The Principles of Switching Circuits

Frederick H. Edwards

AN MIT PRESS CLASSIC

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

The Principles of Switching Circuits

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

The M. I. T. Press
Cambridge, Massachusetts, and London, England

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

The Principles of Switching Circuits

Frederick H. Edwards

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

First MIT Press paperback edition, 2003

Copyright © 1973 by
The Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

This book was designed by The MIT Press Design Department.
It was set in Univers Medium
by Science Press.

Library of Congress Cataloging in Publication Data

Edwards, Frederick H.
The principles of switching circuits.

Includes bibliographies.
1. Switching theory. I. Title.
TK7868.S9E34 621.3815'37 72-10269
ISBN 978-0-262-05011-1 (hc : alk. paper)
978-0-262-55044-4 (pb : alk. paper)

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

To the memory of Carl S. Roys
scholar and friend

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Contents

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Preface

This book is the outgrowth of a course in switching circuits that the writer has taught since 1960. The original class notes for the course were written in an attempt to provide a unified treatment of sequential circuit theory.

An effort has been made to include only those techniques that have been generally accepted and that will have lasting application. No attempt has been made to include current hardware circuit realizations since the continuous advance of technology tends to render specific hardware rapidly obsolete. Circuit realizations are considered only in terms of gate symbols.

The subject matter is organized into nine chapters, one introductory, three on combinational circuit theory, and five on sequential circuit theory.

An introduction to digital systems, number systems, and binary codes is given in Chapter 1, and Boolean algebra and switching functions along with algebraic techniques for the manipulation and minimization of the algebraic expressions are presented in Chapter 2.

Chapter 3 covers the analysis and synthesis of combinational gate circuits and includes the topic of functional completeness. The subject of threshold logic is considered in some detail. In Chapter 4 geometric and tabular techniques are presented for the minimization of algebraic expressions. These techniques are limited to two-level logic realizations since general techniques for multilevel logic realizations are not currently available.

The underlying objectives in the presentation of the material on sequential circuits are to provide a unified view of the various modes in which these circuits can be operated and to provide general techniques for their analysis and synthesis. The author believes that the former is best achieved by classifying the circuit operation as either fundamental mode or pulse mode, and as either clocked or not clocked. Both algebraic and tabular techniques are presented for the analysis and synthesis of these circuits.

The order of presentation of the sequential topics has been chosen to provide a logical sequence. Since the reader has, in his study of combinational circuit theory, become familiar with input signals whose duration in either state is not limited, it is natural to proceed first to a study of sequential circuits that operate in fundamental mode. Further, since he has thus far studied only gate circuits, it appears natural to proceed first with sequential circuits that are realized using only gate elements and where memory is achieved through feedback. The theory can then be extended to circuits that use specific memory elements (which are themselves gate circuits with feedback).

A general approach is emphasized by showing that sequential circuits designed

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

for operation in either fundamental or pulse mode and either clocked or not clocked can each be realized either using gates alone (with the addition of suitable delays) or using a combination of gates and memory elements.

The analysis of sequential circuits is considered first and presented in Chapter 5. Although the techniques are shown explicitly for circuits that operate in fundamental mode, they apply also to circuits that operate in pulse mode. The only difference lies in the interpretation of the final table. Both algebraic and tabular techniques are introduced.

Chapter 6 is concerned with the synthesis of sequential circuits for operation in fundamental mode. Since the flow tables for such circuits are characteristically derived with redundant rows, it is desirable to reduce them before their realization is attempted, and for this reason a simplified technique for flow-table reduction is introduced at this point. Static and essential hazards are included in this chapter.

In Chapter 7 synthesis techniques are presented for circuits that operate in pulse mode. The flow table for this mode of operation is derived as a modification of the flow table for fundamental-mode operation. The motive here is to obtain a circuit design that requires fewer memory states. A comparison of the two modes of sequential circuit operation is made. The distinguishing characteristic of pulse-mode operation is stressed, and several methods are presented for obtaining this characteristic when flip-flops are used in the circuit realizations.

