

Second Edition

**Computer  
System  
Architecture**

**M. MORRIS MANO**

Second Edition

**Computer  
System  
Architecture**

---

**M. MORRIS MANO**

*Professor of Engineering  
California State University, Los Angeles*

**PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632**

*Library of Congress Cataloging in Publication Data*

MANO, M. MORRIS.  
Computer system architecture.

Includes bibliographies and index.

1. Computer architecture. I. Title.

QA76.9.A73M36 1982 621.3819'52 81-15799  
ISBN 0-13-166611-8 AACR2

Editorial/Production Supervision: Nancy Moskowitz  
Manufacturing Buyer: Joyce Levatino  
Cover Design by Mario Piazza  
Art Production by Margaret Mary Finnerty and Steven Frim

©1982 by Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632

All rights reserved. No part of this book  
may be reproduced in any form or  
by any means without permission in writing  
from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3

ISBN 0-13-166611-8

PRENTICE-HALL INTERNATIONAL, INC., *London*  
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*  
PRENTICE-HALL OF CANADA, LTD., *Toronto*  
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*  
PRENTICE-HALL OF JAPAN, INC., *Tokyo*  
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*  
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

---

# Preface

---

Computer architecture is of concern to computer engineers who deal with the hardware design of computer systems, and computer scientists involved in the design of hardware dependent software systems. A computer system is a system that includes both hardware and software. This book is concerned mostly with the hardware aspects of computer systems, but the impact of software on the architecture of the computer has not been neglected.

Computer architecture is sometimes defined to include only those attributes of the computer that are of interest to the programmer. Here, we define computer architecture by considering what professional computer architects are supposed to know. Computer architects must be familiar with the basic hardware building blocks from which computers are constructed. They must have knowledge of the structure and behavior of computer systems and the way they are designed. Thus, computer architecture as defined in this book is concerned with the structural organization and hardware design of digital computer systems.

The physical organization of a particular processor including its registers, the data flow, the micro-operations and control functions are best described symbolically by means of a register transfer language. Such a language is developed in the book and its relation to the hardware organization and design of digital computers is fully explained. The register transfer language is used on many occasions to specify various computer operations in a concise and precise manner.

The plan of the book is to present the simpler material first and introduce the more advanced subjects later. The first six chapters cover material needed for the basic understanding of computer organization, design, and programming of a simple digital computer. The last six chapters present the separate functional units of digital computers with an emphasis on more advanced topics not covered in the earlier part.

The revisions for the second edition are in the last six chapters. Chapter 7, on the central processor unit and Chapter 11, on input-output organization have been completely rewritten. Chapter 8, on microprogram control and Chapter 12, on memory organization have been revised and new material added. The other chapters remain essentially the same as the first edition except for some minor reorganization.

Chapter 1 introduces the fundamental knowledge needed for the design of digital systems when they are constructed with individual gates and flip-flops. It covers Boolean algebra, combinational circuits, and sequential circuits. In order to keep the book within reasonable bounds, it is necessary to limit the discussion of this subject to one introductory chapter. The justification for this approach is that the design of digital computers takes on a different dimension when integrated circuit functions are employed instead of individual gates and flip-flops. The material included in the first chapter provides the necessary background for understanding the digital systems being presented.

Chapter 2 starts by enumerating the general properties of integrated circuits. It covers in detail some of the most basic digital functions such as registers, counters, decoders, multiplexers, random access memories, and read-only memories. The digital functions are used as building blocks for the design of larger units in the chapters that follow.

Chapter 3 presents the various data types found in digital computers and shows how they are represented in binary form in computer registers. The emphasis is on the representation of numbers employed in arithmetic computations and on the binary coding of symbols, such as the letters of the alphabet used in data processing, and other discrete symbols used for specific applications.

Chapter 4 defines a register transfer language and shows how it is used to express in symbolic form the micro-operation sequences among the registers of a digital computer. Symbols are defined for arithmetic, logic, and shift micro-operations as well as for control functions that initiate the micro-operations. The presentation goes to great lengths to show the hardware implications associated with the various symbols and register transfer statements.

Chapter 5 presents the organization and design of a small basic digital computer. The registers of the computer are defined and a set of computer instructions is specified. The computer description is formalized with register transfer statements that specify the micro-operations among the registers as well as the control functions that initiate the micro-operations. It is then shown that the set of micro-operations can be used to design the data processor part of the computer. The control functions in the list of register transfer statements supply the information for the design of the control unit.

