous differences are due to differences in basic physics and other phenomena. The dimensions of nanodevices that have been made and characterized are a hundred times less than even newly designed microelectronic devices (including nanoFETs with 10 nm gate length). Nanoelectronics sizing leads to volume reduction by a factor of 1,000,000 in packaging, not to mention revolutionary enhanced functionality due to multiterminal and spatial features. For example, molecular electronics focuses on the development of electronic devices using small molecules with feature sizes on the order of a few nanometers.

A wide variety of factors, such as voltage scaling and thermal noise, dramatically reduce the reliability of integrated systems. In nanotechnologies, process variations and quantum fluctuations occur in the operations of very-deep submicron transistors. Any computer with nanoscale components will contain a significant number of defects, as well as massive numbers of wires and switches for communication purposes. It therefore makes sense to consider architectural issues and defect tolerance early in the development of a new paradigm.

Two main aspects are critical to the design of nanodevices: *the probabilistic behavior* of nanodevices (electrons, molecules); this means that a valid switching function can be calculated with some probability; and *the high defect rates* of nanodevices; this means that because many of the fabricated devices have defects, their logic correctness is distorted.

There are two types of fault tolerances exhibited by a nanosystem: fault tolerance with respect to (a) data that are noisy, distorted, or incomplete, which results from the manner in which data are organized and represented in the nanosystem, and (b) physical degradation of the nanosystem itself. If certain nanodevices or parts of a network are destroyed, the network will continue to function properly. When the damage becomes extensive, the network will only affect the system's performance, as opposed to causing a complete failure. Self-assembling nanosystems are capable of this type of fault tolerance because they store information in a distributed (redundant) manner, in contrast to traditional storage of data in a specific memory location in which data can be lost in the case of a hardware fault.

The methods of stochastic computing provide another approach to overcoming the problem of the design of reliable computers from unreliable elements, i.e., nanodevices. For example, a signal may be represented by the probability that a logic level has a value of 1 or 0 at a clock pulse. In this way, random noise is deliberately introduced into the data. A quantity is represented by a clocked sequence of logic levels generated by a random process. Operations are performed via completely random data.

1.6 Contemporary CAD of logic networks

The goal of the computer aided design (CAD) of logic network tools is to automatically transform a description of logic networks in the algorithmic or behavioral domains to one in the physical domain, i.e., down to a layout mask for chip production.

A CAD system has to produce logically correct circuits correct results, but because of the complexity of the design process, verifying the correctness of results is a necessary phase of design. Usually, formal verification techniques deal with different data structures and descriptions. Computer aided design tools are intended to support all phases of digital design: description (specification), synthesis (design), including various optimizations to reduce cost and improve performance, and verification of the design with respect to its specification. A CAD system for logic synthesis and analysis employs a suitable representation that allows Boolean functions to be manipulated. These representations are realized internally by means of suitable *data structures*.

Data structures are used for the description or specification of a digital system at various levels. The first approach to describing digital systems consists of a description of their structure through graphical data structures. This description provides a *logic diagram* of the system at different levels, showing the modules and their interconnections. This process is supported by libraries of standard components. The second approach is the use of *hardware description language* (HDL). Both approaches are considered throughout this textbook.

1.7 Summary of design process and technology

This chapter introduces the strategic approaches undertaken in the contemporary design of discrete devices and systems. These approaches and trends are being directed by advances in technology. The key aspects of this chapter are as follows:

(*a*) The ***hierarchical*** approach to design makes a large system more manageable by reducing complexity. In this approach, components can be designed, tested, and manufactured separately. ***Standard*** components can be mass produced at relatively low cost. This facilitates the reduction of overall design time and cost.

(*b*) Both ***top-down*** and ***bottom-up*** design methodologies are employed. A design that evolves from an abstract point of view and proceeds to specific components is a top-down design methodology. In this approach, the system architecture is specified at the highest level first. An alternative process, bottom-up design methodology, requires the components at or near the lowest design level of the hierarchy tree to be designed first.

(*c*) ***Data structures*** are the fundamental components of logic design. No universal data structure exists that can be efficient in all applications. Choosing an appropriate data structure based on its advantages and disadvantages, as well as the relationships between these structures, in order to satisfy design requirements, are crucial points of logic design.

Summary (continuation)

The topics of this chapter are summarized as follows:

- ***Information*** – recorded or communicated facts or ***data*** – takes a variety of physical forms when being stored, communicated, or manipulated. Digital logic networks are characterized by the fact that signals assume a finite number of discrete values. A ***signal*** is defined as a function of one or more variables that conveys information on the nature of a physical phenomenon. A ***system*** is an entity that manipulates one or more signals to accomplish a function, thereby yielding new signals.
- ***Logic design*** emphasizes the development of techniques for the design of logic networks. The basis of logic design is two-valued ***Boolean algebra***, also called ***switching algebra***. Modern logic design includes methods and techniques from various fields.
- The ***specification*** of a system is defined as a description of its function, including various parameters for the evaluation of a system. The ***implementation*** refers to the construction of the system. The ***analysis*** of the system has as its objective the determination of the system's specification from its implementation.
- Complex systems are modeled at various levels of detail, called ***design levels***: (*a*) ***architectural*** or ***system*** level, (*b*) ***logic*** level, and (*c*) ***physical*** level.
- The following design styles are distinguished: (*a*) ***full-custom*** design (provides freedom to the designer), (*b*) ***semi-custom*** design (more possibilities for automation), and (*c*) ***mixed*** design style and ***gate-array*** design (meets the requirements of fabrication).
- Progress in microelectronics striving for miniaturization, optimal design, and technology enhancement is achieved by the scaling down of microdevices. For example, the channel length of metal-oxide-semiconductor field effect transistors (MOSFETs) decreased from 50 μm in 1960 to 1 μm in 1990, and to 45 nm in 2007.
- New horizons in design processes and technology are introduced in the next section, "Further study." For instance, an essential feature of logic design for predictable technologies of the future is that ***random*** signals be used instead of ***deterministic*** signals. Random signals are signals that take on random values at any given instant and must be modeled ***probabilistically***.

1.8 Further study

Historical perspective

1930: Alan Turing is often considered to be the father of modern computer science. Turing provided an influential formalisation of the concept of the algorithm and computation with so-called *Turing machine*. *Turing test* contributed to the debate regarding artificial intelligence: whether it will ever be possible to say that a machine is conscious and can think. During the Second World War Turing worked at Bletchley Park, Britain's codebreaking centre, the section responsible for German naval cryptanalysis. He devised a number of techniques for breaking German ciphers.

1947: A device called a *transistor*, which has several applications in radio where a vacuum tube ordinarily is employed, was demonstrated for the first time at Bell Telephone Laboratories.

1955: Von Neumann's classic work on probabilistic logic and reliable computation upon nonreliable computing elements: "Probabilistic logics and the synthesis of reliable organisms from unreliable components", in C. E. Shannon and J. McCarthy, Eds., *Automata Studies,* pages 329–378, Princeton University Press, Princeton, NJ, 1955.

1958: Jack S. Kilby, employee at Texas Instruments, put all the circuit elements – transistors, resistors, and capacitors, along with their interconnecting wiring – into a single piece of germanium. His rough prototype was a thin piece of germanium about one-half inch long containing five separate components linked together by tiny wires. Since then, the first commercial ICs began to emerge at the beginning of 1960.

1968: Robert Noyce and Gordon Moore established Intel in Santa Clara, California. Intel produced the first 1K RAM (random access memory). Gordon Moore formulated an empirical law stating that the performance of an integrated circuits, including the number of components on it, doubled every 18–24 months with the same chip price. This became known as *Moore's rule*. It is still holding up.

1971: Ted Hoff at Intel invented Intel's first microprocessor (4004). The 4-bit 4004 ran at 108 kHz and contained 2300 transistors. In 1972, the 8008 microprocessor was developed, twice as powerful as the 4004.

1980: The first programmable integrated circuits were developed. These devices contain circuits whose logical function and connectivity can be programmed by the user, rather than being fixed by the integrated circuit manufacturer. This allows a single chip to be programmed to implement different logic functions. Current devices, called field programmable gate arrays (FPGAs) can now implement tens of thousands of logic networks in parallel and operate at up to 550 MHz.

1981: IBM PC was released.

1985: The Intel 80386 32-bit microprocessor featured 275,000 transistors – more than 100 times as many as the original 4004. It could run multiple programs simultaneously.

1985: David Deutsch described how a computer might run using rules of quantum mechanics and why such a computer differs fundamentally from ordinary computers.

1989: The Intel 486TM processor was the first built-in math coprocessor, which speeded up computing because it offered complex math functions from the central processor, greatly speeding up transcendental functions.

1995: Released in the fall of 1995 the Pentium Pro processor, with 5.5 million transistor and with a second speed-enhancing cache memory chips, was designed to support 32-bit server and workstation-level applications, enabling fast computer-aided design, mechanical engineering and scientific computation.

