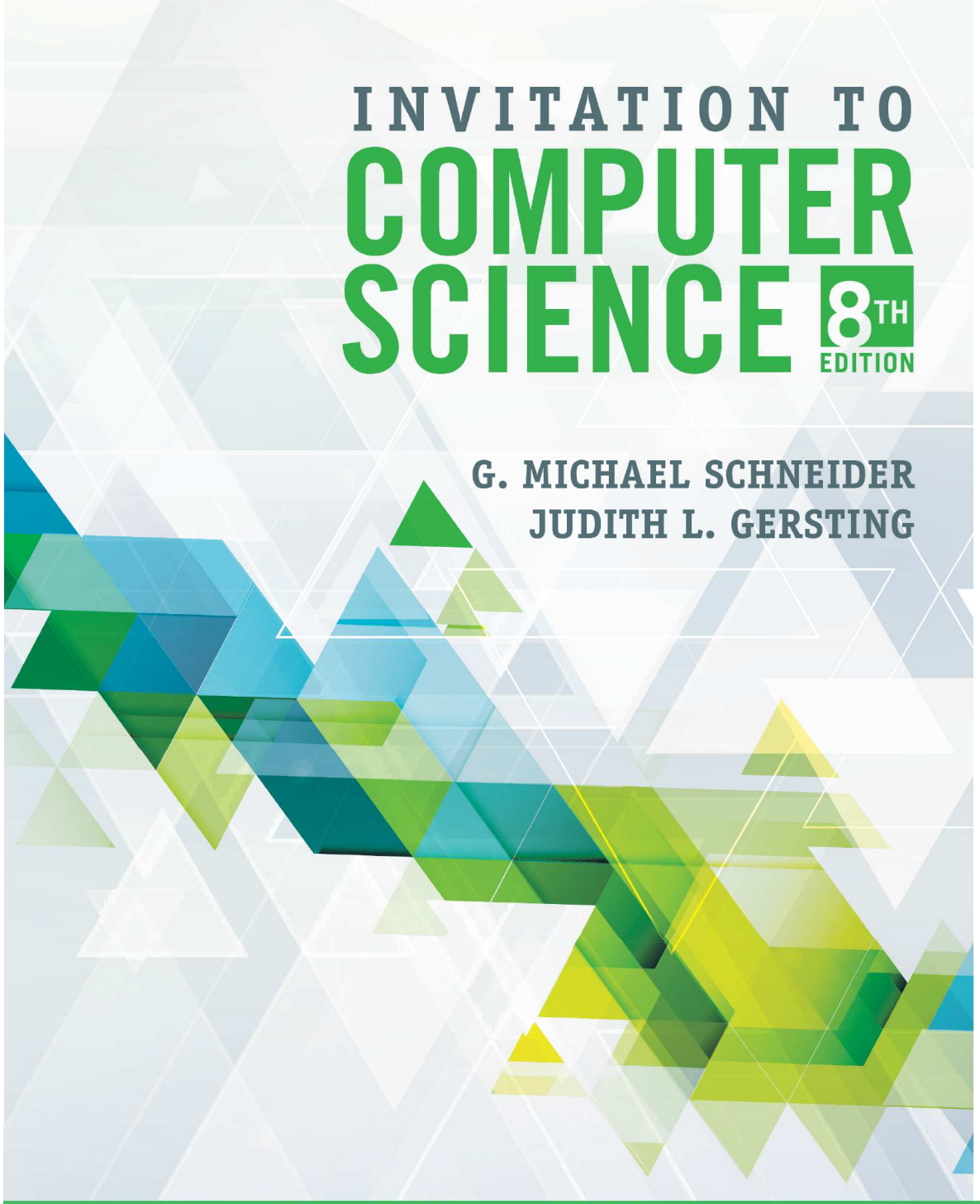


Chapter 4

The Building Blocks: Binary Numbers, Boolean Logic, and Gates

CENGAGE

Learning Objectives (1 of 2)

- Translate between base-ten and base-two numbers, and represent negative numbers using both sign-magnitude and two's complement representations
- Explain how fractional numbers, characters, sounds, and images are represented inside the computer
- Build truth tables for Boolean expressions and determine when they are true or false
- Describe the relationship between Boolean logic and electronic gates

Learning Objectives (2 of 2)

- Construct circuits using the sum-of-products circuit design algorithm, and analyze simple circuits to determine their truth tables
- Explain how large, complex circuits, like 32-bit adder or compare-for-equality circuits, are constructed from simpler, 1-bit components
- Describe the purpose and workings of multiplexer and decoder control circuits

Introduction

- This chapter introduces the fundamental building blocks of all computer systems
  - Binary representation
  - Boolean logic
  - Gates
  - Circuits
- This chapter examines what the computing agent looks like and how it is able to execute instructions and produce results

The Binary Numbering System (1 of 21)