Chapter 6 utilizes the twenty-five instructions of the basic computer defined in Chapter 5 to illustrate many of the techniques commonly used to program a computer. Programming examples in symbolic code are presented for many elementary data processing tasks. The relationships between binary programs, symbolic code programs, and high-level language programs are explained by examples. This leads into the necessity for translation programs such as assemblers and compilers. The basic operation of an assembler is presented together with other system programs. The purpose of this chapter is to introduce the basic ideas of computer software without going deeply into detail. Knowledge of software principles coupled with the hardware presentation should give the reader an overview of a total computer system which includes both hardware and software.

Chapter 7 deals with the central processor unit (CPU) of digital computers. A bus organized processor is presented and a specific arithmetic logic unit (ALU) is designed. The organization of a memory stack is explained with a demonstration of some of its applications. Various instruction formats are illustrated together with their addressing modes. The most common instructions found in a typical computer are enumerated with an explanation of their functions. The microprocessor which is a CPU enclosed in one integrated circuit package is then introduced and its internal and external characteristics analyzed. The chapter concludes with a section on parallel and pipeline processing.

Chapter 8 introduces the concept of microprogramming. A specific control unit is developed to show by example how to generate the microprogram for a typical set of computer instructions. A microprogram sequencer is developed to demonstrate the design procedure with LSI components of the bit-sliced variety. The last section discusses the advantages and applications of microprogramming.

Chapter 9 is devoted to the design of an arithmetic processor. It presents the algorithms for addition, subtraction, multiplication, and division of binary integer numbers in signed-magnitude representation. The arithmetic processor is designed using the register transfer language. The configuration ties the arithmetic processor to the computer designed in Chapter 5. A binary calculator is defined and used for demonstrating the method by which the arithmetic operations can be microprogrammed.

Chapter 10 presents other arithmetic algorithms. Algorithms are developed for signed-2's complement binary data, for floating-point data, and decimal data. The algorithms are presented by means of flow charts that use the register transfer language to specify the sequence of micro-operations and control decisions required for the implementation of the algorithms.

Chapter 11 explains the function of some commonly used input and output devices. The requirement of an interface between the processor and I/O devices is explained and the various configurations for I/O transfers are enumerated. This includes asynchronous transfer, direct memory access, and priority interrupt. Other topics covered are input-output processors, data communication processors, and multiprocessor system organization.

Chapter 12 introduces the concept of memory hierarchy, composed of cache memory, main memory, and auxiliary memory devices such as magnetic disks and tapes. The internal organization and external operation of associative memories is explained in detail. The concept of memory management is introduced through the presentation of the hardware requirements for a cache memory and a virtual memory system.

Every chapter includes a set of problems and a list of references. Some of the problems serve as exercises for the material covered in the chapter. Others are of a more advanced nature and are intended to provide some practice in solving problems associated with the area of digital computer hardware design. A *Solutions Manual* is available for the instructor from the publisher.

The book is suitable for a course in computer architecture in an electrical

engineering, computer engineering, or computer science department. Parts of the book can be used in a variety of ways: (1) As a first course in computer hardware organization by covering Chapters 1 through 5 with additional material from Chapters 7, 8, or 9 as the instructor sees fit. (2) As a course in computer design with previous knowledge of digital logic design by reviewing Chapter 5 and then covering Chapters 7 through 12. (3) As a course in computer hardware systems that covers the five functional units of digital computers: processor (Chapter 7), control (Chapter 8), arithmetic (Chapter 10), input-output (Chapter 11), and memory (Chapter 12). The book is also suitable for self-study by computer engineers and scientists who need to acquire the basic knowledge of computer hardware architecture.

**M. MORRIS MANO**

---

# Contents

---

<b>PREFACE</b>	<b>ix</b>
<b>1 DIGITAL LOGIC CIRCUITS</b>	<b>1</b>
1-1 Logic Gates	1
1-2 Boolean Algebra	5
1-3 Map Simplification	8
1-4 Combinational Circuits	15
1-5 Flip-Flops	21
1-6 Sequential Circuits	26
1-7 Concluding Remarks	35
References	35
Problems	36
<b>2 INTEGRATED CIRCUITS AND DIGITAL FUNCTIONS</b>	<b>39</b>
2-1 Digital Integrated Circuits	39
2-2 IC Flip-Flops and Registers	45
2-3 Decoders and Multiplexers	50
2-4 Binary Counters	54
2-5 Shift Registers	59
2-6 Random-Access Memories (RAM)	62
2-7 Read-Only Memories (ROM)	69
References	72
Problems	72