1997: The 7.5 million-transistor Pentium II processor incorporated Intel MMX technology, which was designed specifically to process video, audio and graphics data efficiently.

2003: The 130 nanometer technology was announced.

2005: The 65 nanometer chip manufacturing process was announced. AMD and NEC have started using a 65 nanometer process.

2007: Intel, IBM, NEC, and AMD started using 45 nanometers for their CPU chips.

Advanced topics of logic design and technology

Topic 1: Specific-area design. The theory and techniques for logic network design, developed to solve problems of computation and information processing, are being very rapidly adopted by those solving problems of communication, control, and artificial intelligence for various specific-area applications such as humanoid robotics, decision-making support systems, bio-medical applications, and biometric-based security systems (in particular, artificial intelligence, encryption, watermarking, and humanoid robotics).

Topic 2: Energy-efficient design of logic networks requires specialized techniques and tools. Mobile devices contain integrated circuits and employ battery-powered systems. The lifetime of the battery decreases as the power consumption of the integrated circuits grows. In energy-efficient design techniques, the most significant power savings are achieved at high levels of abstraction and during early phases of the design. In particular, portable computers, mobile navigation, and robotics. One possible the approach to energy-efficient design is the minimization of the *switching activity* of logic networks. Switching activity is defined as the expected number of logic transitions during one clock cycle.

Topic 3: Trends and potential application for nanoscale devices design. Selected topics in the theory and technique of logic network design are found useful in the design of computing devices based on nanotechnology. As a result, researchers from physics and the chemical sciences have an urgent desire and need to become familiar with the theory and practice of contemporary

logic design. For earlier technologies, the relevant problems were primary concerned with component minimization. With the development of VLSI and ULSI logic network technologies and the advent of nanoscale technologies in digital system design, the problems concerned with the minimization of components have become less relevant. These types of problems have been replaced by less well defined and much more difficult problems, such as physical design including partitioning, layout and routing, structural simplicity, and uniformity of modules. The last problem, uniformity, and related data structures are of particular interest in logic design in nanoscale. Many of these problems cannot be solved based on traditional approaches. However, a great amount of theoretical and practical work has been done in these areas.

Further reading

A. Advanced logic design

Digital Design: Principles and Practices by Richard S. Sandige, Prentice Hall, 2002.

Digital Design Essentials by John F. Wakerly, Prentice Hall, 2001.

"Electronic Design Automation at the Turn of Century", special issue of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 19, number 12, 2000.

Fundamentals of Logic Design by Jr. C. H. Roth, 5th Edition, Thomson Brooks/Cole, 2004.

Fundamentals of Digital Logic with VHDL Design by S. Brown and Z. Vranesic, New York, McGraw-Hill, 2000.

Introduction to Digital Logic Design by John P. Hayes, Addison-Wesley, 1993.

Introduction to Digital Systems by M. D. Ercegovac, T. Lang, and J. H. Moreno, John Wiley & Sons, 1999.

"Logic Design of Computational Nanostructures" by S. Yanushkevich, *Journal of Computational and Theoretical Nanoscience*, American Scientific Publishers, volume 4, number 3, pages 384–407, May 2007.

"Logic Design of Nanodevices" by S. Yanushkevich, In: *Handbook of Theoretical and Computational Nanotechnology* by M. Rieth, and W. Schommers, Editors, Scientific American Publishers, Chapter 110, pages 1–52, 2007.

Logic Design Principles: With Emphasis on Testable Semicustom Circuits by Edward J. McCluskey, Prentice Hall, 1986.

Logic Synthesis and Verification edited by S. Hassoun and T. Sasao, Consulting Editor R. K. Brayton, Kluwer, 2002.

Synthesis and Optimization of Digital Circuits by G. De Micheli, McGraw-Hill, 1994.

Switching Theory for Logic Synthesis by T. Sasao, Kluwer, 1999.

"The Future of Logic Synthesis and Verification" by R. K. Brayton, in *Logic Synthesis and Verification* Edited by S. Hassoun and T. Sasao, Consulting Editor R. K. Brayton, Kluwer, Dordrecht, 2002.

B. Predictable technologies and computing devices

"Computing with Molecules" by M. A. Reed and J. M. Tour, *Scientific American*, pages 86–93, June 2000.

Introduction to Nanotechnology by C. P. Jr. Poole and F. J. Owens, John Wiley & Sons, New York, 2003.

Logic Design of NanoICs by S. Yanushkevich, V. Shmerko, and S. E. Lyshevski, CRC Press, Boca Raton, FL, 2005.

"The Topsy Turvy World of Quantum Computing" by J. Mullins, *IEEE Spectrum*, pages 42–49, February 2001.

Molecular Electronics, Circuits and Processing Platforms by S. E. Lyshevski, CRC Press, Taylor & Francis Group, Boca Raton, FL, 2007.

"Ultimate Theoretical Models of Nanocomputers" by M. P. Frank and T. F. Jr. Knight, *Nanotechnology*, volume 9, pages 162–176, 1998.

2

Number Systems

Computer Arithmetic

- Binary
- Octal
- Decimal
- Hexadecimal

Binary arithmetic

- Sign and magnitude
- 1's complement
- 2's complement
- Addition and subtraction

Residue arithmetics

- Modular adder
- Modular subtractor
- Modular multiplier

Binary codes

- Gray code and Hamming distance
- Binary-coded decimal codes
- Weighted codes

Advanced topics

- Number systems and cryptography
- Numbers and information

2.1 Introduction

The binary number system is the most important number system in digital design. This is because it is suited to the binary nature of the phenomena used in dominant microelectronic technology. Even in situations where the binary number system is not used as such, binary codes are employed to represent information at the signal level. For example, multi-valued logic values are often encoded using binary representations. However, humans prefer decimal numbers, – thus, that is, binary numbers must be converted into decimal numbers.

In this chapter, various number systems are examined that are used in digital data structures. These number systems, such as octal and hexadecimal, are used to simplify the manipulation of binary numbers.

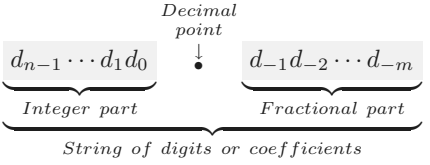
2.2 Positional numbers

A number system is defined by

- Its basic symbols, called *digits* or *numbers*, and
- The ways in which the digits can be combined to represent the full range of numbers we need.

2.2.1 The decimal system

The ten digits $0, 1, 2, \ldots, 9$ can be combined in various ways to represent any number. The fundamental method of constructing a number is to form a *sequence* or *string* of digits or *coefficients*:



where integer and fractional parts are represented by n and m digits to the left and to the right of the *decimal point*, respectively. The subscript $i = -m, m-1, \ldots, 0, 1, \ldots, n$ gives the position of the digit. Depending on the position of digits in the string, each digit has an associated value of an integer raised to the power of 10 as follows:

The decimal system

$$
\begin{aligned}
N &= \underbrace{d_{n-1}d_{n-2}\cdots d_1 d_0 \overset{\text{Decimal point}}{\overset{\downarrow}{\bullet}} d_{-1}d_{-2}\cdots d_{-m}}_{String\ of\ coefficients} \\
&= \underbrace{d_{n-1}\times 10^{n-1} + d_{n-2}\times 10^{n-2} + \cdots + d_1 \times 10^1 + d_0 \times 10^0}_{Computing\ the\ integer\ part} \\
&= \underbrace{d_{-1}\times 10^{-1} + d_{-2}\times 10^{-2} + \cdots + d_{-m} \times 10^{-m}}_{Computing\ the\ fractional\ part} \\
&= \sum_{i=-m}^{n-1} d_i 10^i
\end{aligned}
$$

This method of representing numbers is called the *decimal system*. In the positional representation of digits:

Positional representation

- Each digit has a fixed value, or *weight*, determined by its position.
- All the weights used in the decimal number system are powers of 10.
- Each decimal digit d_i ranges between 0 and 9.
- The weighting of the digits is defined relative to the *decimal point*. This symbol means that digits to the left are weighted by positive powers of 10, giving integer values, while digits to the right are weighted by negative powers of 10, giving fractional values.
- Fractions are denoted by sequences of digits whose weights are negative powers of 10.

Example 2.1 (Decimal numbers.) *The four digits in the number 2008 represent, from left to right, thousands (digit 2), hundreds (number 0), tens (number 0), and ones (number 8). Hence, this four-digit number can be represented in the following form:*

$$2008 = 2 \times \overset{Weight}{10^3} + 0 \times \overset{Weight}{10^2} + 0 \times \overset{Weight}{10^1} + 8 \times \overset{Weight}{10^0}$$

Practice problem 2.1. (Decimal numbers.) Write the decimal number 747 in positional form.
Answer: $747 = 7 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$.