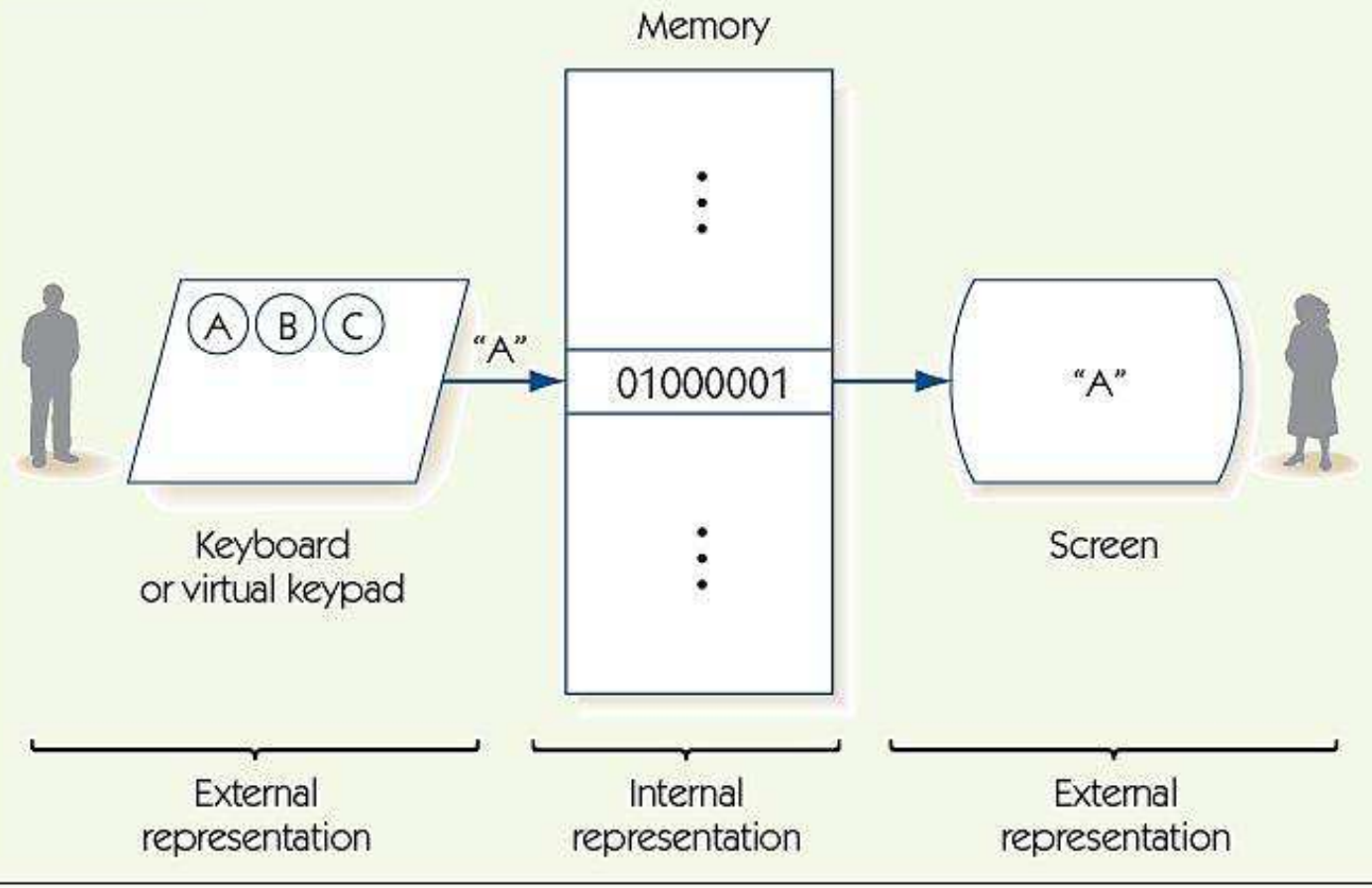
- How can an electronic (or magnetic) machine represent information?
- Key requirements: clear, unambiguous, and reliable
- External representation is human-oriented
  - Base-10 numbers
  - Keyboard characters
- Internal representation is computer-oriented
  - Base-2 numbers
  - Base-2 codes for characters

The Binary Numbering System (2 of 21)

- Binary is the simple idea of On/Off, Yes/No, True/False, and Positive/Negative.
- Binary is important to computing systems because of it's stability and reliability.
  - Even when an electrical system degrades, there is still a clear "On/Off."
- All data stored inside a computer is stored in binary (also called machine language) and interpreted to display on the screen in human language.

The Binary Numbering System (3 of 21)

FIGURE 4.1



Distinction between external and internal representation of information

The Binary Numbering System (4 of 21)

- The **binary numbering system** is a base-2 **positional numbering system**
- Base ten:
  - Uses 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - Each place corresponds to a power of 10
  - $1{,}943 = (1 \times 10^3) + (9 \times 10^2) + (4 \times 10^1) + (3 \times 10^0)$
- Base two:
  - Uses 2 digits: 0 and 1
  - Each place corresponds to a power of 2
  - $1101 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13$

The Binary Numbering System (5 of 21)

FIGURE 4.2 Binary-to-decimal conversion table

Binary	Decimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Binary	Decimal
10000	16
10001	17
10010	18
10011	19
10100	20
10101	21
10110	22
10111	23
11000	24
11001	25
11010	26
11011	27
11100	28
11101	29
11110	30
11111	31

The Binary Numbering System (6 of 21)

- Converting from binary to decimal
  - Add up powers of two where a 1 appears in the binary number
- Converting from decimal to binary
  - Repeatedly divide by two and record the remainder
  - Example, convert 11:
    - 11 ÷ 2 = 5, remainder = 1, binary number = 1
    - 5 ÷ 2 = 2, remainder = 1, binary number = 11
    - 2 ÷ 2 = 1, remainder = 0, binary number = 011
    - 1 ÷ 2 = 0, remainder = 1, binary number = 1011

The Binary Numbering System (7 of 21)

- Computers use fixed-length binary numbers for integers, e.g., 4 bits could represent 0 to 15
- **Arithmetic overflow**: when the computer tries to make a number that is too large, e.g., 14 + 2 with 4 bits
- Binary addition: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1,

  1 + 1 = 10 (that is, 0 with a carry of 1)
- Example: 0101 + 0011 = 1000

The Binary Numbering System (8 of 21)

- Signed integers include negative numbers
- **Sign/magnitude notation** uses 1 bit for the sign and the rest for the value

  +5 = 0101, -5 = 1101

  0 = 0000 and 1000!
- **Two's complement representation**: to make the negative of a number, flip every bit and add one