Chapter 8 covers the clocked or synchronous operation of sequential circuits and includes circuits for operation in both fundamental mode and pulse mode. Use of the same problem specifications enhances the comparison of circuits designed for operation in each mode; for example, the fact that circuits designed for operation in pulse mode often require fewer memory elements is clearly evident. Both algebraic and tabular design techniques are included.

A unified treatment of the two modes of sequential circuit operation is facilitated by the use of only one type of flip-flop in the presentation of the sequential theory. The type RS flip-flop is used since it is the only flip-flop in current use that can be operated unclocked in either mode. Other types of flip-flops are considered in Chapter 8 after clocked operation has been presented, and examples are given in which the flip-flop input-gating costs are compared and related to the form of the next-state equations.

Lastly, techniques are presented for the realization of sequential circuits when the circuit specifications are given in terms of control states and register transfers.

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

In Chapter 9 the subject of sequential circuit minimization is considered. A more general technique is presented for the reduction of flow tables, and the state-assignment problem is discussed.

The material in this book can essentially be covered in a single-semester advanced undergraduate course. Experience has shown that laboratory experimentation with switching devices provides valuable aid to the reader's understanding of the mathematical models. Answers are provided to selected problems.

Frederick H. Edwards
Amherst, Massachusetts

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

1

Introduction

Switching theory is concerned with the development of models and techniques useful in the analysis and synthesis of circuits for which the information is represented in discrete or digital form. This form is opposed to the analog form in which information is represented in a continuous manner.

Insight into the fundamental difference between analog and digital representation of information will be gained if we consider the representation of the independent variable x over some closed interval. In an analog system this variable would be represented by a physical quantity such as a voltage level that is continuously variable over a range of values. In a digital system the interval for the variable x would be partitioned into a finite number of subintervals each of which is placed in correspondence with a discrete *state* that the system can assume. An example of a discrete circuit device is provided by the ordinary two-position switch that can assume either of two states—either open or closed. A system in which n such switches are interconnected could provide a total of 2^n different states that the circuit could assume.

Before proceeding with a study of switching theory it is desirable to have first a brief look at some of the areas in which digital circuits have application and at some subject areas a knowledge of which will prove useful later.

A rapid application of digital techniques is currently occurring over a wide range of human endeavor and is likely to continue since their range of application does not appear to be limited. One of the largest and most spectacular examples of a digital system is still provided by the dial telephone system, where a great many decisions are continuously made and taken for granted each moment of the day, determining call destinations, available interconnecting linkages, call dispositions such as the return of busy signals, call completions, cost allocations, and the provision of fault information should a system failure occur. However, as in the past, it is the use of digital data-processing machines that is stimulating the greatest interest in research in and application of digital techniques. The question might be asked: Why digital?

The answer lies in the fact that digital data-processing techniques provide reliable and accurate interpretation and processing of data combined with high rates of processing. An important result is the capability of decision making in real time while an activity is in actual progress, whether it be some industrial process or a spacecraft in flight. The precision of digital processing techniques results

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

from error-free interpretation and manipulation of digital data. The representation of data in digital form requires the quantization of time-varying data; this is commonly achieved by assigning levels of quantization nearest the value of the function at instants of time defined by a timing source or clock. If the data are quantized in binary form, the possibility of error in data interpretation is reduced to a minimum since it is necessary to distinguish only between two extreme data values. Once data are in binary form they can be processed without the compounding errors that occur in a nonquantized data system (such as those due to calibration errors). The end result of a set of operations on binary data will always be the same no matter how many times it is repeated. This is an important factor, for example, in the processing of commercial data where results are required not to within some tolerance of error but to a precise figure.

The need for reliability plays an important part in the choice of digital systems as digital devices continue to improve. Examples are provided by current space programs: reliability of space communication is mandatory where the mission duration is measured in months or years, and reliability of control is mandatory where human safety is involved. An example of the latter is provided by the use of a digital data processing link to close the control loop in the Apollo Command and Lunar flight control system. A variety of operating modes is required for this system, such as a checkout mode, primary mode, backup mode, pilot-control mode, and so on. With an analog system the mode switching would involve the changing of connections with an increase in component cost and a decrease in the reliability estimate, whereas with a digital system the mode switching is accomplished by branching to another stored program. In the latter case the high reliability of fixed memory does not decrease the reliability estimate (Miller 1966).