Example 2.2 (Integer and fraction.) *The decimal number 12.3456 consists of an integer part (12) and a fractional part (3456) separated by the decimal point. Thus, this number can be represented in the following form:*

$$12.3456 = \underbrace{1 \times 10^1 + 2 \times 10^0}_{Integer\ part} + \underbrace{3 \times 10^{-1} + 4 \times 10^{-2} + 5 \times 10^{-3} + 5 \times 10^{-4}}_{Fractional\ part}$$

The number 0.34_{10} *is represented as*

$$N = \sum_{i=-2}^{-1} 10^i \times d_i = 3 \times 10^{-1} + 4 \times 10^{-2} = {}^{34}/_{100}$$

Practice problem 2.2. (Fraction.) What does the number 7.53_8 represent?
Answer is given in "Solutions to practice problems."

2.2.2 Number radix

In general, an n-digit number in radix r consists of n digits, each taking one of r values: $\underbrace{0,\ 1,\ 2,\ \ldots,\ r-1}_{Radix\ r\ system}$. A general number N in a positional number system is represented by the following formula (Figure 2.1):

Number radix

$$N = \overbrace{\underbrace{a_{n-1}a_{n-2}\cdots a_1a_0}_{Integer\ part} \overset{\downarrow}{\bullet} \underbrace{a_{-1}a_{-2}\cdots a_{-m}}_{Fractional\ part}}^{n+m\ digits\ (Radix\ point)}$$

$$= \underbrace{a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \cdots + a_1 \times r^1 + a_0 \times r^0}_{Integer\ part}$$

$$+ \underbrace{a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdots + a_{-m} \times r^{-m}}_{Fractional\ part} = \sum_{i=-m}^{n-1} a_i r^i \quad (2.1)$$

where a_i denotes a digit in the number system such that

$$0 \le a_i \le (r-1),$$

where r is the base of the number system, n is the number of digits in the integer part of N, and m is the number of digits in the fractional part of N.

The integer part is separated from the fractional part by the *radix point*. The digits a_{n-1} and a_{-m} are referred to as the *most significant digits* (MSD) and the *least significant digits* (LSD) of the number N, respectively.

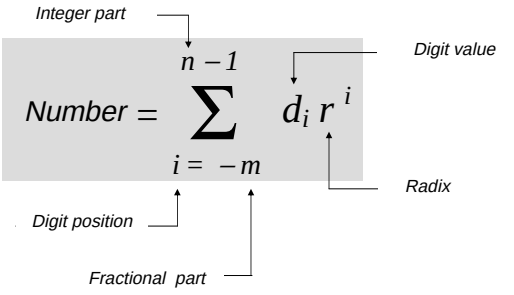


FIGURE 2.1
Number representation in the positional system.

A number system is said to be of *base*, or *radix* r, because the digits are multiplied by powers of r, and this system uses r distinct digits.

> **Example 2.3 (Radix.)** *The decimal number system is said to be of* base, *or* radix, 10, *because the digits are multiplied by powers of 10, and this system uses 10 distinct digits.*

To avoid possible confusion, the radix of the number system is often written as a decimal subscript appended to the number; that is, the subscript is placed after the LSD to indicate the radix of the number. When the context makes the radix obvious, it is not necessary to indicate the radix.

> **Example 2.4 (Radix.)** *The radix of the number system is written as a decimal subscript as follows. The binary ($r = 2$) number 10110 can be written in the form* 10110_2. *The octal ($r = 8$) number 67344.25 is indicated in the form* 67344.25_8. *The decimal ($r = 10$) number 67390.845 is indicated in the form* 67390.845_{10}.

Equation 2.1 is used in number representation as follows:

Algorithm for representing a number in the radix r system

Step 1. Choose the radix r of a number system
Step 2. Choose the number of digits n in the integer part of N
Step 3. Choose the number of digits m in the fractional part of N
Step 4. Write the number N in the radix r number system

Example 2.5 (Positional number systems.) *In Table 2.1, the most useful number systems are listed. Observe that for those number systems of base less than 10, a subset of the digit symbols of the decimal number system is used. For example, maximal two digit numbers in various radix are as follows:* $11_2 = 3_{10}$, $22_3 = 8_{10}$, $33_4 = 15_{10}$, $77_8 = 63_{10}$, 99_{10}, $FF_{16} = 16 \times F + F$.

TABLE 2.1
The most important positional number systems used in data representation and computing (Example 2.5).

Base r	Number system	Digit symbols
2	Binary	0,1
3	Ternary	0,1,2
4	Quaternary	0,1,2,3
8	Octal	0,1,2,3,4,5,6,7
10	Decimal	0,1,2,3,4,5,6,7,8,9
16	Hexadecimal	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Practice problem 2.3. (Positional number systems.) Using digit symbols for the hexadecimal number system, write digit symbols for the *duodecimal* (base 12) system.
Answer: 0,1,2,3,4,5,6,7,8,9,A, and B.

Example 2.6 (Positional number system.) *Given* $n = 3$, *Equation 2.1 represents a number in hexadecimal, decimal, octal, and binary number systems by the following:*

$$N_{16} = \sum_{i=0}^{2} 16^i \times h_i = h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

$$N_{10} = \sum_{i=0}^{2} 10^i \times d_i = d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$$

$$N_8 = \sum_{i=0}^{2} 8^i \times o_i = o_2 \times 8^2 + o_1 \times 8^1 + o_0 \times 8^0$$

$$N_2 = \sum_{i=0}^{2} 2^i \times b_i = b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

The largest numbers that can be represented by three digits in these systems are $Max(N_{16}) = FFF_{16} = 4081_{10}$, $Max(N_{10}) = 999_{10}$, $Max(N_8) = 777_8 = 511_{10}$, *and* $Max(N_2) = 111_2 = 7_{10}$.

2.2.3 Fractional binary numbers

In the binary number positional representation, $B = \sum_{i=-n}^{m} 2^i \times b_i$, each binary digit or bit, b_i, is 0 or 1. The symbol "." becomes a *binary point*, separating bits on the left being weighted by positive powers of two, and those on the right being weighted by negative powers of two.

Example 2.7 (Binary numbers.) *The binary number* 101.11_2 *is written as follows:*

$$N = \sum_{i=-2}^{1} 2^i \times b_i$$
$$= \underbrace{1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0}_{Integer\ part} + \underbrace{1 \times 2^{-1} + 1 \times 2^{-2}}_{Fractional\ part} = 5\ {}^3/_4$$

Practice problem 2.4. (Binary numbers.) What does the num-ber 10.01_2 represent?
Answer is given in "Solutions to practice problems."

Shifting the binary point one position to the left has the effect of dividing the number by two.

Example 2.8 (Dividing by two.) *The binary number* 101.11_2 *represents the decimal number* $5\ {}^3/_4$. *Shifting the point one position to the left results in* 10.111_2, *that is,*

$$10.111_2 = 2 + 0 + {}^1/_2 + {}^1/_4 + {}^1/_8 = 2\ {}^7/_8$$

Practice problem 2.5. (Dividing by two.) Divide the number $2\ {}^1/_4 = 10.01_2$ by two.
Answer is given in "Solutions to practice problems."

Similarly, shifting the binary point one position to the right has the effect of multiplying the number by two. In computing, replacing arithmetic by shifts can occur when multiplying by constants.

Example 2.9 (Multiplying by two.) *The binary number* 101.11_2 *represents the decimal number* $5\ {}^3/_4$:

$$101.11_2 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$$
$$= 5\ {}^3/_4$$

Shifting the point one position to the right results in $1011.1_2 = 11\ {}^1/_2$, *that is,*

$$1011.1_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1})$$
$$= 8 + 0 + 2 + 1 + {}^1/_2 = 11\ {}^1/_2$$

Practice problem **2.6. (Multiplying by two.)** Multiply the number $1\ ^1/_8$ by two.

Answer is given in "Solutions to practice problems."

Example 2.10 (Fractional part.) *Samples of integer and fractional parts of binary numbers and their corresponding decimal equivalents are given in Table 2.2.*

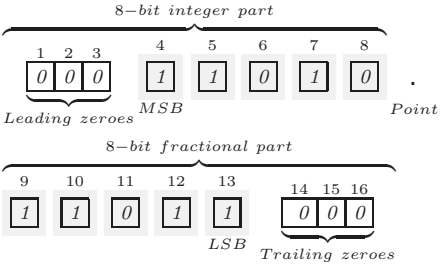
TABLE 2.2
Integer and fractional parts of binary numbers (Example 2.10).

Binary	Fraction (decimal)	Integer (decimal)
1.		1
10.		2
100.		4
1000.		8
10000.		16
100000.		32
0.1	$^1/_2$	0.5
0.01	$^1/_4$	0.25
0.001	$^1/_8$	0.125
0.0001	$^1/_{16}$	0.0625

2.2.4 Word size

In discrete systems, a *word* refers to a set of bits. *Word size* is defined as the number of bits in the binary numbers. Word size is typically a power of two and ranges from 8 bits, called a *byte*, to 32, 64, 128, or even 256 in some computers.