  +5 = 0101, -5 = 1010 + 1 = 1011

  0 = 0000, -0 = 1111 + 1 = 0000

The Binary Numbering System (9 of 21)

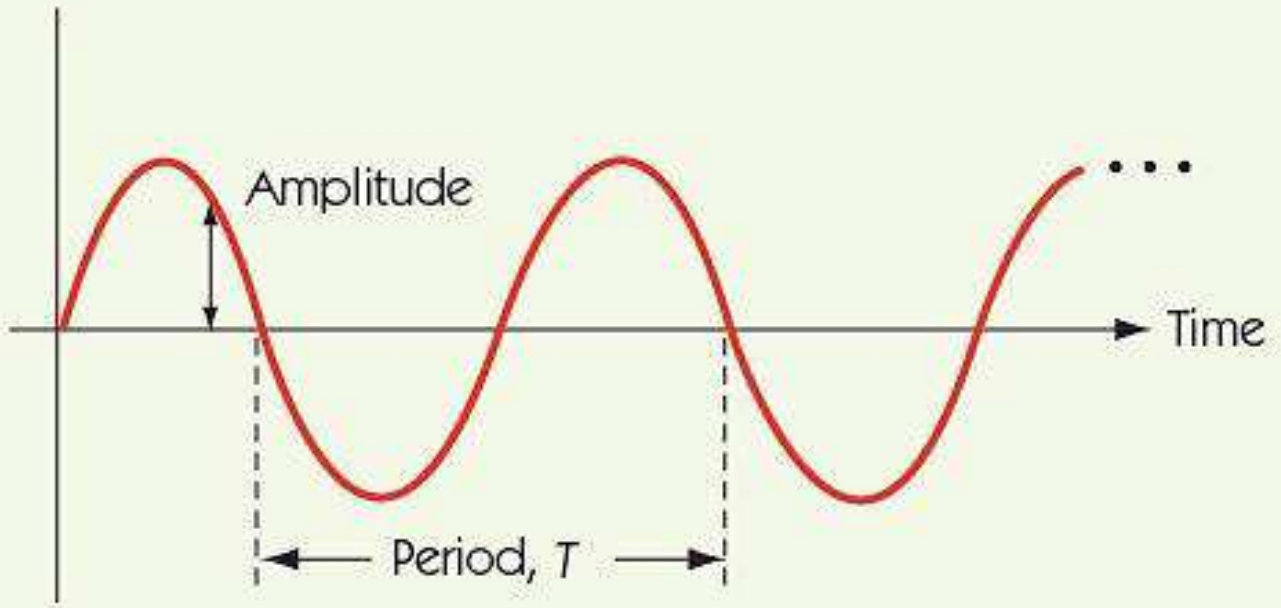
- Floating point numbers use binary scientific notation
  - **Scientific notation**, base 10: 1.35×10^{-5}
  - Base 2: $3.25_{10} = 11.01_2 = 1.101 \times 2^1$
- Characters and text: map characters onto binary numbers in a standard way
  - **ASCII** (8-bit numbers for each character)
  - **Unicode** (Minimum of 16-bit numbers for each character)

The Binary Numbering System (10 of 21)

- Sounds and images require converting naturally **analog representations** to **digital representations**
- Sound waves characterized by:
  - **Amplitude:** height of the wave at a moment in time
  - **Period:** length of time until wave pattern repeats
  - **Frequency:** number of cycles per unit time

The Binary Numbering System (11 of 21)

FIGURE 4.4



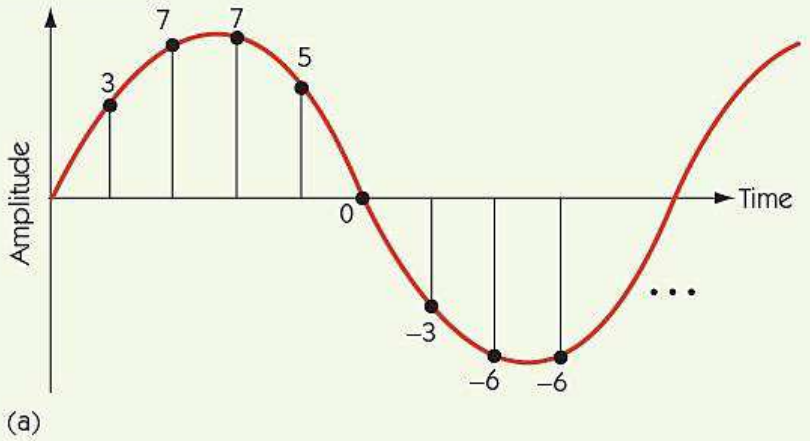
Example of sound represented as a waveform

The Binary Numbering System (12 of 21)

- **Digitize**: to convert to a digital form
- **Sampling**: record sound wave values at fixed, discrete intervals
- To reproduce sound, approximate using samples
- Quality is determined by
  - **Sampling rate:** number of samples per second
    - More samples ▸ more accurate waveform
  - **Bit depth:** number of bits per sample
    - More bits ▸ more accurate amplitude

The Binary Numbering System (13 of 21)