<b>3</b>	<b>DATA REPRESENTATION</b>	<b>75</b>
3-1	Data Types	75
3-2	Fixed-Point Representation	82
3-3	Floating-Point Representation	89
3-4	Other Binary Codes	92
3-5	Error Detection Codes	95
	References	97
	Problems	97
<b>4</b>	<b>REGISTER TRANSFER AND MICRO-OPERATIONS</b>	<b>101</b>
4-1	Register Transfer Language	101
4-2	Inter-Register Transfer	102
4-3	Arithmetic Micro-Operations	113
4-4	Logic Micro-Operations	118
4-5	Shift Micro-Operations	126
4-6	Control Functions	128
	References	133
	Problems	133
<b>5</b>	<b>BASIC COMPUTER ORGANIZATION AND DESIGN</b>	<b>137</b>
5-1	Instruction Codes	137
5-2	Computer Instructions	140
5-3	Timing and Control	145
5-4	Execution of Instructions	149
5-5	Input-Output and Interrupt	156
5-6	Design of Computer	161
5-7	Concluding Remarks	166
	References	167
	Problems	168
<b>6</b>	<b>COMPUTER SOFTWARE</b>	<b>172</b>
6-1	Introduction	172
6-2	Programming Languages	174
6-3	Assembly Language	177

6-4	The Assembler	182
6-5	Program Loops	188
6-6	Programming Arithmetic and Logic Operations	190
6-7	Subroutines	196
6-8	Input-Output Programming	201
6-9	System Software	206
	References	212
	Problems	213

## **7      CENTRAL PROCESSOR ORGANIZATION      217**

7-1	Processor Bus Organization	217
7-2	Arithmetic Logic Unit (ALU)	219
7-3	Stack Organization	228
7-4	Instruction Formats	236
7-5	Addressing Modes	241
7-6	Data Transfer and Manipulation	248
7-7	Program Control	254
7-8	Microprocessor Organization	264
7-9	Parallel Processing	273
	References	284
	Problems	284

## **8      MICROPROGRAM CONTROL ORGANIZATION      290**

8-1	Control Memory	290
8-2	Address Sequencing	292
8-3	Microprogram Example	298
8-4	Microprogram Sequencer	308
8-5	Microinstruction Formats	312
8-6	Software Aids	318
8-7	Advantages and Applications	320
	References	323
	Problems	323

## **9      ARITHMETIC PROCESSOR DESIGN      328**

9-1	Introduction	328
9-2	Comparison and Subtraction of Unsigned Binary Numbers	329

9-3	Addition and Subtraction Algorithm	334
9-4	Multiplication Algorithm	338
9-5	Division Algorithm	341
9-6	Processor Configuration	346
9-7	Design of Control	350
9-8	Microprogrammed Calculator	352
	References	361
	Problems	361
<b>10</b>	<b>ARITHMETIC ALGORITHMS</b>	<b>364</b>
10-1	Introduction	364
10-2	Arithmetic with Signed-2's Complement Numbers	366
10-3	Multiplication and Division	369
10-4	Floating-Point Arithmetic Operations	377
10-5	Decimal Arithmetic Unit	387
10-6	Decimal Arithmetic Operations	391
	References	398
	Problems	398
<b>11</b>	<b>INPUT-OUTPUT ORGANIZATION</b>	<b>403</b>
11-1	Peripheral Devices	403
11-2	I/O Interface	406
11-3	Asynchronous Data Transfer	415
11-4	Direct Memory Access (DMA)	428
11-5	Priority Interrupt	434
11-6	Input-Output Processor (IOP)	443
11-7	Multiprocessor System Organization	454
11-8	Data Communication Processor	462
	References	473
	Problems	474
<b>12</b>	<b>MEMORY ORGANIZATION</b>	<b>478</b>
12-1	Auxiliary Memory	478
12-2	Microcomputer Memory	482
12-3	Memory Hierarchy	487

12-4	Associative Memory	489
12-5	Virtual Memory	495
12-6	Cache Memory	501
12-7	Memory Management Hardware	509
	References	518
	Problems	518

---

---

# Digital Logic Circuits

---

---

## 1-1 LOGIC GATES

A digital computer, as the name implies, is a digital system that performs various computational tasks. The word *digital* implies that the information in the computer is represented by variables that take a limited number of *discrete* or *quantized* values. These values are processed internally by components that can maintain a limited number of discrete states. The decimal digits 0, 1, 2, . . . , 9, for example, provide 10 discrete values. In practice, digital computers function more reliably if only two states are used. Because of the physical restriction of components, and because human logic tends to be binary (i.e., *true* or *false*, *yes* or *no* statements), digital components that are constrained to take discrete values are further constrained to take only two values and are said to be *binary*.

Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a *bit*. Information is represented in digital computers in groups of bits. By using various coding techniques groups of bits can be made to represent not only binary numbers but also any other discrete symbols, such as decimal digits or letters of the alphabet. By judicious use of binary arrangements and by using various coding techniques, the binary digits or groups of bits may be used to develop complete sets of instructions for performing various types of computations.

In contrast to common decimal numbers that employ the base 10 system, binary numbers use a base 2 system. For example, the binary number 101101 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows:

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

The six bits 101101 represent a binary number whose decimal equivalent is 45. However, the group of six bits could also represent a binary code for a letter of the alpha-

bet or a control code for specifying some decision logic in a particular digital system. In other words, groups of bits in a digital computer are used to represent many different things. This is similar to the concept that the same letters of an alphabet are used to construct different languages, such as English and French.

Binary information is represented in a digital system by physical quantities called *signals*. Electrical signals such as voltages exist throughout a digital system in either one of two recognizable values and represent a binary variable equal to 1 or 0. For example, a particular digital system may employ a signal of 3 V to represent a binary 1 and 0.5 V for binary 0. As shown in Fig. 1-1, each binary value has an acceptable deviation from the nominal. The intermediate region between the two allowed regions is crossed only during state transition. The input terminals of digital circuits accept binary signals within the allowable tolerances and the circuits respond at the output terminals with binary signals that fall within the specified tolerances.

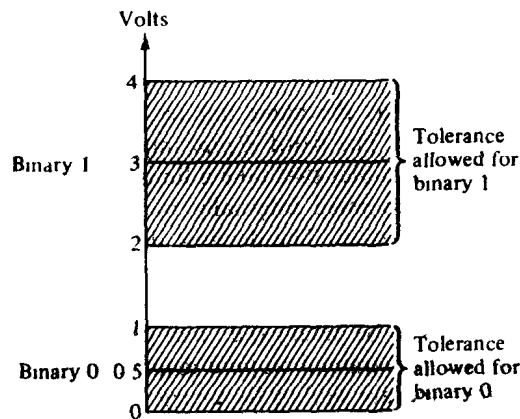


Fig. 1-1 Example of a binary signal.

*Binary logic* deals with binary variables and with operations that assume a logical meaning. It is used to describe, in algebraic or tabular form, the manipulation and processing of binary information. The manipulation of binary information is done by logic circuits called *gates*. Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied. Various logic gates are commonly found in digital computer systems. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic function. The input-output relationship of the binary variables for each gate can be represented in tabular form in a *truth table*.

The names, graphic symbols, algebraic functions, and truth tables of eight logic gates are listed in Fig. 1-2. Each gate has one or two binary input variables designated by  $A$  and  $B$  and one binary output variable designated by  $x$ . The AND gate produces the AND logic junction: that is, the output is 1 if input  $a$  *and* input  $B$  are *both* binary 1, otherwise, the output is 0. These conditions are also specified in the

truth table for the AND gate. The table shows that output  $x$  is 1 only when both input  $A$  and input  $B$  are 1. The algebraic operation symbol of the AND function is the same as the multiplication symbol of ordinary arithmetic. We can either use a dot between the variables or concatenate the variables without an operation symbol between them. AND gates may have more than two inputs and by definition, the output is 1 if and only if *all* inputs are 1.

The OR gate produces the inclusive-OR function, that is, the output is 1 if input  $A$  or input  $B$  or both inputs are 1; otherwise, the output is 0. The algebraic symbol of the OR function is  $+$ , similar to arithmetic addition. OR gates may have more than two inputs and by definition, the output is 1 if *any* input is 1.