Digital processing of information requires that the information be numerically coded. For this reason an introduction is included in this chapter on the fundamental characteristics of number systems in general and on the binary number system in particular. Since there are applications where the number of calculations per data set is small but the number of data sets large, it is desirable to facilitate the input and output of data rather than the calculations themselves. This is accomplished by a binary coding of each decimal digit instead of a conversion of the decimal number to its binary equivalent; a discussion of binary-coded-decimal numbers is therefore included. Finally since analog-digital conversion forms a necessary adjunct to many digital applications, a discussion of cyclic codes is included.

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

1.1 Number Representation

An efficient number representation utilizes what is called positional notation. The concept of positional notation was actually used in the ancient computing device known as the abacus but was not fully appreciated until centuries later. In positional notation the value of a numerical symbol depends not only on the symbol itself but also upon its position. The advantage of this notation is that new symbols do not have to be invented as numbers become progressively larger. An example where positional notation is not used is provided by the Roman numeral system. In this system large numbers must be represented by a large number of repetitions of given symbols. The system is cumbersome but has the advantage of simple arithmetic—addition or subtraction requires no memorization of addition tables and is not concerned with carry or borrow generation. A historical account of the various number systems that have been used by mankind is given by Gardner (1968).

In our familiarity with decimal numbers we tend to forget that the digits have positional characteristics. If we express the number $(371.625)_{10}$ as a polynomial in powers of 10,

$$(371.625)_{10} = 3 \times 10^2 + 7 \times 10^1 + 1 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3},$$

then we see that each digit position has a *weight* associated with it. The position of digit 3, for example, has the weight 10^2. The number 10 from which the weights are derived is called the *base* or *radix* of the number system.

Numbers other than 10 can be used to determine the weights in positional notation. In general a number (N) is expressed in a base-r positional number system as

$$(N)_r = (b_i b_{i-1} \cdots b_0 \,.\, b_{-1} \cdots b_{-j})_r,$$

where r is an integer > 1, and b_i is an integer whose value is given by the relation $0 \leqslant b_i \leqslant r - 1$. The symbol "." is called the *radix* point. The set of juxtaposed symbols is a shorthand notation for the polynomial

$$(N)_r = b_i r^i + b_{i-1} r^{i-1} + \cdots + b_0 r^0 + b_{-1} r^{-1} + \cdots + b_{-j} r^{-j}.$$

Two number systems useful in computing machines are the octal and binary number systems. The octal number system has eight symbols, 0 through 7, and

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Table 1.1 Binary and Octal Equivalents of the Decimal Digits

Decimal	Octal	Binary
0	00	0000
1	01	0001
2	02	0010
3	03	0011
4	04	0100
5	05	0101
6	06	0110
7	07	0111
8	10	1000
9	11	1001

the binary number system has two symbols, 0 and 1. Table 1.1 shows the octal and binary representations of the decimal digits.
The decimal number $(371.625)_{10}$ is represented in the octal number system as $(563.5)_8$, and in the binary number system as $(101110011.101)_2$.

1.2 Number-System Conversion

It is frequently necessary to be able to convert a number from one base to another. This can be achieved by expressing the number as a polynomial in the given number system and then evaluating this polynomial using the arithmetic of the desired number system. For example, the decimal equivalent of the octal number 563.5 is given by the summation

$$\begin{aligned}(N)_{10} &= 5 \times 8^2 + 6 \times 8^1 + 3 \times 8^0 + 5 \times 8^{-1} \\ &= 320 + 48 + 3 + .625 \\ &= (371.625)_{10}.\end{aligned}$$

and that of the binary number 101110011.101 by the summation

$$\begin{aligned}(N)_{10} &= 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 \\ &\quad + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 256 + 64 + 32 + 16 + 2 + 1 + .5 + .125 \\ &= (371.625)_{10}.\end{aligned}$$

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

The conversion of a number from one base to another can also be obtained by an iterative algorithm involving repeated division and multiplication by the base of the number system *to* which we wish to convert. The arithmetic is performed in the number system *from* which we wish to convert, and the integral and fractional parts are obtained separately. The procedure is best illustrated by conversion from the decimal system. We consider the integral part first.