FIGURE 4.5



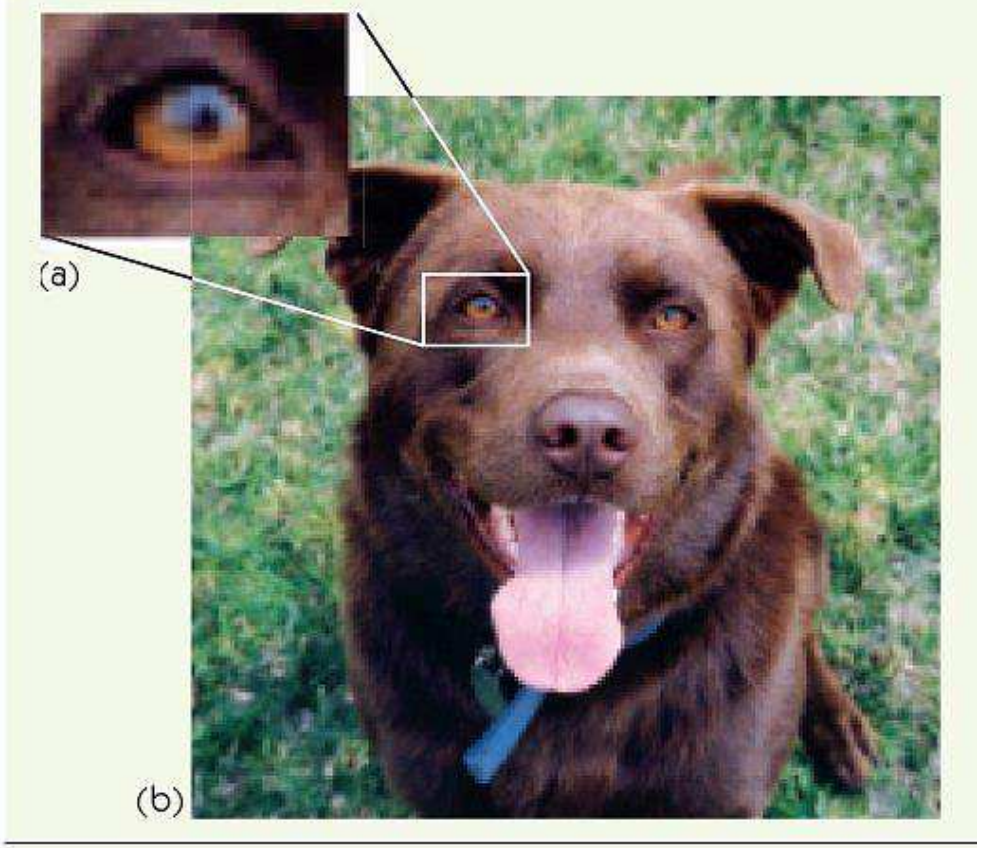
Digitization of an analog signal
(a) Sampling the original signal
(b) Recreating the signal from the sampled values

The Binary Numbering System (14 of 21)

- Image sampling: record color or intensity at fixed, discrete intervals in two dimensions
- Pixels: individual recorded samples
- **RGB encoding scheme**:
  - Colors are combinations of red, green, and blue
  - One byte each for red, green, and blue
- **Raster graphics** store picture as two dimensional grid of pixel values

The Binary Numbering System (15 of 21)

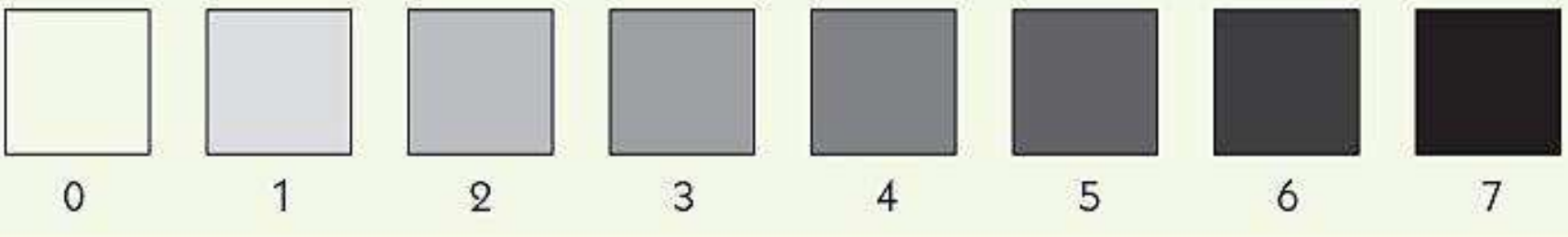
FIGURE 4.7



Example of a digitized photograph
(a) Individual pixels in the photograph
(b) Photograph

The Binary Numbering System (16 of 21)

FIGURE 4.8



An eight-level gray scale

The Binary Numbering System (17 of 21)

- What is the space necessary to store the following data?
  - 1000 integer values
  - 10-page text paper
  - 60-second sound file
  - 480 by 640 image
- **Data compression**: storing data in a reduced-size form to save space/time
  - Lossless: data can be perfectly restored
  - Lossy: data cannot be perfectly restored

The Binary Numbering System (18 of 21)

Letter	4-bit Encoding	Variable Length Encoding
A	0000	00
I	0001	10
H	0010	010
W	0011	110
E	0100	0110
O	0101	0111
M	0110	11100
K	0111	11101
U	1000	11110
N	1001	111110
P	1010	1111110
L	1011	1111111
	(a)	(b)