Example 2.11 (Byte.) *A 16-bit (two-byte) representation of a binary number with 8 bits for the integer part and 8 bits for the fractional part is shown below:*



Practice problem **2.7. (Byte.)** Represent the number 7.0625_{10} using an 8-bit (byte) number (four bits for the integer part and four bits for the fractional part).
Answer is given in "Solutions to practice problems."

2.3 Counting in a positional number system

Counting is the fundamental operation in digital systems. A positional number system is well-suited for counting; that is, the counting is well-automated and implemented in software and hardware.

Example 2.12 (Positional number systems.) *In Table 2.3, the counting process is illustrated in various number systems in order to show regularities, which are useful for automation.*

Practice problem **2.8. (Positional number systems.)** Using Table 2.3, write the number 16_{10} in the binary, ternary, octal, and hexadecimal number systems.
Answer: $16_{10} = 10000_2 = 121_3 = 20_8 = 10_{16}$.

2.4 Basic arithmetic operations in various number systems

The four basic arithmetic operations:

- Addition,
- Subtraction,
- Multiplication, and
- Division

can be performed in various positional number systems.

Example 2.13 (Basic operations.) *Addition and subtraction on the integer numbers in the binary, octal, decimal, and hexadecimal number systems are given in Table 2.4.*

TABLE 2.3
The first 16 integers in the binary, ternary, octal, decimal, and hexadecimal number systems (Example 2.12).

Binary	Ternary	Octal	Decimal	Hexadecimal
0	0	0	0	0
1	1	1	1	1
10	2	2	2	2
11	10	3	3	3
100	11	4	4	4
101	12	5	5	5
110	20	6	6	6
111	21	7	7	7
1000	22	10	8	8
1001	100	11	9	9
1010	101	12	10	A
1011	102	13	11	B
1100	110	14	12	C
1101	111	15	13	D
1110	112	16	14	E
1111	120	17	15	F

TABLE 2.4
Some arithmetic operations with unsigned numbers.

Techniques for computing with unsigned numbers

Radix	Technique	
Binary ($r = 2$)	$(3_{10})\quad 1\ 1\ 1\ 1_2$ $(2_{10})\ +\ 0\ 1\ 1\ 0_2$ $(5_{10})\quad 1\ 0\ 1\ 0\ 1_2$	$(5_{10})\quad 1\ 1\ 1\ 1_2$ $(6_{10})\ -\ 0\ 1\ 1\ 0_2$ $(11_{10})\quad 1\ 0\ 0\ 1_2$
Octal ($r = 8$)	$(15_{10})\quad 1\ 7_8$ $(6_{10})\ +\ 6_8$ $(21_{10})\quad 2\ 5_8$	$(15_{10})\quad 1\ 7_8$ $(6_{10})\ -\ 6_8$ $(9_{10})\quad 1\ 1_8$
Decimal ($r = 10$)	$1\ 5_{10}$ $+\ 6_{10}$ $2\ 1_{10}$	$1\ 5_{10}$ $-\ 6_{10}$ 9_{10}
Hexadecimal ($r = 16$)	$(15_{10})\quad \mathrm{F}_{16}$ $(6_{10})\ +\ 6_{16}$ $(21_{10})\quad 1\ 5_{16}$	$(15_{10})\quad \mathrm{F}_{16}$ $(6_{10})\ -\ 6_{16}$ $(9_{10})\quad 9_{16}$

2.5 Binary arithmetic

In the decimal system, the sign of a number is indicated by a special symbol, "+" or "-". In the binary system, the sign of a number is denoted by the

left-most bit. Positive numbers are represented using the positional number representation. The so-called *unsigned* representation (magnitude only) is used to denote positive numbers.

Negative numbers can be represented in different ways. The most commonly used are sign-and-magnitude and complemented, which can be 1's complement or 2's complement notation (Figure 2.2).

(*a*) (*b*)

FIGURE 2.2
Unsigned number format (a) and sign-and-magnitude format (b).

Sign-and-magnitude is a two-point binary notation (one bit for sign and the rest for magnitude):

$$\text{NUMBER} = \langle\text{SIGN}\rangle\langle\text{MAGNITUDE}\rangle$$

The *sign bit* is the left-most bit, called the *most significant bit*, and it is equal to 0 for positive numbers and 1 for negative numbers.

While performing addition, the magnitudes are added, and the resulting sum is given the sign of the operands. If the operands have opposite signs, it is necessary to subtract the smaller number from the larger one (logic networks that compare and subtract numbers are needed). The range of signed integers is $-(2^{n-1}-1) \leq x \leq 2^{n-1}-1$. In such a system, zero has two representations: positive zero $00\ldots0$ and negative zero $10\ldots0$.

Example 2.14 (Sign-and-magnitude.) *The 8-bit sign-and-magnitude representation of the number* 18_{10} *is*

$$18_{10} = \boxed{0}\ \overbrace{0010010}^{Magnitude}$$

The 8-bit sign-and-magnitude form of the number -18_{10} *is*

$-18_{10} = \boxed{1}\ \overbrace{0010010}^{Magnitude}$.

Practice problem 2.9. (Sign-and-magnitude.) Represent the

number -7_{10} in 8-bit sign-and-magnitude form.
Answer: $-7_{10} = 10000111_2$.

Techniques for manipulation of binary numbers in sign-and-magnitude format are given in Table 2.5.

TABLE 2.5
Sign-and-magnitude techniques.

Techniques for computing in sign-and-magnitude format

Example		Technique
(3) 0 0 1 1 (2) + 0 0 1 0 (5) 0 1 0 1	(7) 0 1 1 1 (7) + 0 1 1 1 (−6) 1 1 1 0 Incorrect	If sign bits are both 0, perform addition of two binary numbers. If sum of the magnitudes is greater then $2^{n-1}-1$, the result is incorrect.
(+3) 0 0 1 1 (−2) + 1 0 1 0 (1) 0 0 0 1	(−3) 1 0 1 1 (+2) + 0 0 1 0 (−1) 1 0 0 1	If signs are different, compare the magnitudes. Subtract the smallest magnitude from the greater, and assign to the result the sign of the greater magnitude.
(−3) 1 0 1 1 (−2) + 1 0 1 0 (−5) 1 1 0 1	(−7) 1 1 1 1 (−1) + 1 0 0 1 (0) 1 0 0 0 0 Incorrect	If sign bits are both 1, perform addition of both magnitudes; the sign bit of the result is 1. If sum of the magnitudes is greater then $2^{n-1}-1$, the result is incorrect.

2.6 Radix-complement representations

Consider the number D that consists of n digits $d_i,\ i = 1, 2, \ldots, n$ in the radix r number system. There are two types of *radix-complements* for representation of the number:

The r radix-complement

$$\overline{D} = r^n - D \tag{2.2}$$

The $r-1$ radix-complement

$$\overline{D} = (r^n - 1) - D \tag{2.3}$$

Example 2.15 (Radix-complement.) *Given the binary number* $D = 101_2$ *(*$n = 3$*), the 2's and 1's complements are* $\overline{D} = 2^3 - 101 = 1000 - 101 = 011_2$ *and* $\overline{D} = (2^3 - 1) - 101 = 111 - 101 = 010_2$*. Given the the decimal number* $D = 125_{10}$ *(*$n = 3$*), the 10's and 9's complements are* $\overline{D} = 10^3 - 125 = 875_{10}$ *and* $\overline{D} = 10^3 - 1 - 125 = 874_{10}$*.*

While the sign-and-magnitude system makes a number negative by changing its sign, a *complement number system* makes a number negative by taking its complement. Two numbers in a complement number system can be added or subtracted directly without the sign and magnitude checks required by the sign-and-magnitude system.

The *radix-complement* representation for decimal ($r = 10$) and binary ($r = 2$) number systems is shown in Figure 2.3.

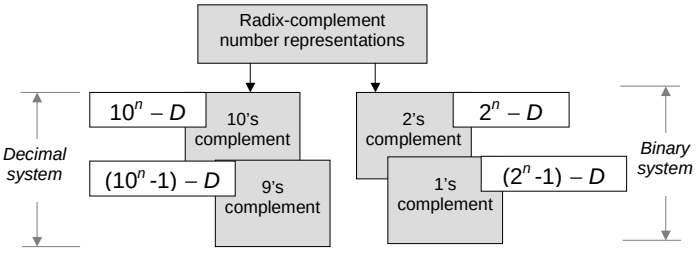


FIGURE 2.3
Radix-complement representations of decimal and binary numbers.

2.6.1 10's and 9's complement systems

In 10's and 9's complement systems, positive numbers are represented using the same binary code as for unsigned numbers. Negative numbers are represented in a complement form.