The integral part of the new number representation is obtained by repeated division of the decimal integral part by the base of the new number system; the remainders resulting from each division form the new base digits. For example, the decimal number 371 is converted to its equivalent octal representation as follows:

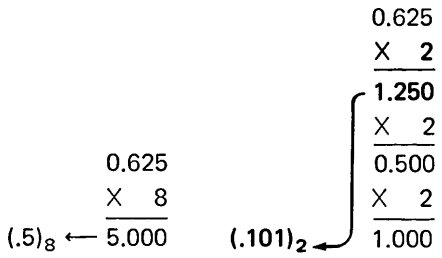
$371 \div 8 = 46$ with remainder 3
$46 \div 8 = 5$ with remainder 6
$5 \div 8 = 0$ with remainder 5
$\rightarrow (563)_8$.

The remainders form the coefficients of increasing powers of 8; the coefficient 3 represents the number of units (8^0); the coefficient 6 represents the number of eights (8^1); and so forth. The conversion of $(371)_{10}$ to its equivalent binary representation is as follows:

$371 \div 2 = 185$ with remainder 1
$185 \div 2 = 92$ with remainder 1
$92 \div 2 = 46$ with remainder 0
$46 \div 2 = 23$ with remainder 0
$23 \div 2 = 11$ with remainder 1
$11 \div 2 = 5$ with remainder 1
$5 \div 2 = 2$ with remainder 1
$2 \div 2 = 1$ with remainder 0
$1 \div 2 =$ **0 with remainder 1**
$\rightarrow$ **$(101110011)_2$.**

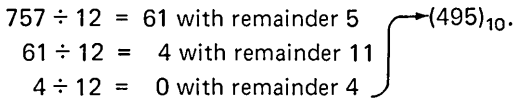
The fractional part of the new number representation is obtained by repeated multiplication of the decimal fraction by the new base. The integral values $(0 \leqslant b_0 \leqslant r - 1)$ obtained for each product form the new digits, and the digits are formed in the order of decreasing significance. The integral part of each product is disregarded in the succeeding multiplication. Thus, the conversion of $(.625)_{10}$ to its octal and binary equivalents is performed as follows:

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024



It should be noted that it is not in general possible to convert a terminating fraction in one number system to a terminating fraction in another number system.

As an example of number-system conversion using nondecimal arithmetic consider the conversion of the octal number 757 to its equivalent decimal representation using octal arithmetic:



If we prefer to avoid arithmetic in unfamiliar number systems, we can first convert the number to the decimal number system by evaluation of the polynomial, and then to the desired number system using repeated division and/or multiplication.

Conversion from one number system to another becomes very simple when the base of one number system is an integral power n of the base of another since each combination of the n digits of the latter corresponds to one digit of the former. The conversion can be obtained by inspection: in the conversion from binary to octal the binary digits are grouped in sets of three starting at the binary point (in either direction), and each set is then replaced by its octal equivalent; in the conversion from octal to binary each octal digit is replaced by its equivalent of three binary digits.

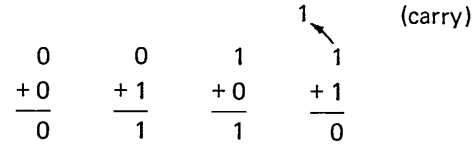
1.3 Binary Arithmetic

In this section, we consider some elementary arithmetic operations using binary arithmetic. More-detailed treatments of the arithmetic techniques used in computers are given by Chu (1962) and Richards (1971).

