Miloš ERCEGOVAC | Tomás LANG

Digital Arithmetic

Digital Arithmetic

This Page Intentionally Left Blank

Digital Arithmetic

Miloš D. Ercegovac

COMPUTER SCIENCE DEPARTMENT

UNIVERSITY OF CALIFORNIA, LOS ANGELES

Tomás Lang

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF CALIFORNIA, IRVINE

MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER SCIENCE

SAN FRANCISCO SAN DIEGO NEW YORK BOSTON

LONDON SYDNEY TOKYO

Senior Editor	Denise E. M. Penrose
Publishing Services Manager	Simon Crump
Project Manager	Justin Palmeiro
Project Management	Graphic World Publishing Services
Editorial Coordinator	Alyson Day
Cover Design Manager	Cate Rickard Barr
Cover Design Coordinator	Elisabeth Beller
Cover Art	© Jasper Johns/Licensed by VAGA, New York, NY
Cover Photos	© CNAC/MNAM/Dist. Réunion des Musées Nationaux/Art Resource, NY Johns, Jasper. Zero of Figures 0–9. 1960–1971. Lithograph, re-worked with paint, fiber collage and Japanese papers. Dimensions: $12\,^{3}/_{4}'' \times 10\,^{1}/_{2}''$. Inv.: Am 1983–314(1). Photo: Philippe Migeat. Musee National d'Art Moderne, Centre Georges Pompidou, Paris, France. Johns, Jasper. One of Figures 0–9. 1960–1971. Lithograph, re-worked with paint, fiber collage and Japanese papers. Dimensions: $12\,^{7}/_{8}'' \times 10\,^{1}/_{2}''$. Inv.: Am 1983–314(2). Photo: Philippe Migeat. Musee National d'Art Moderne, Centre Georges Pompidou, Paris, France.
Text Design	Frances Baca Design
Composition	International Typesetting and Composition
Printer	Maple-Vail, York

Morgan Kaufmann Publishers
An imprint of Elsevier Science
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205, USA
www.mkp.com

Printed in the United States of America

07 06 05 04 03 5 4 3 2 1

Library of Congress Control Number: 2002114337

ISBN: 1-55860-798-6

This book is printed on acid-free paper.

About the Authors

Miloš D. Ercegovac is a professor and chair in the UCLA Computer Science Department. He earned an MS (1972) and PhD (1975) in computer science from the University of Illinois, Urbana-Champaign, and a BS in electrical engineering (1965) from the University of Belgrade, Yugoslavia. Dr. Ercegovac specializes in research and teaching in digital arithmetic, digital design, and computer system architecture. His research contributions have been extensively published in journals and conference proceedings. He is a coauthor of two textbooks on digital design and a monograph in the area of digital arithmetic. Dr. Ercegovac has been involved in organizing the IEEE Symposia on Computer Arithmetic since 1978. He served as an editor of the *IEEE Transactions on Computers* and as a subject area editor for the *Journal of Parallel and Distributed Computing*. Dr. Ercegovac is a fellow of the IEEE Computer Society and a member of the ACM.

Tomás Lang is a professor in the Department of Electrical and Computer Engineering at the University of California, Irvine. Previously he was a professor in the Computer Architecture Department of the Polytechnic University of Catalonia, Spain, and a faculty member of the Computer Science Department at the University of California, Los Angeles. He received an electrical engineering degree from the Universidad de Chile in 1965, an MS from the University of California Berkeley in 1966, and the PhD from Stanford University in 1974. Dr. Lang's primary research and teaching interests are in digital design and computer architecture, with current emphasis on high-speed and low-power numerical processors and multiprocessors. He is a coauthor of two textbooks on digital systems, two research monographs, one IEEE Tutorial, and the author or coauthor of numerous research contributions to scholarly publications and technical conferences.

This Page Intentionally Left Blank

Contents

CHAPTER 11

CORDIC Algorithm and Implementations 609

This Page Intentionally Left Blank

Preface

Objectives and Importance

Our main objective in preparing this book is to provide a comprehensive discussion of the main ideas and concepts in digital arithmetic, reflecting both the theory and design aspects, and to help students and practicing engineers develop a good understanding of the "arithmetic style" of algorithms and designs. The research in digital arithmetic continues to be active, and new areas of applications are being introduced, making such a book useful in understanding the state of the art in digital arithmetic in order to develop sound solutions and avoid mistakes and repetitions. Lastly, a thorough exposition of digital arithmetic is likely to stimulate interest in the field.