10's complement system

In the decimal number system, the radix-complement is called *10's complement.*

The rule for forming the 10's complement

Given: `A decimal number` $D = d_{n-1}d_n \ldots d_0$.
Step 1: `Subtract each` d_i `from 9:` $(9-d_{n-1}), (9-d_{n-2}), \ldots, (9-d_0)$.
Step 2: `Add 1 to the resulting number.`

Example 2.16 (10's complement.) *Given the decimal number 125, its 10's complement is calculated as follows:*

$$\begin{aligned}\overline{125} &= (9-d_2)(9-d_1)(9-d_0)+1\\ &= (9-1)(9-2)(9-5)+1 = 874+1 = 875\end{aligned}$$

which is the same result as the that obtained by the calculation in Example 2.15.

Practice problem 2.10. (10's complement.) Find the 10's complement of the number 17_{10}.
Answer: $\overline{17}_{10} = (9-1)(9-7)+1 = 82+1 = 83_{10}$.

9's complement

The 9's complement of a decimal number is formed as follows:

The rule for forming the 9's complement

Given: `A decimal number` $D = d_{n-1}d_n \ldots d_0$.
Step 1: `Subtract each` d_i `from 9:` $(9-d_{n-1}), (9-d_{n-2}), \ldots, (9-d_0)$.

Example 2.17 (9's complement.) *Given the decimal number 46, 125 and 5329, their 9's complements are calculated as follows:*

$$\begin{aligned}\overline{46} &= (9-d_1)(9-d_0) = (9-4)(9-6) = 53\\ \overline{125} &= (9-d_2)(9-d_1)(9-d_0) = (9-1)(9-2)(9-5) = 1874\\ \overline{5329} &= (9-d_3)(9-d_2)(9-d_1)(9-d_0)\\ &= (9-5)(9-3)(9-2)(9-9) = 4670\end{aligned}$$

Practice problem 2.11. (9's complement.) Find the 9's complement of the number 17_{10}.
Answer: $\overline{17}_{10} = (9-1)(9-7) = 82_{10}$.

2.6.2 1's complement system

In 1's complement system, positive numbers are represented in the same way as unsigned numbers. Let a negative number $-P$ be given, where P is the magnitude of the number. An n-bit negative number K is obtained by subtracting the positive number P (magnitude) from $2^n - 1$:

1's complement system

$$K = (2^n - 1) - P \tag{2.4}$$

An advantage of 1's complement representation is that a negative number is generated by complementing all bits of the corresponding positive number (magnitude). The addition of 1's complement numbers may require a correction, and the time needed to add two 1's complement numbers may be twice as long as the time needed to add two unsigned numbers.

Rule for forming the 1's complement

Given: A binary number $B = b_{n-1}b_n \dots b_0$.

Step 1: Complement each digit of the binary number:

$$(1 - b_{n-1}), (1 - b_{n-2}), \dots, (1 - b_0)$$

Step 2: Add 1 to the result.

Example 2.18 (1's complement.) *An n-bit binary representation of numbers between +7 and -7 with 1's complement representation of negative numbers is given below.*

Binary number	Decimal number	Binary number	Decimal number
0111	*+7*	*1111*	*-0*
0110	*+6*	*1110*	*-1*
0101	*+5*	*1101*	*-2*
0100	*+4*	*1100*	*-3*
0011	*+3*	*1011*	*-4*
0010	*+2*	*1010*	*-5*
0001	*+1*	*1001*	*-6*
0000	*+0*	*1000*	*-7*

For example, the negative number -7_{10} is represented as a 4-bit number using the equation $K = (2^4 - 1) - 7 = 8_{10} = \boxed{1}\,000_2$. Alternatively, given $-7_{10} = 0111_2$, its complement is the desired number $-7_{10} = \overline{0111}_2 = \boxed{1}\,000_2$.

Practice problem 2.12. (1's complement.) Represent the numbers -5 and 5 using 4-bit binary code.

Answer: $-5_{10} = (2^4 - 1) - 5 = 10_{10} = 1010_2$, or $-5_{10} = \overline{0101}_2 = 1010_2$. $+5_{10} = 0101_2$.

2.6.3 2's complement

A code for representing an n-bit negative number K is obtained by subtracting its equivalent positive number P from 2^n, i.e., $K = 2^n - P$. An advantage of 2's complement representation is that when the numbers are added, the

result is always correct. If there is a carry-out from the sign-bit position, it is simply ignored.

The rule for forming the 2's complement

Given: The binary code for the magnitude of the negative number $B = b_{n-1}b_n \ldots b_0$
Step 1: Complement each digit of the code

$$(1 - b_{n-1}), (1 - b_{n-2}), \ldots, (1 - b_0)$$

Example 2.19 (2's complement.) *Binary representation of numbers between +7 and −7 with 2's complement representation of negative numbers is given below. Note that the binary codes of the positive numbers are represented exactly like the unsigned numbers.*

Binary number	Decimal number
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0

Binary number	Decimal number
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7

For example, the negative number -7_{10} is represented in 2's complement form as

$$-7_{10} = 2^4 - 7 = 9_{10} = 1001_2$$

Alternatively,

$$-7_{10} = \overline{1111}_2 + 1 = 1000_2 + 1 = 1001_2$$

Practice problem **2.13. (2's complement.)** Represent the numbers -6_{10} and 6_{10} in 2's complement form.
Answer: $-6_{10} = \overline{0110}_2 + 1 = 1010_2$. $6_{10} = 0110_2$.

Example 2.20 (2's complement.) *Techniques for the addition of 2's complement binary numbers are demonstrated in Table 2.6.*

TABLE 2.6
Techniques for representation and addition of binary numbers: unsigned, sign-and-magnitude, 1's and 2's complement.

Techniques for computing binary numbers

Radix	Technique	
Unsigned	(3) 0 0 1 1 (2) + 0 0 1 0 (5) 0 1 0 1	(5) 0 1 0 1 (6) + 0 1 1 0 (11) 1 0 1 1
Sign-and-magnitude	(+7) 0 1 1 1 (−2) + 1 0 1 0 (+5) 0 1 0 1	(−5) 1 1 0 1 (−2) + 1 0 1 0 (−7) 1 1 1 1
1's complement $K = (2^n - 1) - P$ $-5 = \underbrace{1111}_{2^4-1} - \underbrace{0101}_{P=5} = 1010$ $-9 = \underbrace{11111}_{2^5-1} - \underbrace{01001}_{P=9} = 10110$	(+5) 0 1 0 1 +(−2) + 1 1 0 1 (+3) $\boxed{1}$ 0 0 1 0 + ⟶ 1 0 0 1 1	(−5) 1 0 1 0 +(−2) + 1 1 0 1 (−7) $\boxed{1}$ 0 1 1 1 + ⟶ 1 1 0 0 0
2's complement $K = 2^n - P$ $-5 = \underbrace{1000}_{2^4} - \underbrace{0101}_{P=5} = 1011$ $-9 = \underbrace{10000}_{2^5} - \underbrace{01001}_{P=9} = 10111$	(+5) 0 1 0 1 +(−2) + 1 1 1 0 (+3) $\boxed{1}$ 0 0 1 1 ↑ *Ignore*	(−5) 1 0 1 1 +(−2) + 1 1 1 0 (−7) $\boxed{1}$ 1 0 0 1 ↑ *Ignore*

2.7 Conversion of numbers in various radices

Consider a radix r number that includes a radix point. First, the number must be separated into an integer part and a fractional part, since the parts must be converted differently. There are various algorithms for the conversion of numbers between systems. In Figure 2.4, possible conversions between binary, octal, hexadecimal, and decimal systems are shown.

Conversion of numbers from decimal to other radices

Given a decimal number, the conversion of this number to a radix r number is as follows:

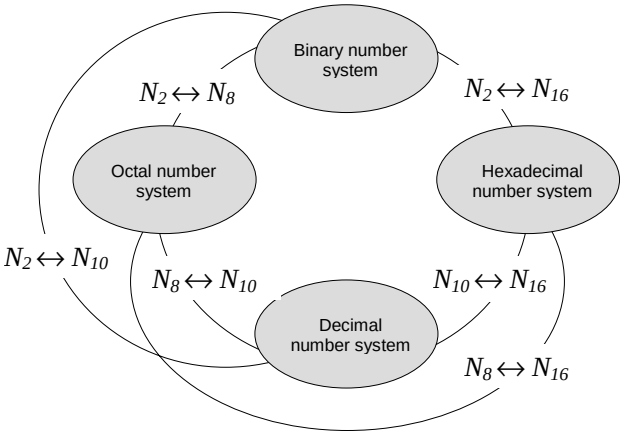


FIGURE 2.4
Conversion of numbers in various radices $N_{r_i} \leftrightarrow N_{r_j}$.