Using variable-length code sets

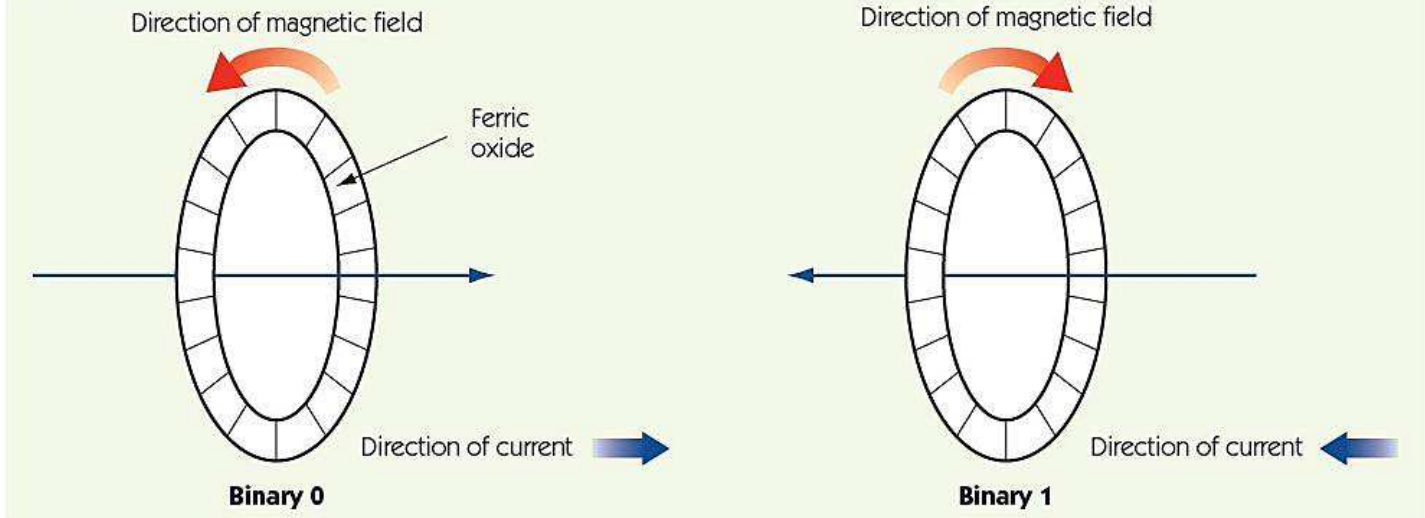
(a) Fixed Length

(b) Variable Length

The Binary Numbering System (19 of 21)

- Computers use binary because “bistable” systems are reliable
  - Current on/off
  - Magnetic field left/right