Digital arithmetic has continued to play an important role in the design of digital processors and application-specific (embedded) systems found in signal processing, graphics, and communications. In spite of a mature body of knowledge, it is not unusual that in each new generation of processors or digital systems new arithmetic design problems need to be solved. A good solution benefits greatly from a comprehensive exposition to digital arithmetic as provided in this book.

Audience

The material covered in this book is intended for graduate students in computer engineering/electrical engineering and computer science who are interested in the design of digital arithmetic for general-purpose processors, application-specific and embedded digital systems, and signal processing systems. It will also be useful to practicing digital design engineers involved in logic and circuit design of arithmetic and floating-point units and in their implementation in VLSI technologies. The background expected consists primarily of college-level mathematics, digital

systems and logic design, and, for those interested in applying the material to implementation, a knowledge of VLSI design tools.

Features and Approach

The main feature of our approach has been in providing a unified treatment of digital arithmetic, tying the underlying theory and design practice in a technology-independent manner. We consistently use an algorithmic approach in defining arithmetic operations, illustrate with examples of designs at the logic level, and discuss cost/performance characteristics. To enhance learning, we developed a large set of exercises with solutions and extensive reading lists. These are included in each chapter, and general references (books and compilation of articles) are given in Chapter 1. For instructors we have developed a complete set of lecture viewgraphs.

Ways of Use

The main use of this book is as a text for a graduate course. As such, it can be covered completely in a semester course or alternatively, by eliminating some material, in a quarter course. Many options exist for what is not covered, depending on the emphasis required. For instance, for an emphasis in floating-point units, the most-detailed parts of Chapters 9 and 11 can be skipped; on the other hand, if the emphasis is on other applications, such as signal processing, it might be better to skip parts of Chapter 8. In our opinion, it would be best to cover the chapters in order, to make best use of the knowledge acquired before; however, other sequences are possible. For instance, the chapter on floating point could be covered earlier since it does not depend much on the details of previous chapters. The exercises at the end of the chapters allow for practice and extension of the material and can be used for design and implementation projects. The "Further Readings" sections and extensive bibliography provide material for additional self-study.

The book can also be used as a reference for designers of hardware for numerical applications. In this case, if they have not had a comprehensive course on the topic, the most profitable approach would be to study complete chapters, instead of only particular algorithms or implementations. This approach would provide the basis to experiment with alternative designs to choose the best for the particular requirements and constraints.

Additional Resources

The book is supported with a Web site (http://www.cs.ucla.edu/digital_arithmetic) that contains

- Appendix A: Material for instructors, consisting of solutions to all exercises, sample exams, and source files for lecture viewgraphs. This material will be available to instructors in a password-protected section of the Web site.
- Appendix B: One-third of solutions to exercises.
- Appendix C: Lecture viewgraphs associated with each chapter (in PS and PDF forms).
- Appendix D: Short notes on selected topics.
- Appendix E: Comments and errata.

Overview of Topics

The book begins with a review of basic material in terms of representations and algorithms for the basic operations (Chapter 1) and provides an introduction to the notation and description formats used. It then concentrates on a thorough presentation of alternative algorithms and implementations for addition/subtraction (of two and more than two operands), multiplication, division, and square root (Chapters 2–7). These algorithms and implementations can be directly used for fixed-point applications.

Chapter 8 concentrates on floating-point representation and on the corresponding algorithms and implementations. It contains an extensive discussion of alternative implementations for floating-point addition/subtraction and multiplication and describes the basic approaches to produce correctly rounded results in division and square root.

Chapter 9 presents serial arithmetic, both least-significant-digit first (LSDF) and most-significant-digit first (MSDF). The LSDF approach is effective for algorithms consisting only of additions and multiplications, whereas MSDF can be used for cases that include also division, square root, and comparisons. After considering the basic operations, the chapter illustrates their use in composite operations and in multimodule systems.