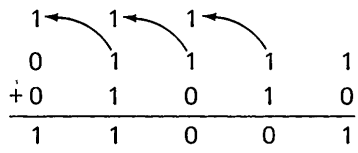
Binary addition follows rules similar to those we are accustomed to using in decimal addition; these are shown below for single-digit binary numbers. We see

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

that the sum digit is 1 whenever only one of the digits has the value 1 and that a carry of 1 is formed whenever both digits have the value 1.

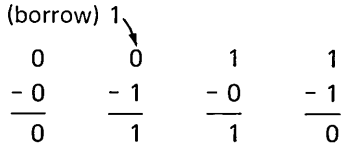


As an example of multiple digit binary addition consider the addition of $(15)_{10}$ and $(10)_{10}$ shown next. The arrows indicate carries of 1 into the next more significant digit or *bit* positions. Note that a carry of 1 is first formed only when both the augend and addend digits have the value 1, but that once formed, additional carries continue to be generated so long as either the augend or addend digit, or both, has the value 1.



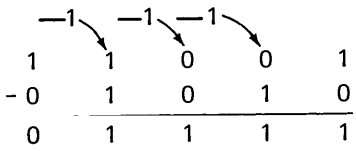
A general rule for adding a column of digits (in any base) is to add the digits decimally and then divide by the base. The sum is given by the remainder and the carry by the quotient.

The rules for binary subtraction are now given. In subtracting 1 from 0, we borrow from the next more significant digit position.



In multiple digit subtraction there can never be a borrow from the least-significant digit position just as in addition there can never be a carry into the least-significant digit position. The binary subtraction of $(10)_{10}$ from $(25)_{10}$ follows. Notice that the borrow from the mid-digit position results in borrows from the two remaining digit positions.

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024



A problem arises in subtraction when the subtrahend is larger than the minuend. In longhand calculation this situation is dealt with by subtracting the smaller number from the larger and affixing a negative sign to the difference. The representation of negative numbers by sign and magnitude, however, may not be the most convenient form for computing machines. A number system using *complement* representation of negative numbers permits the implementation of both addition and subtraction using only one or the other of these operations.

Since the number of digit positions in any machine computation is finite, we can obtain a negative number representation from the fact that the sum of a number and its negative must be zero. If we subtract an n-digit binary number from 2^n we obtain what is called the *true* or 2's complement of the number. For example, if we represent the number $(10)_{10}$ by the five-digit binary number 01010, then the 2's complement representation of $-(10)_{10}$ is obtained by subtracting 01010 from 2^5 as follows:

$$\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ - & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

The five-digit negative representation is then 10110. In the complement number system the leftmost bit, called the *sign-bit*, is 1 for negative numbers and 0 for positive numbers. The 2's complement representation of a binary number is conveniently obtained by complementing each binary digit (1's are replaced by 0's and vice versa) and then adding 1 in the least-significant digit position. The subtraction of $(10)_{10}$ from $(15)_{10}$ can now be obtained by the addition of $-(10)_{10}$ to yield $(00101)_2$ or $(5)_{10}$; the most significant 1-bit is disregarded.

$$\begin{array}{cccccc} & 0 & 1 & 1 & 1 & 1 \\ + & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Another complement number system, called the *diminished-radix* or 1's com-

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

plement system, is obtained if we subtract the binary number from $2^n - 1$. Thus, the 1's complement of $-(10)_{10}$ is obtained as

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1 \\ -\,0\ 1\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 0\ 1 \end{array}$$

In this case the subtraction of $(10)_{10}$ from $(15)_{10}$ is obtained as follows:

$$\begin{array}{r} 0\ 1\ 1\ 1\ 1 \\ +\,1\ 0\ 1\ 0\ 1 \\ \hline \boxed{1}\ 0\ 0\ 1\ 0\ 0 \\ \qquad\qquad\qquad\qquad\ \ \rightarrow 1 \\ \hline 0\ 0\ 1\ 0\ 1 \end{array}$$

A disadvantage of the 1's complement system is that an *end-around* carry, generated out of the sign-bit position, may be necessary. A further disadvantage is that there are two representations for zero: all 1's if it is obtained as a result of addition, and all 0's if it is obtained as a result of subtraction. An advantage is that the complement is obtained by simply complementing each binary digit.