Conversion of numbers from decimal to other radices

- The conversion of a decimal integer to a radix r number is achieved by dividing the number and all successive quotients by r and accumulating the remainders.
- The conversion of a fraction to a radix r fraction is accomplished by multiplying by radix r to give the integer and fraction; the new fraction is multiplied by r to give a new integer and a new fraction. This process is continued until the fractional part equals 0 or until there are enough digits to achieve sufficient accuracy.

Conversion of decimal integers to binary integers

For the conversion of decimal integers to binary integers, the following steps are needed:

Conversion of decimal integers to binary integers

Given: A decimal integer
Step 1: Divide the decimal number by two to give a quotient and a remainder
Step 2: Divide the quotient by two to give a new quotient and a remainder
Step 3: Repeat division until the fractional part is 0 or until required number of digits is achieved
Result: The remainders are written bottom-up (from recent to the first obtained) of the desired binary number

Example 2.21 (Conversion of a decimal into a binary number.) *Convert the decimal integer number* 50_{10} *into a binary number. Figure 2.5a illustrates the conversion:*

$$\begin{aligned} 50 : 2 &= 25 + \boxed{0} \\ 25 : 2 &= 12 + \boxed{1} \\ 12 : 2 &= 6 + \boxed{0} \\ 6 : 2 &= 3 + \boxed{0} \\ 3 : 2 &= 1 + \boxed{1} \\ 1 : 2 &= 0 + \boxed{1} \end{aligned}$$

The sequence of remainders is 0,1,0,0,1,1. The result of conversion is obtained by reading the remainders in reverse order (from the bottom up in Figure 2.5),

$$50_{10} = 110010_2$$

Practice problem 2.14. (Conversion of a decimal into a binary number.) Convert the decimal number 2159 into a binary number.
Answer is given in "Solutions to practice problems."

Conversion of a decimal fraction to a binary fraction

For the conversion of a decimal fraction to a binary fraction, the following steps are needed:

Conversion of a decimal fraction to a binary fraction

Given: A decimal fraction
Step 1: Multiply the fraction by 2
Step 2: Write down the obtained integer part. Multiply the fractional part of the obtained result by 2
Step 3: Repeat until 0 is obtained as a fractional part, or until the required accuracy is achieved

Example 2.22 (Conversion of a decimal into a binary number.) *Convert the decimal fraction* 0.4_{10} *into a binary fraction. Represent the result using 8 bits. Figure 2.5b illustrates the conversion. The result of conversion is obtained by reading the remainders from the bottom-up, that is,* $0.4_{10} = 01100110_2$.

Practice problem 2.15. (Conversion of a decimal into a binary number.) Convert $+12.0625_{10}$ into a binary number.
Answer is given in "Solutions to practice problems."

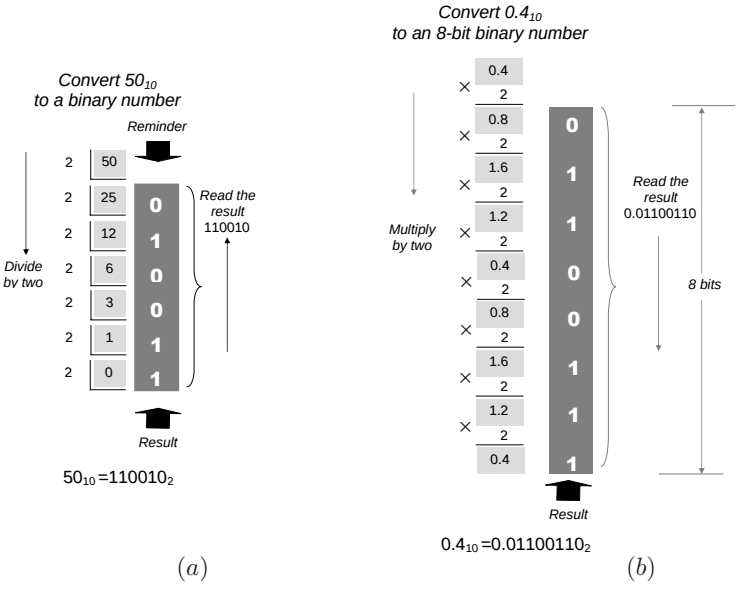


FIGURE 2.5
Conversion of a integer decimal number into a binary one (a) and conversion of a decimal fraction into a binary one (b) (Examples 2.21 and 2.22).

Conversion of decimal integers to octal integers

The following steps are needed to convert a decimal integer to an octal integer:

Conversion of decimal integer to octal integer

Given: A decimal integer.
Step 1: Divide the decimal number by 8 to give a quotient and a remainder.
Step 2: Divide the quotient by 8 to give new quotient and a remainder.
Step 3: Repeat division until the fractional part is 0 or until the required number of digits is achieved.
Result: The remainders written bottom-up (from recent to the first obtained) of the desired octal number.

Practice problem **2.16. (Conversion of a decimal into an octal number.)** Convert the decimal number 2159 into an octal number using the intermediate binary number system (use the result of Practice problem 2.15).

Answer is given in "Solutions to practice problems."

Conversion of decimal integers to hexadecimal integers

The following steps are needed to convert decimal integer into hexadecimal integer:

Conversion of decimal integer to hexdecimal integer

Given: A decimal integer.
Step 1: Divide the decimal number by 16 to give a quotient and a remainder.
Step 2: Divide the quotient by 16 to give a new quotient and a remainder.
Step 3: Repeat division until the fractional part is 0 or until the required number of digits is achieved.
Result: The remainders written bottom-up (from most recent to first obtained) of the desired hexadecimal number.

Practice problem 2.17. (Conversion of a decimal into a hexadecimal number.) Convert the decimal number 2159 into a hexadecimal number using the intermediate binary number system (use the result of Practice problem 2.15).
Answer is given in "Solutions to practice problems."

A binary number can be converted to a decimal one via an intermediate system (octal or hexadecimal) (Figure 2.6).

Practice problem 2.18. (Conversion of decimal into octal and hexadecimal numbers.)

(*a*) Convert the decimal number 746_{10} into an octal number.
(*b*) Convert the decimal number 746_{10} into a hexadecimal number.

Answer is given in "Solutions to practice problems."

Techniques for conversion between binary, octal, and hexadecimal number systems are given in Table 2.7.

2.8 Overflow

The operations under consideration here are executed within the binary system as well other systems of restricted word length or number of digits.

TABLE 2.7
Techniques for conversion between binary, octal, and hexadecimal numbers systems.

Techniques for conversion between numbers

Example	Technique
Binary to octal	
$1101001_2 = \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{001}_{1} = 151_8$	Separate the bits into groups of three, starting from the right.
$10101.0101_2 = \underbrace{010}_{2}\ \underbrace{101}_{5}.\underbrace{010}_{2}\ \underbrace{100}_{4} = 25.24_8$	For the fractional part, start from the left.
Octal to binary	
$457.24_8 = \underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{111}_{7}.\underbrace{010}_{2}\ \underbrace{100}_{4}$	Replace each octal digit with a 3-bit string. For the fractional part, start from the left.
Binary to hexadecimal	
$101101101_2 = \underbrace{0001}_{1}\ \underbrace{0110}_{6}\ \underbrace{1101}_{\mathsf{D}} = 16\mathsf{D}_{16}$	Separate the bits into groups of four, starting from the right.
$10111.11_2 = \underbrace{0001}_{1}\ \underbrace{0111}_{7}.\underbrace{0011}_{3} = 17.3_{16}$	For the fractional part, start from the left.
Hexadecimal to binary	
$5\mathsf{E}4_{16} = \underbrace{0101}_{5}\ \underbrace{1110}_{\mathsf{E}}\ \underbrace{0100}_{4}$	Replace each hexadecimal digit with the corresponding 4-bit string.
$\mathsf{C}2.\mathsf{A}1_{16} = \underbrace{1100}_{\mathsf{C}}\ \underbrace{0010}_{2}.\underbrace{1010}_{\mathsf{A}}\ \underbrace{0001}_{1}$	For the fractional part, start replacing from the left.
Octal to hexadecimal	
$1352_8 = \underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{101}_{5}\ \underbrace{110}_{6}$ $= \underbrace{0010}_{2}\ \underbrace{1110}_{\mathsf{E}}\ \underbrace{1010}_{\mathsf{A}} = 2\mathsf{EA}_{16}$	Separate the bits into groups of three, starting from the right, and then regroup into groups of four.
Hexadecimal to octal	
$2\mathsf{EA}_{16} = \underbrace{0010}_{2}\ \underbrace{1110}_{\mathsf{E}}\ \underbrace{1010}_{\mathsf{A}}$ $= \underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{101}_{5}\ \underbrace{010}_{2} = 1352_8$	Replace each hexadecimal digit with a 4-bit binary string. Separate the bits into groups of four, starting from the right, and then regroup into groups of three.