Chapters 10 and 11 discuss methods for function generation. Two main approaches are considered: (1) approximations based on multiplications, additions, and table lookup and (2) recurrences with linear convergence. The first approach

results in polynomial approximations (also included are methods based only on addition and table lookup) and is applicable to a large variety of functions. On the other hand, the second method is based on multiplicative and additive normalization and is practical only for some important functions, such as logarithm, exponential, sine, cosine, and arctan. In particular, the CORDIC algorithm presented in Chapter 11 is attractive for the multivariable functions rotation by an angle, modulus of a vector, and arctan(y/x). The discussion in that chapter also considers the generalization to hyperbolic and linear coordinates.

Topics Not Covered

Several major areas of digital arithmetic, such as residue number system arithmetic, logarithmic number system arithmetic, modular arithmetic, asynchronous multiplication and division, design for low-power arithmetic, arithmetic error codes, and verification and testing, are not included in this book. This does not imply that the omitted topics are less important than the ones presented; there will be short notes and a bibliography on the Web site.

Acknowledgments

We thank the many people who have influenced us in developing this book and, in particular, our colleague at UCLA, Algirdas Avizienis, for his work in digital arithmetic and contributions to the graduate course CS 252A (Arithmetic Algorithms and Processors). In addition, seminal works of James E. Robertson, Daniel E. Atkins, and Antonin Svoboda had a strong impact on our work.

We have benefited greatly from interactions and collaborations with numerous colleagues from academia and industry including Elisardo Antelo, Jean-Claude Bajard, Javier Bruguera, Neil Burgess, Luigi Dadda, Luigi Ciminiera, Jordi Cortadella, Warren Ferguson, Michael Flynn, David Goldberg, Mary Jane Irwin, Graham Jullien, William Kahan, Simon Knowles, Israel Koren, Peter Kornerup, Willy McAllister, David Matula, Paolo Montuschi, Jean-Michel Muller, Vojin Oklobdzija, Stott Parker, Michael Schulte, Renato Stefanelli, Earl Swartzlander, Naofumi Takagi, George Taylor, Alexandre Tenca, Arnaud Tisserand, Julio Villalba, and Dan Zuras. We thank them all and in particular those that reviewed the manuscript and provided constructive comments.

Our former and current students provided comments that helped us in developing this book: Charles Chien, Raffi Dionysian, John Fernando, Ian Ferguson, Abdolali Gorji-Sinaki, John Harding, Zhijun Huang, Jeong-A Lee, Marianne Louie, Robert McIlhenny, Peter Montgomery, Alberto Nannarelli, Vojin Oklobdzija, John Pipan, Alexandre Tenca, Paul Tu, Dean Tullsen, and Osaaki Watanuki. We thank them all for their suggestions and interest.

We have been very pleased working with our publisher, Morgan Kaufmann. Our thanks to our editor, Denise Penrose; editorial coordinator, Alyson Day; production manager, Jodie Allen; and production editor, Carol O'Connell for their effort and excellent guidance. The secretarial help of Terry Valai at UCLA has been invaluable and enjoyable.

This Page Intentionally Left Blank

Symbols and Notation

$\cdot(+)$	logical AND (logical OR)
$(p:$	a column of p bits
$:q]$	a row of q bits (weighted)
[3:2]	reduction of 3 to 2 digit-vectors ([3:2] adder; [3:2] carry-save adder (CSA))
[4:2]	reduction of 4 to 2 digit-vectors ([4:2] adder; [4:2] carry-save adder)
$(p:q]$	p-input, q-output counter
$[p:2]$	reduction of p to 2 digit-vectors ($[p:2]$ adder (compressor))
b	base of floating-point representation $x = M_x \times b^{E_x}$
B	bias in floating-point representation of exponent
CLA	carry-lookahead adder
CLG	carry-lookahead generator
CMPL	complementer
CPA	carry-propagate adder
CRA	carry-ripple adder
CS	carry-save form
CSA	carry-save adder
CSK	carry-skip adder

δ	on-line delay: number of initial cycles in online operation
δ	number of bits of estimate of divisor/argument in division/square root
EOP	effective operation
FA	full-adder
G	guard bit
HA	half-adder
INCR	incrementer
L_k (U_k)	lower (upper) boundary of selection interval
LOD	leading-one detection
LOP	leading-one prediction
LSDF	least-significant-digit-first mode of computation
LZA	leading-zeros anticipation
MAC	multiply-accumulate
MAF	multiply-add fused
MG	multiple generator
MSDF	most-significant-digit-first mode of computation
MUX	multiplexer (selector)
NAN	not-a-number
ovf	overflow condition
q_j	jth quotient digit
r	radix (base) of number representation
R	round bit
$\rho = a/(r-1)$	redundancy factor for maximum digit value a and radix r