We see then that the complement representation of negative numbers enables us to subtract using addition and to add using subtraction.

Multiplication in the binary system also follows the same rules as used in the decimal system, but since the digits can only have the value 0 or 1, it is much simpler. The process is illustrated here for the multiplication of $(12.5)_{10}$ by $(5.0)_{10}$. The location of the binary point is determined in the usual manner by adding the number of digits to the right of the binary point in the multiplicand and multiplier.

$$\begin{array}{r} 1100.1 \\ 101.0 \\ \hline 0000\ 0 \\ 11001 \\ 00000 \\ 11001 \\ \hline 111110.10 \end{array}$$

Note that the product can be obtained in iterative fashion by successive shift-

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

and-add operations. The multiplication of negative numbers is commonly obtained by forming the product of the magnitudes with the sign determined separately.

Division is similar to multiplication in that both operations involve three numbers, one of which is the product of the other two. In binary multiplication we saw that the product could be obtained by the iterated addition of the multiplicand and zero corresponding to 1's and 0's in the multiplier. It would appear to follow that the quotient in division could be obtained in similar fashion by iterated subtraction of the divisor or zero from the dividend. However, a difficulty arises in knowing which should be subtracted. In longhand division, this knowledge is obtained through trial and error or simply by inspection where binary arithmetic is used. An algorithmic approach to the problem is to first subtract the divisor and then note whether the remainder is positive or negative. A positive remainder implies a quotient digit of 1 and a negative remainder a quotient digit of 0. In the latter case the subtraction can be nullified by adding the divisor back into the remainder. The divisor is then shifted one position to the right and the procedure repeated.

In another and shorter procedure, called *nonrestoring division*, a negative remainder is followed by a right shift and addition. The result is the same either way, since in the first procedure a negative remainder is followed by the operations $+D$ and $-D/2$ (where D is the divisor and a shift right of one position corresponds to division by two) and in the second by the operation $+D/2$. The technique is illustrated by the binary division of $(35)_{10}$ by $(10)_{10}$.

Before proceeding with the foregoing technique, let us first, for comparative purposes, divide by hand as shown here:

```
          011.1
1010 √100011.0
       1010
       ----
        1111
        1010
        ----
         1010
         1010
         ----
         0000
```

In nonrestoring division the first subtraction is performed with the leftmost digits of the dividend and divisor aligned as shown:

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

	011.1	
	1010 √100011.0	
Subtract	1010	
Negative remainder	(−) 0001010	$q_0 = 0$
Shift and add	1010	
Positive remainder	011110	$q_1 = 1$
Shift and subtract	1010	
Positive remainder	01010	$q_2 = 1$
Shift and subtract	1010	
Positive remainder	0000	$q_3 = 1$

We see that a negative remainder is followed by a shift-and-add operation, whereas a positive remainder is followed by a shift-and-subtact operation. Note that the remainders corresponding to each quotient digit of $q_i = 1$ are the same as those obtained in the longhand division. In the interest of clarity, negative numbers have been represented by sign and magnitude in this example. In machine computation, the 2's complement representation is commonly used (Richards 1971)—see problem 1.15.

1.4 Binary Codes

A code may be regarded as a system of symbols arbitrarily used to represent alphabetical, numerical, or other symbols. In this section, we will consider only binary codes and will limit our discussion of these to include only computational and cyclic codes.