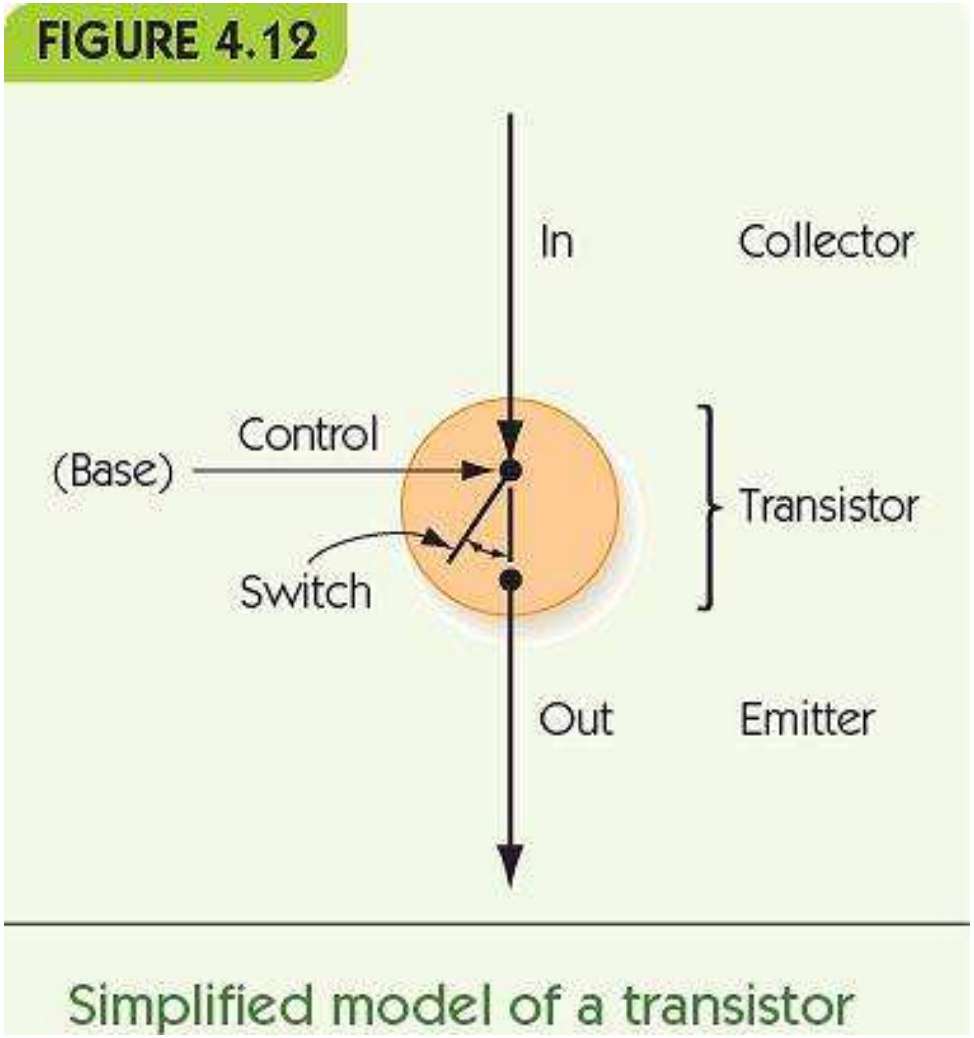
FIGURE 4.10



Using magnetic cores to represent binary values

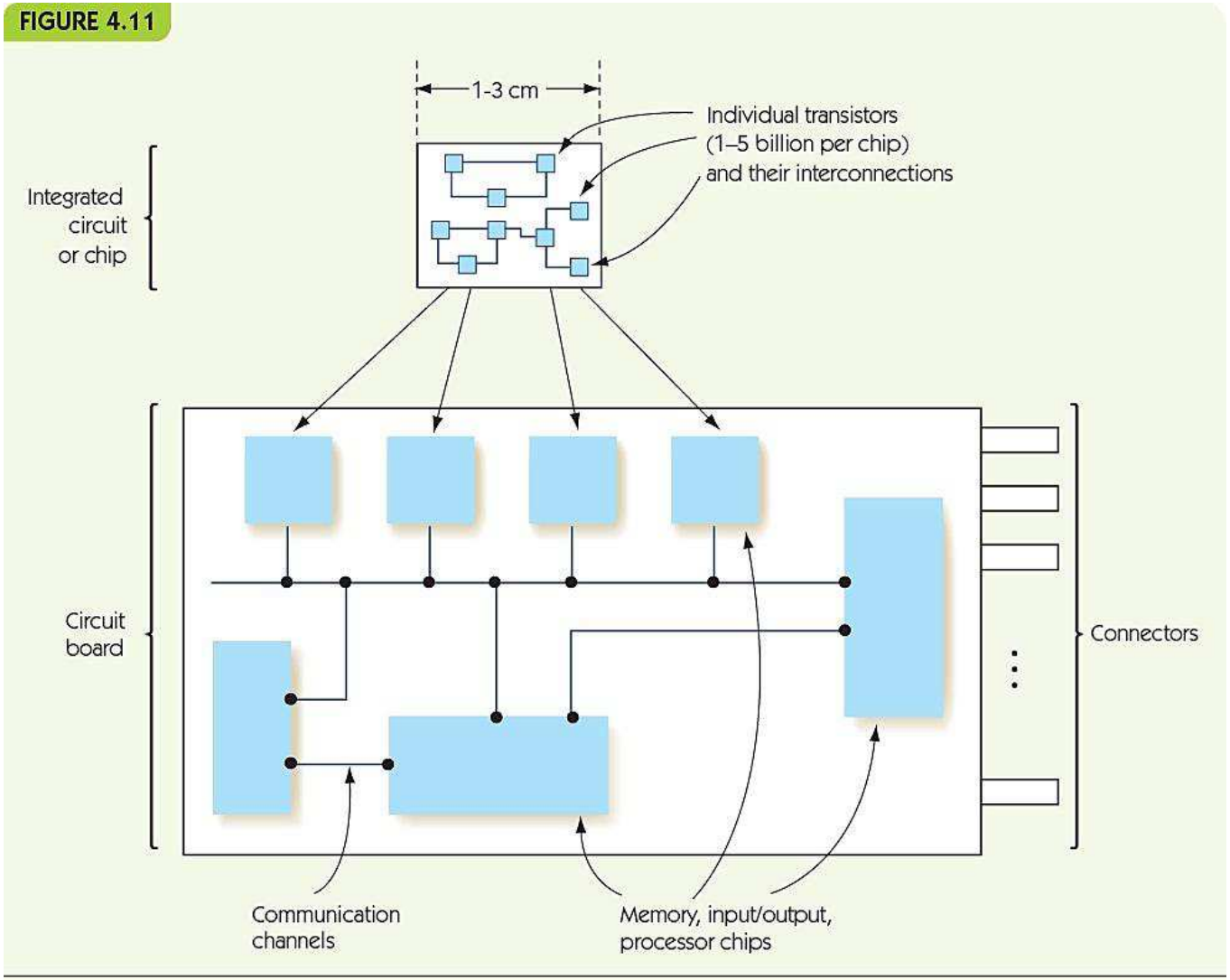
The Binary Numbering System (20 of 21)

- **Transistors**
  - Solid-state switches
  - Change on/off when given power on control line
  - Extremely small (billions per chip)
  - Enable computers that work with **gigabytes** of data



The Binary Numbering System (21 of 21)

FIGURE 4.11



Relationships among transistors, chips, and circuit boards

Boolean Logic and Gates (1 of 9)

- **Boolean logic** is the rule for manipulating true/false expressions for binary machine language
- Boolean expressions can be converted to circuits
- **Hardware design/logic design** pertains to the design and construction of new circuits
- Binary 1/0 maps to true/false of Boolean logic
- Boolean expressions: $x \leq 35$, $a = 12$
- Boolean operators: $(0 \leq x)$ AND $(x \leq 35)$, $(a = 12)$ OR $(a = 13)$, NOT $(a = 12)$

  $(0 \leq x) \bullet (x \leq 35)$, $(a = 12) + (a = 13)$, $\sim(a = 12)$

Boolean Logic and Gates (2 of 9)

- **Truth tables** lay out true/false values for Boolean expressions, for each possible true/false input

Boolean Logic and Gates (3 of 9)

FIGURE 4.14

Inputs: a	Inputs: b	Output a AND b (also written a . b)
False	False	False
False	True	False
True	False	False
True	True	True

Truth table for the AND operation

Boolean Logic and Gates (4 of 9)

FIGURE 4.15

Inputs: a	Inputs: b	Output a OR b (also written a + b)
False	False	False
False	True	True
True	False	True
True	True	True

Truth table for the OR operation

Boolean Logic and Gates (5 of 9)

FIGURE 4.16

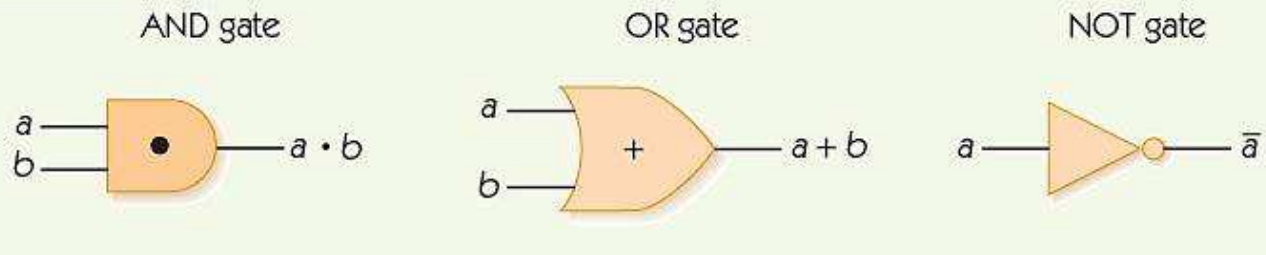
Inputs: a	**Output NOT a** (also written ā)
False	True
True	False