REC	recoder
s_j	jth square root digit
SD	signed digit
T	sticky bit
ulp	unit in the last place
VAND (VOR)	vector AND (OR) gate
$w[j]$	residual
$\overline{w}(\underline{w})$	upper bound (lower bound) of w
WS (WC)	pseudosum (stored-carry) bit-vectors
$X = (x_{n-1}, \ldots, x_0)$	n-digit vector
$X[_j]$	digit-vector X at step (iteration) j
x_i	digit in the ith position of a digit-vector
$\{x\}_t$	x truncated to t fractional bits
$\overline{X}$	bit-complement of vector X
$\hat{y}$	low-precision estimate of the scaled residual $rw[j]$

This Page Intentionally Left Blank

CHAPTER 1

Review of Basic Number Representations and Arithmetic Algorithms

In this chapter we briefly review basic number representations and algorithms used in digital arithmetic. The treatment is very concise; readers that need a more detailed review should consult some of the references listed at the end of the chapter. More advanced algorithms as well as the implementations are the topic of later chapters.

1.1 Digital Arithmetic and Arithmetic Units

Digital arithmetic encompasses the study of number representations, algorithms for operations on numbers, implementations of arithmetic units in hardware, and their use in general-purpose and application-specific systems.

An *arithmetic unit (processor)* is a system that performs operations on numbers. We limit ourselves to the most common cases in which these numbers are

1. fixed-point numbers
   - integers $I = \{-N, \ldots, N\}$
   - rational numbers of the form $x = a/2^f$ ("binary" rationals), $a \in I$ and f positive integer
2. floating-point numbers $x \times b^E$, x rational number, b the integer base, and E integer exponent. The floating-point numbers approximate real numbers and facilitate computations over a wide dynamic range.

Collectively, we refer to these numbers as DA (digital arithmetic) numbers.

An arithmetic processor operates on one, two, or more *operands* depending on the operation. The operands are characterized by a *representation* and a *set* of values as defined in the next section. The *operation* is selected from an allowable set, which usually includes addition, subtraction, multiplication, division, square root, change of sign, comparison, and so on. The *results* can be DA numbers, logical variables (conditions), and/or singularity conditions (exceptions). Logical results occur for operations such as comparison, check for zero, and the like. Singularity conditions correspond to overflow, divide by zero, square root of a negative number, hardware error, and so on.

The *parameters* that describe the processor to the user include the number representation and precision, the operation set, the time required to execute each operation, the cost of the processor, and its energy consumption.

The function (*functional description*) of the arithmetic processor can be given at three levels:

1. **Abstract (mathematical) level.** The domain of operands and results is the set of numbers. The operations are specified as functions (sets of pairs). Also, some abstract properties such as commutativity, associativity, and distributivity can be given. At this level the objective is a functional specification (description). This is also known as high-level description. It has no implementation details.
2. **Arithmetic-algorithm level.** The numbers are represented by vectors of digits (digit-vectors), and the operations are described by algorithms composed of primitive operations (transformations) that are performed on these digit-vectors. This level provides a behavioral description, typically using arithmetic expressions and composition of functions. It introduces constraints affecting implementation.
3. **Implementation level.** The digit-vectors are encoded on bit-vectors. The operations are described by register-transfer algorithms. The description at this level is structural, specifying the modules, their interconnection, and the control flow.

In this text, we discuss arithmetic processors at the arithmetic-algorithm and implementation levels.

In the next section we introduce the basic number systems for fixed-point representation, which are used in the following chapters. Other representations

are discussed in later chapters together with their uses. We then present the basic algorithms.

1.2 Basic Fixed-Point Number Representation Systems

To perform operations on fixed-point numbers at the arithmetic-algorithm level, a specific number representation is required. In a *digital representation*, such a number is represented by an ordered n-tuple. Each of the elements of the n-tuple is called a *digit*, and the n-tuple is called a *digit-vector*. The number of digits n is called the *precision* of the representation. We begin with the representation of nonnegative integers, followed by the representation of signed integers, and concluding with an extension to fixed-point numbers.