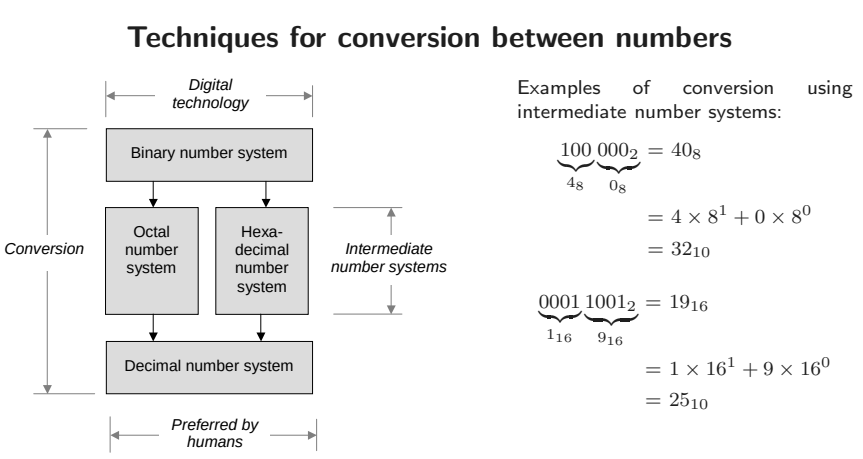


FIGURE 2.6
Binary numbers are converted into decimal ones using octal and hexadecimal systems.

Below we will focus on the binary number system. Binary word size determines the range of the allowed number values of both operands and results.

Example 2.23 (The range.) *Given a 4-bit word, the binary numbers that can be represented are varied:* (*a*) *from* $-7_{10} = 1111_2$ *to* $+7_{10} = 0111_2$ *for unsigned numbers;* (*b*) *from* $-7_{10} = 1111_2$ *to* $+7_{10} = 0111_2$ *in the sign-magnitude system;* (*c*) *from* $-8_{10} = 1000_2$ *to* $+7_{10} = 0111_2$ *for the 2's complement system.*

In particular, if an addition operation in the 2's complement system produces a result that does not fit the range -2^{n-1} to $2^{n-1} - 1$, then we say that *arithmetic overflow* has occurred. To ensure the correct operation, it is important to be able to detect the occurrence of overflow.

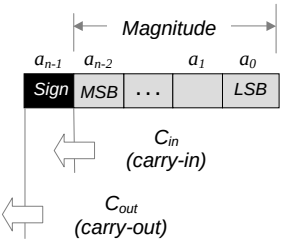
The following rules are used for detecting overflow:

The rules for detecting overflow

Rule 1: An addition overflows if:

(*a*) The signs of the addends are the same and
(*b*) The sign of the sum is different from the addends' sign

Rule 2: An addition overflows (Figure 2.7) if the carry-in, C_{in}, and the carry-out of the most significant bit in 2's complement representation are different, $C_{in} \neq C_{out}$.



An addition overflows if the carry bit C_{in} into and carry bit C_{out} out of the sign position are different:

$$C_{in} \neq C_{out}$$

There is no overflow if

$$C_{in} = C_{out}$$

FIGURE 2.7
Overflow detection using carries.

Example 2.24 (Overflow.) *Figure 2.8 illustrates four cases of overflow detection using carry bits C_{in} and C_{out} for the two 4-bit binary numbers. Note that in all cases the sign bit is included in computing the sum.*

Practice problem 2.19. (Detection of overflow.) Find which of the following operations result in overflow using 6-bit 2's complement codes of the given decimal numbers:

(a) $14_{10} + 22_{10}$
(b) $(-18_{10}) + (-11_{10})$
(c) $(-18_{10}) + (-27_{10})$
(d) $25_{10} + (-17_{10})$

Answer is given in "Solutions to practice problems."

2.9 Residue arithmetic

Residue arithmetic offers the alternative number format. An arithmetic operation performed on n-bit numbers may produce a result that is too long to be represented completely by n bits; that is, overflow occurs. In residue arithmetic, the results of all arithmetic operations are confined to some fixed set of m values such as $0, 1, \ldots, m-1$. Residue arithmetic ensures a finite word size in computing and is widely used in computing devices.

2.9.1 The basics of residue arithmetic

A *residue* is defined as the remainder after a division. Given the representation of an integer N,

$$N = Im + r$$

Techniques for overflow detection

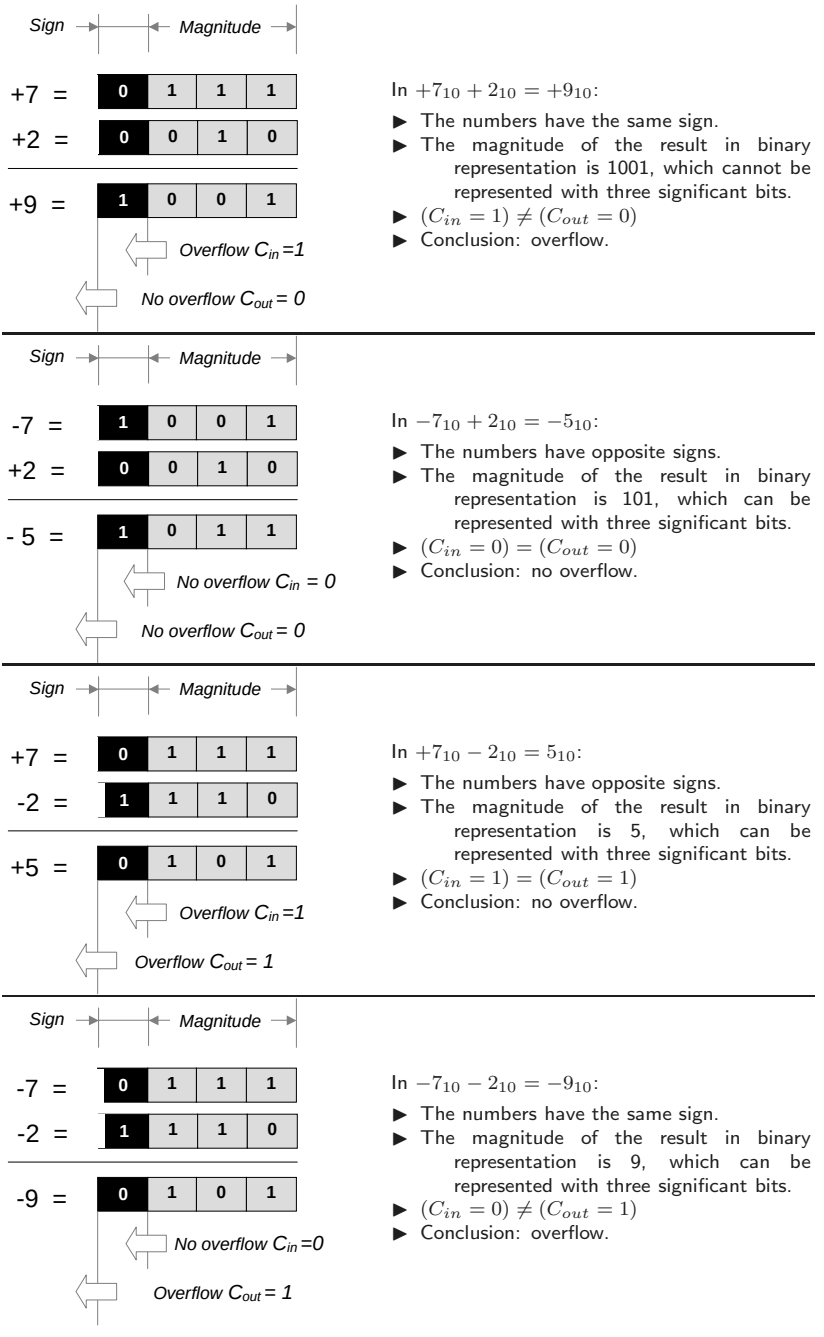


FIGURE 2.8
Overflow detection using carries C_{in} and C_{out}.

where m is a check base and I is an integer, so that $0 \leq r \leq m$, and N is said to be equivalent to modulo m to r:

$$r \equiv N \ (mod\ m)$$

The numbers a and b are said to be *equivalent to modulo m* if the remainder obtained when a is divided by m is the same as the remainder that is obtained when b is divided by m:

$$a \equiv b \ \ (mod\ m)$$

(read as "a is congruent b modulo m").

Example 2.25 (Residue arithmetic.) *If* $a = 10$ *and* $b = 18$, *then* $10 \equiv 18 \ \ (mod\ 8)$ *since*

$$10 = 1 \times 8 + \overbrace{\boxed{2}}^{Remainder} \quad and \quad 18 = 2 \times 8 - \overbrace{\boxed{2}}^{Remainder}$$

Example 2.26 (Residue arithmetic.)