Truth table for the NOT operation

Boolean Logic and Gates (6 of 9)

- **Gate:** an electronic device that operates on inputs to produce outputs
- Each gate corresponds to a Boolean operator

FIGURE 4.17



a	b	a · b
0	0	0
0	1	0
1	0	0
1	1	1

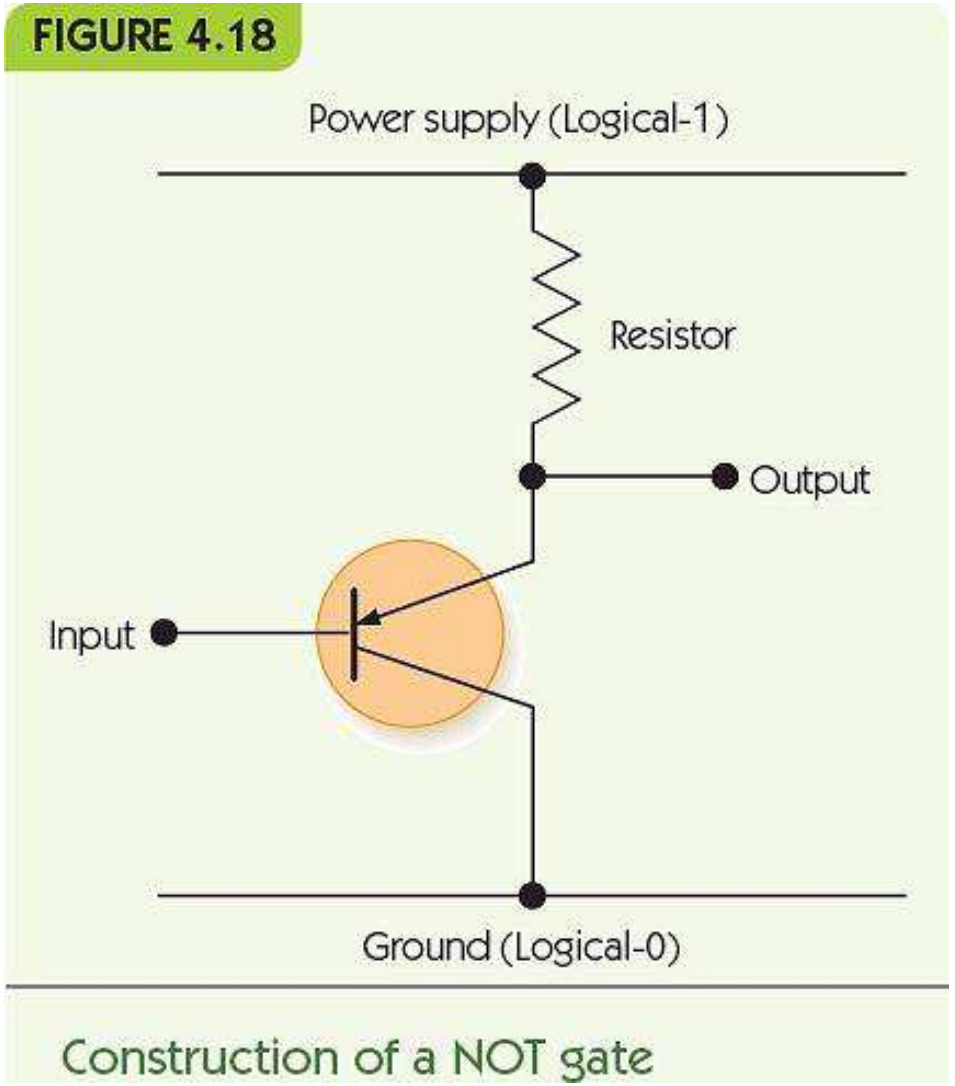
a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	1