1.2.1 Representation of Nonnegative Integers

The digit-vector that represents the integer x is denoted by

$$X = (X_{n-1}, X_{n-2}, \ldots, X_1, X_0) \qquad \textbf{1.1}$$

Note that we use a zero-origin, leftward-increasing indexing.

The *number system* to represent x consists of the following elements:

1. The number of digits n.
2. A set of (numerical) *values* for the digits. We call D_i the set of values of X_i. The cardinality of set D_i is denoted by $|D_i|$. For example, $\{0, 1, 2, \ldots, 9\}$ is the digit set for the conventional decimal number system with cardinality 10.
3. A rule of *interpretation*. This rule corresponds to a mapping between the set of digit-vector values and the set of integers.

There are many number systems differing in these elements.

The set of integers, each represented by a digit-vector with n digits, is a *finite set* with at most $K = \prod_{i=0}^{n-1} |D_i|$ different elements since this is the maximum number of different digit-vectors. For example, in a conventional decimal system a digit-vector of six digits can represent a million values. Sets that have been found

generally useful to perform basic arithmetic operations include, for example, all integers from 0 to $K - 1$.

A number system is *nonredundant* if each digit-vector represents a different integer; that is, if the representation mapping is one to one. It is *redundant* if there are integers that are represented by more than one digit-vector. Redundant number systems are sometimes used to reduce the complexity of the arithmetic algorithms and increase the speed of execution.

The number systems most frequently used are *weighted systems*. For them the representation mapping is

$$x = \sum_{i=0}^{n-1} X_i W_i \tag{1.2}$$

where $W = (W_{n-1}, \ldots, W_0)$ is the *weight-vector*.

A *radix number system* is a weighted number system in which the weight-vector is related to the *radix-vector* $R = (R_{n-1}, \ldots, R_0)$ as follows:

$$W_0 = 1; \quad W_i = W_{i-1} \cdot R_{i-1} \quad (1 \leq i \leq n-1) \tag{1.3}$$

This is equivalent to

$$W_0 = 1; \quad W_i = \prod_{j=0}^{i-1} R_j \tag{1.4}$$

Radix number systems are classified according to the radix-vector into fixed-radix and mixed-radix systems.

In a *fixed-radix* system all elements of the radix-vector have the same value r (the radix). Consequently, the weight vector is

$$W = (r^{n-1}, \ldots, r^2, r, 1) \tag{1.5}$$

and

$$x = \sum_{i=0}^{n-1} X_i \cdot r^i \tag{1.6}$$

The most frequently used radices are powers of two, such as 2 (binary), 4 (quaternary), 8 (octal), and 16 (hexadecimal). The corresponding weight-vectors are $W = (\ldots, 16, 8, 4, 2, 1)$ for $r = 2$, $W = (\ldots, 256, 64, 16, 4, 1)$ for $r = 4$, and so on. The other radix that is sometimes used is 10 (decimal); this is done because of our familiarity with this representation and because the interface with humans is more convenient in decimal. Because some arithmetic algorithms are simpler in binary than in decimal, in many systems the input-output is decimal but the internal processing is done in binary. Conversion is therefore required between these representations.

In a *mixed-radix* system the elements of the radix-vector are different. For example, the representation of time in terms of hours, minutes, and seconds in a 24-hour period uses a radix-vector $R = (24, 60, 60)$. The corresponding weight-vector is $W = (3600, 60, 1)$. Consequently, the digit-vector $X = (5, 37, 43)$ represents 20,263 seconds.

According to the *set of digit values*, the radix number systems are classified into canonical and noncanonical systems.

In a *canonical system* the set of values for D_i is $\{0, 1, \ldots, R_i - 1\}$ with $|D_i| = R_i$. For example, the canonical digit sets in the binary, quaternary, octal, and hexadecimal number systems are respectively $\{0, 1\}$, $\{0, 1, 2, 3\}$, $\{0, 1, 2, \ldots, 7\}$, and $\{0, 1, 2, \ldots, 15\}$. The corresponding range of values of x represented with n radix-r digits is

$$0 \leq x \leq r^n - 1 \qquad \textbf{1.7}$$

In a *noncanonical system* the set of digit values is not canonical. For example, $D_i = \{-4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ is a digit set in a noncanonical decimal system and $\{-1, 0, 1\}$ and $\{0, 1, 2\}$ in noncanonical binary systems.