(*a*) $3 \equiv 17 \ (mod\ 7)$ *because* $3 - 17 = -14$ *is divisible by 7.*

(*b*) $-2 \equiv 13 \ (mod\ 3)$ *because* $-2 - 13 = -15$ *is divisible by 3.*

(*c*) $60 \equiv 10 \ (mod\ 25)$ *because* $60 - 10 = -50$ *is divisible by 25.*

(*d*) $-4 \equiv -49 \ (mod\ 9)$ *because* $-4 - (-49) = 45$ *is divisible by 9.*

It follows from these examples that an integer a is congruent to 0, $a \equiv 0 \ (mod\ m)$, if and only if it is divisible by m. Additional properties of congruence are:

- $a \equiv a \ (mod\ m)$ (reflexive property),
- If $a \equiv b \ (mod\ m)$, then $b \equiv a \ (mod\ m)$ (symmetry property), and
- If $a \equiv b \ (mod\ m)$ and $b \equiv c \ (mod\ m)$, then $a \equiv c \ (mod\ m)$ (transitive property).

2.9.2 Addition in residue arithmetic

If $a \equiv b \ (\text{mod}\ m)$ and $c \equiv d \ (\text{mod}\ m)$, then

Addition in residue arithmetic

$$a + c \equiv b + d \ (mod\ m)$$
$$a - c \equiv b - d \ (mod\ m)$$

Example 2.27 (Addition.) *Find the sum* $1017 + 2876$ *(mod* 7*). There are two ways to calculate the sum. First approach:* $1017 + 2876 = 3893 \equiv 1$ *(mod* 7*). Second approach:* $1017 \equiv 2$ *(mod* 7*),* $2876 \equiv 6$ *(mod* 7*), and* $1017 + 2876 \equiv 2 + 6 = 8 \equiv 1$ *(mod* 7*). The second approach is preferable because it keeps the numbers involved small.*

Practice problem 2.20. (Addition.) Find the sum of two congruences: $13 \equiv 4 \pmod 9$ and $16 \equiv -2 \pmod 9$.
Answer: $13 + 16 \equiv 4 - 2 = 29 \equiv 2 \pmod 9$.

2.9.3 Multiplication in residue arithmetic

If $a \equiv b \pmod m$ and $c \equiv d \pmod m$, then

Multiplication in residue arithmetic

$$ac \equiv bd \ (mod\ m)$$

Example 2.28 (Multiplication.) *Find the product* 1017×2876 *(mod* 7*). Solution:* $1017 \equiv 2$ *(mod* 7*),* $2876 \equiv 6$ *(mod* 7*), and* $1017 \times 2876 \equiv 2 \times 6 = 12 \equiv 5$ *(mod* 7*)*4.

Practice problem 2.21. (Multiplication.) Find the multiplication of two congruences: $13 \equiv 4 \pmod 9$ and $16 \equiv -2 \pmod 9$.
Answer: $13 \times 16 \equiv 4 \times (-2) = 208 \equiv -8 \pmod 9$.

Example 2.29 (Addition and multiplication modulo *m*=5.) *Addition and multiplication modulo 5 of two residues are given in the following tables:*

ADDITION
$a + c \equiv b + d \ (mod\ 5)$

+	*0*	*1*	*2*	*3*	*4*
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

SUBTRACTION
$a - c \equiv b - d \ (mod\ 5)$

−	*0*	*1*	*2*	*3*	*4*
0	0	4	3	2	1
1	1	0	4	3	2
2	2	1	0	4	3
3	3	2	1	0	4
4	4	3	2	1	0

MULTIPLICATION
$ac \equiv bd \ (mod\ m)$

×	*0*	*1*	*2*	*3*	*4*
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

For example, the residue 3 subtracted from the residue 1 modulo 5 is calculated as $1-3$ *equals* -2*, and* $-2 \equiv 3$ *(mod* 5*), so the difference is the residue 3. The product of the two residues 3 and 4 modulo5 is the residue 2, because* $3 \times 4 \equiv 2$ *(mod* 5*).*

2.9.4 Computing powers in residue arithmetic

If $a \equiv b \pmod{m}$, then

Computing powers in residue arithmetic

$a^n \equiv b^n \ (mod\ m)$ for every positive integer n

Example 2.30 (Computing powers.) *Find the products* $1017^2 \ (mod\ 7)$, $1017^3 \ (mod\ 7)$, $1017^4 \ (mod\ 7)$, $1017^5 \ (mod\ 7)$, $1017^{12} \ (mod\ 7)$. *Since* $1017 \equiv 2 \ (mod\ 7)$,

$$\begin{aligned} 1017^2 &\equiv 2^2 \ (mod\ 7) \\ 1017^3 &= 1017^2 \times 1017 \equiv 4 \times 2 = 8 \equiv 1 \ (mod\ 7) \\ 1017^4 &= 1017^3 \times 1017 \equiv 1 \times 2 = 2 \ (mod\ 7) \\ 1017^5 &= 1017^4 \times 1017 \equiv 2 \times 2 = 4 \ (mod\ 7) \\ 1017^{12} &= ((1017^4))^3 \equiv 2^3 = 8 \equiv 1 \ (mod\ 7) \end{aligned}$$

Practice problem **2.22. (Computing powers.)** Compute 3^{13}.

Answer is given in "Solutions to practice problems."

The following property of a residue arithmetic is useful, for example, in electronic cash systems. The quadratic residues of a set modulo n are the elements that have a square root in the set.

Example 2.31 (Computing powers.) *For* $n = 11$, *the number 4 is a quadratic residue because* $2^2 \ (mod\ 11) = 4$. *The number 5 is also a quadratic residue because* $7^2 \ (mod\ 11) = 5$. *If* n *is prime, then there are* $(n-1)/2$ *quadratic residues. In the case of* $n = 11$, *the residues are:*

$$\begin{aligned} 1^2 &= 1 \ (mod\ 11) & \qquad 6^2 &= 3 \ (mod\ 11) \\ 2^2 &= 4 \ (mod\ 11) & \qquad 7^2 &= 5 \ (mod\ 11) \\ 3^2 &= 9 \ (mod\ 11) & \qquad 8^2 &= 9 \ (mod\ 11) \\ 4^2 &= 5 \ (mod\ 11) & \qquad 9^2 &= 4 \ (mod\ 11) \\ 5^2 &= 3 \ (mod\ 11) & \qquad 10^2 &= 1 \ (mod\ 11) \end{aligned}$$

Each quadratic residue has two square roots if n is prime. One of them is smaller than $n/2$. The other is larger.

2.9.5 Solving modular equations

To solve a congruence or a system of congruences involving one or more unknowns means to find all possible values of the unknowns which make the

congruence true.

Example 2.32 (Modular equation.) *Solution of the congruence* $3x \equiv 1 \ (mod\ 5)$ *can be found by trying all possible values of* x *modulo 5:*

$$\begin{aligned}
&\text{IF } x = 0, \text{ THEN } && 3x = 0 \equiv 0 \ (mod\ 5)\\
&\text{IF } x = 1, \text{ THEN } && 3x = 3 \equiv 3 \ (mod\ 5)\\
&\text{IF } x = 2, \text{ THEN } && 3x = 6 \equiv 1 \ (mod\ 5)\\
&\text{IF } x = 3, \text{ THEN } && 3x = 9 \equiv 4 \ (mod\ 5)\\
&\text{IF } x = 4, \text{ THEN } && 3x = 12 \equiv 2 \ (mod\ 5)
\end{aligned}$$

Since the modulus is 5, the integer x *is in the range* $0 \leq x \leq 5$. *Thus, the only solution to the congruence is* $x = 2$.

Practice problem **2.23. (Modular equation.)** Solve the following congruences if possible: (a) $3x \equiv 1 \ (mod\ 6)$ and (b) $3x \equiv 3 \ (mod\ 6)$. ***Answer*** is given in "Solutions to practice problems."

2.9.6 Complete residue systems

In residue arithmetic, an integer is represented as a set of residues with respect to a set of relatively prime integers called *moduli*. Residue arithmetic is defined in terms of a set of relatively prime moduli $\{r_1, r_2, \ldots, r_s\}$, where the greatest common divisor is equal to 1 for each pair of moduli. The set of integers $\{r_1, r_2, \ldots, r_s\}$ is called a *complete residue system modulo* m if $r_i \neq r_j \pmod m$ whenever $i \neq j$, and for each integer n exists a corresponding r_i such that $n = r_i \pmod m$.

Example 2.33 (Complete residue system.) *The sets* $\{1, 2, 3\}$, $\{-1, 0, 1\}$, *and* $\{1, 7, 9\}$ *are all complete residue systems modulo 3. The set* $\{0, 1, 2, 3, 4, 5\}$ *is a complete residue system modulo 6. It should be noted that this set can be reduced to* $\{1, 5\}$.

While in ordinary arithmetic there is an infinite number of integers $0, 1, 2, \ldots$, in modular arithmetic there is essentially only a finite number of integers.