The inverter circuit inverts the logic sense of a binary signal. It produces the NOT, or *complement*, function. The algebraic symbol used for the logic complement






Name	Graphic Symbol	Algebraic Function	Truth Table															
AND		$x = A \cdot B$ or $x = AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$x = A + B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
inverter		$x = A'$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																	
0	1																	
1	0																	
buffer		$x = A$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	x	0	0	1	1									
A	x																	
0	0																	
1	1																	
NAND		$x = (AB)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	1																
0	1	1																
1	0	1																
1	1	0																

Fig. 1-2 Digital logic gates.




Name	Graphic Symbol	Algebraic Function	Truth Table															
NOR		$x = (A + B)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
exclusive-NOR or equivalence		$x = A \odot B$ or $x = A'B' + AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Fig. 1-2 (Continued).

is either a prime or a bar over the variable symbol. This book uses a prime for the logic complement of a binary variable, while a bar over the letter is reserved for designating a complement micro-operation as defined in Chap. 4.

The small circle in the output of the graphic symbol of an inverter designates a logic complement. A triangle symbol by itself designates a buffer circuit. A buffer does not produce any particular logic function since the binary value of the output is the same as the binary value of the input. This circuit is used merely for signal amplification. For example, a buffer that uses 3 V for binary 1 will produce an output of 3 V when its input is 3 V. However, the current supplied at the input is much smaller than the current produced at the output. This way, a buffer can drive many other gates requiring a large amount of current not otherwise available from the small amount of current applied to the buffer input.

The NAND function is the complement of the AND function, as indicated by the graphic symbol which consists of an AND graphic symbol followed by a small circle. The designation NAND is derived from the abbreviation of NOT-AND. A more proper designation would have been AND-invert since it is the AND function that is inverted. The NOR gate is the complement of the OR gate and uses an OR graphic symbol followed by a small circle. Both NAND and NOR gates may have more than two inputs, and the output is always the complement of the AND or OR function, respectively.



The exclusive-OR gate has a graphic symbol similar to the OR gate except for the additional curved line on the input side. The output of this gate is 1 if any input is 1 but excludes the combination when both inputs are 1. The exclusive-OR function has its own algebraic symbol or can be expressed in terms of AND, OR, and complement operations as shown in Fig. 1-2. The exclusive-NOR is the complement of the exclusive-OR as indicated by the small circle in the graphic symbol. The output of this gate is 1 only if both inputs have the same binary value. We shall refer to the exclusive-NOR function as the *equivalence* function. Since the exclusive-OR and equivalence functions are not always the complement of each other. A more fitting name for the exclusive-OR operation would be an *odd* function; i.e., its output is 1 if an odd number of inputs are 1. Thus, in a three-input exclusive-OR (odd) function, the output is 1 if only one input is 1 or if all three inputs are 1. The equivalence function is an *even* function; that is, its output is 1 if an even number of inputs are 0. For a three-input equivalence function, the output is 1 if none of the inputs are 0 (all inputs are 1) or if two of its inputs are 0 (one input is 1). Careful investigation will show that the exclusive-OR and equivalence functions are the complement of each other when the gates have an even number of inputs, but the two functions are equal when the number of inputs is odd. These two gates are commonly available with two inputs and only seldom are they found with three or more inputs.

## 1-2 BOOLEAN ALGEBRA

Boolean algebra is an algebra that deals with binary variables and logic operations. The variables are designated by letters such as  $A$ ,  $B$ ,  $x$ , and  $y$ . The three basic logic operations are *AND*, *OR*, and *complement*. A Boolean function is an algebraic expression formed with binary variables, the logic operation symbols, parentheses, and equal sign. For a given value of the variables, the Boolean function can be either 1 or 0. Consider, for example, the Boolean function

$$F = x + y'z$$

The function  $F$  is equal to 1 if  $x$  is 1 or if both  $y'$  and  $z$  are equal to 1;  $F$  is equal to 0 otherwise. But saying that  $y' = 1$  is equivalent to saying that  $y = 0$  since  $y'$  is the complement of  $y$ . Equivalently, we may say that  $F$  is equal to 1 if  $x = 1$  or  $yz = 01$ . The relationship between a function and its binary variables can be represented in a truth table. To represent a function in a truth table we need a list of the  $2^n$  combinations with 1's and 0's of the  $n$  binary variables. As shown in Fig. 1-3(a), there are eight possible distinct combinations for assigning bits to the three variables. The function  $F$  is equal to 1 for those combinations where  $x = 1$  or  $yz = 01$ ; it is equal to 0 for all other combinations.

A Boolean function can be transformed from an algebraic expression into