The term computational codes here refers to the binary codes used in the coding of individual decimal digits when it is desired to facilitate machine input and output of data rather than the calculations on the data. The conversion of decimal numbers to binary-coded representations is performed much more readily than is their conversion to equivalent binary numbers. It is necessary to use a minimum of 4 binary digits to code the 10 decimal digits since there are only $2^3 = 8$ different combinations of 3 binary digits. The 4 binary digits provide $2^4 = 16$ different combinations of which only 10 are used; the 16 combinations provide $16!/6! \approx 2.9 \times 10^{10}$ different code sets. Only a few of these code sets have actually been used and for special reasons. It is desirable that the decimal digit represented can be determined directly from the binary code digits. Such codes are generally *weighted*, each digit position in the group carrying an assigned weight. The decimal digit represented is then given by the summation

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

Table 1.2 Binary-Coded-Decimal Representations

Dec. No.	8421	2421	735-6	Excess-3	2 Out of 5
0	0000	0000	0000	0011	00011
1	0001	0001	1001	0100	00101
2	0010	0010	0111	0101	00110
3	0011	0011	0010	0110	01001
4	0100	0100	1011	0111	01010
5	0101	1011	0100	1000	01100
6	0110	1100	1101	1001	10001
7	0111	1101	1000	1010	10010
8	1000	1110	0110	1011	10100
9	1001	1111	1111	1100	11000

$\Sigma_{i=0}^{3} b_i w_i$, where the b_i are the binary digits and the w_i are the assigned weights. A weighted code may use all positive weights, or it may use a combination of both positive and negative weights. A number of weighted and unweighted binary codes is shown in table 1.2.

In the 8421 code each decimal digit is represented by its exact binary equivalent—the four weights are derived from 2^3, 2^2, 2^1, and 2^0. For this particular code each decimal digit is uniquely represented. All other positive-weighted codes provide a choice of coding for some decimal digits. The 2421 code provides a choice of coding for the digits 2 through 7; the digit 3, for example, can be coded eigher as 0011 or as 1001.

The choice of code groups shown in table 1.2 for the 2421 code has the useful computational property (in subtraction) that the code for the 9's complement of a decimal digit is obtained by simply complementing each binary digit of the decimal digit representation. For example the binary code for decimal 6 (9's complement of decimal 3) is given as 1100 and this is obtained by complementation of the binary code 0011 for decimal 3. If the code for decimal 3 has been chosen as 1001 then its complement (0110) would not be the binary code for decimal 6. Codes for which the representation of the 9's complement of a decimal digit can be obtained by complementation of each binary digit are called *self-complementing* codes. The 753-6 code is an example of a code employing both positive and negative weights that is self-complementing. A necessary requirement for a weighted code to be self-complementing is that the algebraic sum of the weights is 9.

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

The excess-3 code is a nonweighted code which is self-complementing. It is formed by adding 3 to each of the 8421 code groups, and has the advantage of providing immediate carry information.

The last code uses five binary bits and has the advantage that a single digit error produces an invalid code that is easily detected. More complex codes can be constructed for multiple error detection, error correction, or both (Peterson and Weldon 1972).

In the conversion of continuous data into binary form it is desirable to use what are called *cyclic* codes. These codes have the characteristic that successive code groups differ in only one digit position. The advantage of such codes lies in the avoidance of spurious incorrect coding. Consider, for example, the encoding of the angular position of a shaft by means of 4 binary bits. It is not possible to build sensing devices such that changes in each bit position always occur simultaneously. As a result large errors could occur. Suppose that successive angular positions of the shaft are to be coded in the binary number system and that the shaft position is such that the binary code should change from 0111 to 1000. Then a momentary change of only the most significant bit would result in the generation of the 1111 binary code, which would indicate a change in shaft position of half a revolution! If only one bit position is ever required to change from one code to the next, then such errors cannot occur.

A particularly useful cyclic code is the Gray or *reflected binary* code shown in table 1.3.

This code can be converted to the binary code, or vice versa, by means of single-digit binary addition. For example, the conversion of the binary code to the Gray code proceeds as follows: Starting in the least-significant position each digit

Table 1.3 The Gray Code

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

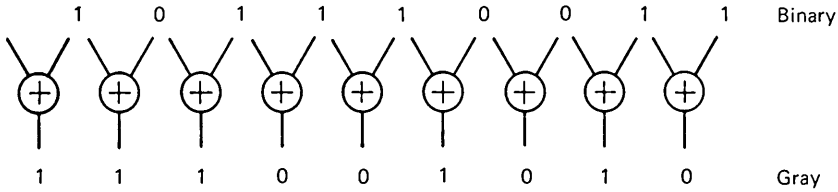


Figure 1.1 Conversion of binary code to Gray code.

of the binary code is added to the next more significant digit to form the corresponding Gray-code digits; the most-significant digit remains unchanged. The process is illustrated in figure 1.1.