A noncanonical digit set D_i such that $|D_i| > R_i$ produces a redundant system allowing more than one representation of a value; for example, in the $\{-1, 0, +1\}$ binary system the vectors $(1, 1, 0, 1)$ and $(1, 1, 1, -1)$ both represent the integer "thirteen."[1]

A system with fixed positive radix r and canonical set of digit values is called a *radix-r conventional number system*. These are by far the most commonly

1. To distinguish the integer from its radix-10 representation, we give the name of the number as its decimal representation in letters.

Number System	Digit Vector
Conventional radix-2 system (binary)	0011110
Conventional radix-3 system	0001010
Conventional radix-4 system	0000132
Conventional radix-10 system	0000030
Radix-2 system with digit set $\{-1 = \bar{1}, 0, 1\}$	0011110
	$0100\bar{1}0$
Residue system with $P = (17, 13, 11, 7, 5, 3, 2)$	(13)482000

TABLE 1.1 Representations of the integer "thirty."

used number systems. As indicated, the favored radices are powers of 2 and 10 (decimal). In the following sections we discuss algorithms for these systems emphasizing the conventional binary system.

There exist also *nonradix number systems* in which weights are not defined recursively as in (1.3). One example is the *residue number system* (RNS), where for a given set of pairwise relatively prime integers $P = (P_{n-1}, \ldots, P_0)$, a positive integer x (for $0 \leq x < \prod_{i=0}^{n-1} P_i$) is represented by the vector X such that

$$X_i = x \bmod P_i \tag{1.8}$$

This is a nonredundant system that allows fast implementation of addition and multiplication. In this system there is no notion that the digits on the left are more significant than the digits on the right. In that sense, there is no notion of "weight," and RNS is sometimes classified as a nonweighted system. As an example, we represent in Table 1.1 the integer "thirty" in several number systems using a digit-vector with seven components.

In this text we use fixed radix and mainly radix 2.

Bit-Vector Representation

For the implementation of arithmetic algorithms in (binary) digital systems, it is necessary to represent the digit-vectors by bit-vectors. This is done by defining a *code* for a digit and mapping the digit-vector by mapping each digit according to this code.

In the binary (conventional) number system, the code is direct: the binary-digit values 0 and 1 are represented by the binary-variable values 0 and 1, respectively.

For higher power-of-two radices the most common code is the binary code in which a digit d is represented by a bit vector $(d_{k-1}, \ldots, d_0)$ of $k = \log_2 r$ bits such that

$$d = \sum_{i=0}^{k-1} d_i 2^i \qquad \textbf{1.9}$$

The use of this code for each digit results in a bit-vector for x that is the same for any power-of-two radix, the only difference being the way the bits are grouped to form a digit. In the binary case, each bit corresponds to a digit, while in the radix-r case, groups of $\log_2 r$ bits form a digit. Therefore, conversion from a bit-vector in a radix-2 representation to a radix-r representation and vice versa is trivial. For example, the bit-vector

$$\begin{aligned} X &= (1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1) \\ &= ((1, 1, 0), (0, 0, 1), (0, 1, 1), (1, 0, 1)) \\ &= ((1, 1, 0, 0), (0, 1, 0, 1), (1, 1, 0, 1)) \end{aligned} \qquad \textbf{1.10}$$

corresponds to the octal digit-vector (6, 1, 3, 5) and the hexadecimal digit-vector $(C, 5, D)$.[2]

The fact that the bit-vectors are identical permits the use of some binary algorithms to perform operations on integers represented in these higher radices.

1.2.2 Representation of Signed Integers

In the previous section we presented the representation of nonnegative integers. We now extend the discussion to the representation of signed integers (positive and negative). Two representations are by far the most common: the sign-and-magnitude representation and the true-and-complement representation; these are the topic of this section.

---

2. The integers 10, 11, ..., 15 are denoted with letters A, B, ..., F, respectively.

Sign-and-Magnitude (SM) System

A signed integer x is represented in the SM system by a pair (x_s, x_m), where x_s is the *sign* and x_m is the *magnitude* (positive integer). The two values of the sign $(+, -)$ are represented by a binary variable, where traditionally 0 corresponds to $+$ and 1 to $-$.

The magnitude can be represented by any system for the representation of positive integers. If a conventional radix-r system is used, the range of signed integers, for n digits in the representation of the magnitude, is

$$0 \leq x_m \leq r^n - 1 \qquad \textbf{1.11}$$

Note that zero has two representations: $x_s = 0, x_m = 0$ (positive zero) and $x_s = 1, x_m = 0$ (negative zero).

True-and-Complement (TC) System

In the true-and-complement system there is no separation between the representation of the sign and the representation of the magnitude, but the whole signed integer is represented by a positive integer. Consequently, this representation involves an additional mapping as indicated in Figure 1.1.

The signed integer x is represented by a positive integer x_R, which in turn is represented by the digit-vector X. Map 2 defines the mapping between integers

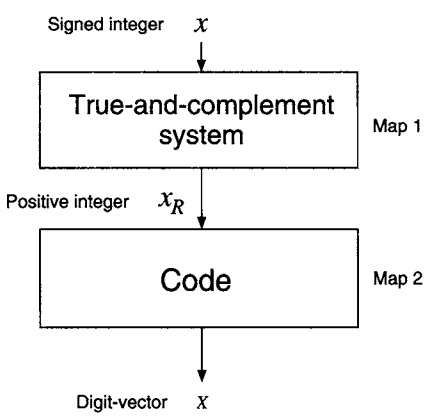

FIGURE 1.1 Signed integer represented by positive integer.

and digit-vectors as discussed in the previous section. We now define the mapping Map 1 for the true-and-complement system.

A signed integer x is represented in the true-and-complement system by a positive integer x_R such that

$$x_R = x \bmod C \tag{1.12}$$

where C is a positive integer, called the *complementation constant*. By the definition of the mod function, for $\max |x| < C$, this is equivalent to

$$x_R = \begin{cases} x & \text{if } x \geq 0 \\ C - |x| = C + x & \text{if } x < 0 \end{cases} \tag{1.13}$$

In order to have an unambiguous representation, the region for $x > 0$ should not overlap with the region for $x < 0$. This requires that

$$\max |x| < C/2 \tag{1.14}$$

The converse mapping is

$$x = \begin{cases} x_R & \text{if } x_R < C/2 \\ x_R - C & \text{if } x_R > C/2 \end{cases} \tag{1.15}$$

When $x_R = C/2$ is representable, it is usually assigned to $x = -C/2$, making the representation asymmetrical.

The representations of positive integers are called *true forms*, and those of negative integers, *complement forms*.

The positive integer x_R can be represented in any system for positive integers. For a digit-vector of n digits, the range is

$$0 \leq x_R \leq r^n - 1 \tag{1.16}$$

The usual choices for the complementation constant are $C = r^n$ (*Range Complement System (RC)*) and $C = r^n - 1$ (*Digit Complement System (DC)*). We now consider a radix-2 representation and leave as an exercise the more general radix-r case.

The choice $C = 2^n$ defines the *two's complement* system. The corresponding mapping is illustrated in Table 1.2. In this system the value $x_R = C$ is outside the range, and, therefore, there is only one representation of $x = 0$. The value $x_R = 2^{n-1}$ could represent either $x = 2^{n-1}$ or $x = -2^{n-1}$, resulting in an asymmetric representation. It is usual to make the second choice in order to

x	x_R	
0	0	
1	1	
2	2	
–	–	True forms
–	–	(positive)
–	–	$x_R = x$
$2^{n-1} - 1$	$2^{n-1} - 1$	
-2^{n-1}	2^{n-1}	
$-(2^{n-1} - 1)$	$2^{n-1} + 1$	
–	–	
–	–	Complement forms
–	–	(negative)
-2	$2^n - 2$	$x_R = 2^n - \|x\|$
-1	$2^n - 1$	

TABLE 1.2 Mapping in the two's complement system.

simplify the sign detection, as discussed later in Section 1.2.3. The range of signed integers is

$$-2^{n-1} \leq x \leq 2^{n-1} - 1 \qquad \textbf{1.17}$$

The choice $C = 2^n - 1$ defines the *ones' complement* system. The corresponding mapping is shown in Table 1.3. In this system $x_R = C$ is representable with n digits so that there are two representations of $x = 0$: $x_R = 0$ and $x_R = 2^n - 1$.

The range of signed integers is

$$-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1 \qquad \textbf{1.18}$$

EXAMPLE 1.1 Represent $-4 \leq x \leq 3$ in the two's complement and ones' complement systems. The mappings

$$x \rightarrow x_R \rightarrow X$$