It will be noted that the first and last code members of the Gray-code group shown in table 1.3 also differ in only one digit position. Cyclic codes with this characteristic are said to be *closed. A complete* cyclic code is one that utilizes all possible combinations of the digit values.

References

Chu, Y. 1962. *Digital computer design fundamentals.* New York: McGraw-Hill.

Gardner, M. 1968. Counting systems and the relationship between numbers and the real world. *Sci. Amer.* 219:218–230.

Miller, J. E., ed. 1966. *Space navigation guidance and control.* Maidenhead, England: Technivision.

Peterson, W. W., and Weldon, E. J., Jr. 1972. *Error-correcting codes.* 2d ed. Cambridge, Mass.: MIT Press.

Richards, R. K. 1971. *Digital design*. New York: Wiley.

Problems

1.1

Convert the following binary numbers to both their octal and decimal equivalents:

a. 11010110

b. 01010101

c. 1101.0110

d. 0101.0101

1.2

Convert the following decimal numbers to both their binary and octal equivalents:

a. 137.8125

b. 13.28

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

1.3

Convert the following numbers to both their binary and decimal equivalents:

a. $(765.24)_8$

b. $(2120.12)_3$

1.4

Perform the following binary-arithmetic operations:

a. 011.01 + 101.10 + 001.11 + 100.01

b. 11011.101 - 1101.011 - 1001.101

1.5

Convert the number $(371)_{10}$ to its equivalent quinary and octal number by appropriate evaluations of the decimal polynomial.

1.6

Convert the number $(0.1101)_2$ to its equivalent decimal fraction using repeated multiplication by the binary equivalent of decimal ten.

1.7

Convert the octal number 563 to its equivalent quaternary number by—

a. quaternary evaluation of the octal polynomial.

b. repeated division by the base four.

c. inspection, forming first the equivalent binary number and then the quaternary number.

1.8

Convert the decimal number 0.625 to its equivalent octal number by octal evaluation of the decimal polynomial.

1.9

Perform the arithmetic operations shown below using binary addition only and assuming—

a. a 6-bit 2's complement number representation.

b. a 6-bit 1's complement number representation.

+ 22	- 22	- 22
- 8	+ 8	- 8

1.10

Subtract $(3/32)_{10}$ from $-(3/16)_{10}$ assuming—

a. a 6-bit 2's complement representation.

b. a 6-bit 1's complement representation.

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

1.11
Specify the conditions under which an end-around carry will be generated in 1's complement addition.

1.12
Derive an expression for the range of integral decimal numbers that can be represented by an n-bit binary representation in—
a. the 2's complement system.
b. the 1's complement system.

1.13
Repeat problem 1.12 for fractional decimal numbers.

1.14
Perform the following binary arithmetic operations:
a. 1110.1 X 101.0
b. 100110.001 ÷ 101.0

1.15
Show how the number $(35)_{10}$ would be divided by the number $(10)_{10}$ using nonrestoring division and 2's complement arithmetic.

1.16
Derive the binary-coded-decimal representations for the following weighted codes.
a. 7421
b. 86-41
c. 441-2
d. 87-4-2
e. 731-2
Which codes are self-complementing?

1.17
Form the diminished-radix complement representation of each of the following numbers:
a. $(1101.01)_2$
b. $(1201.10)_3$
c. $(0.5643)_7$
d. $(98.257)_{10}$

1.18
Derive a four-digit closed cyclic code which has ten code groups. What restraint exists in the number of code groups in a closed cyclic code?

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024

1.19

Construct addition and subtraction tables for a radix-3 number system.

1.20

Can you derive an algorithm for converting Gray code to the ordinary binary code?

Downloaded from http://direct.mit.edu/books/monograph/4830/bookpreview-pdf/2398415 by guest on 14 November 2024