a	ā
0	1
1	0

The three basic gates and their symbols

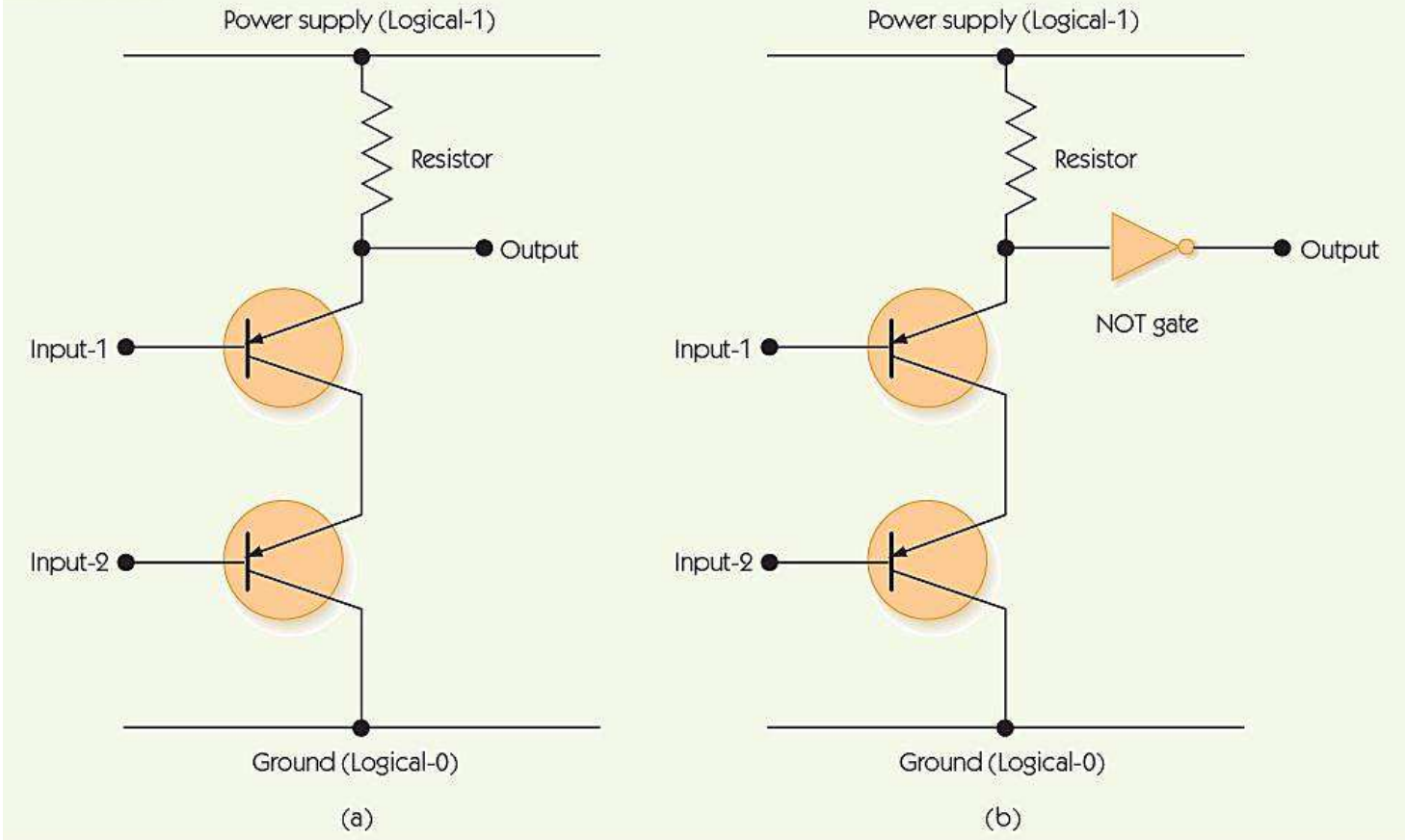
Boolean Logic and Gates (7 of 9)

- Gates are built from transistors
- NOT gate: 1 transistor
- AND gate: 3 transistors
- OR gate: 3 transistors
- NAND and NOR: 2 transistors
- Transistors can be in series or parallel



Boolean Logic and Gates (8 of 9)

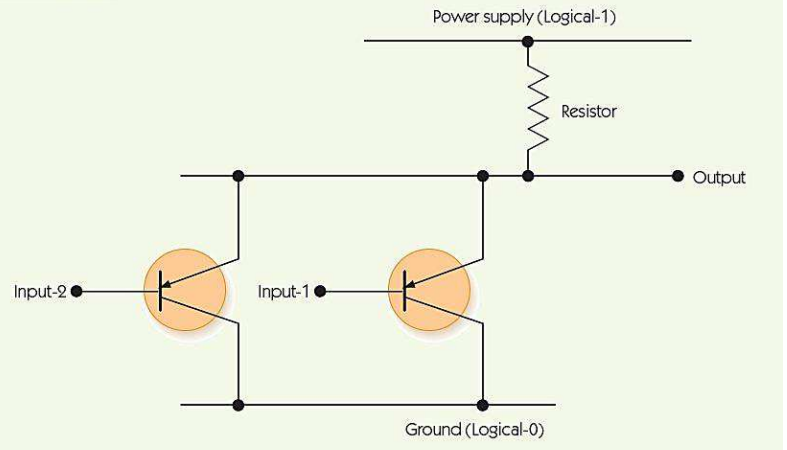
FIGURE 4.19

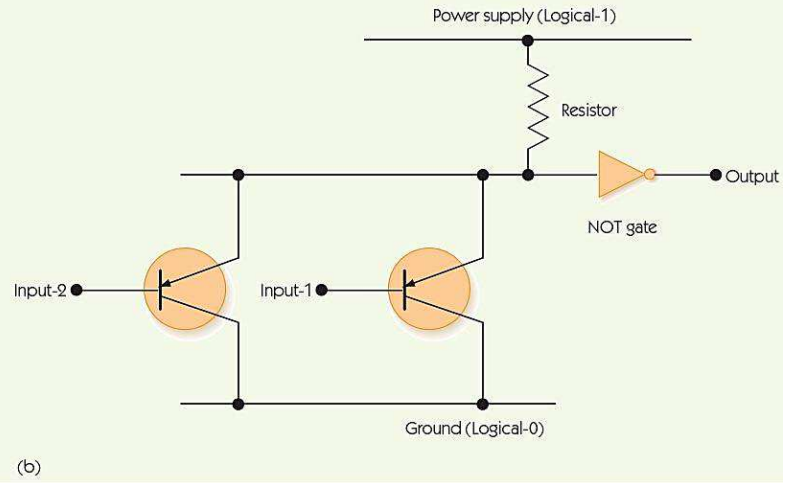


Construction of NAND and AND gates
(a) A two-transistor NAND gate
(b) A three-transistor AND gate

Boolean Logic and Gates (9 of 9)

FIGURE 4.20





Construction of NOR and OR gates
(a) A two-transistor NOR gate
(b) A three-transistor OR gate

Building Computer Circuits (1 of 18)

- **Circuit:** has input wires, contains gates connected by wires, and has output wires
- Outputs depend only on current inputs: no state

FIGURE 4.21

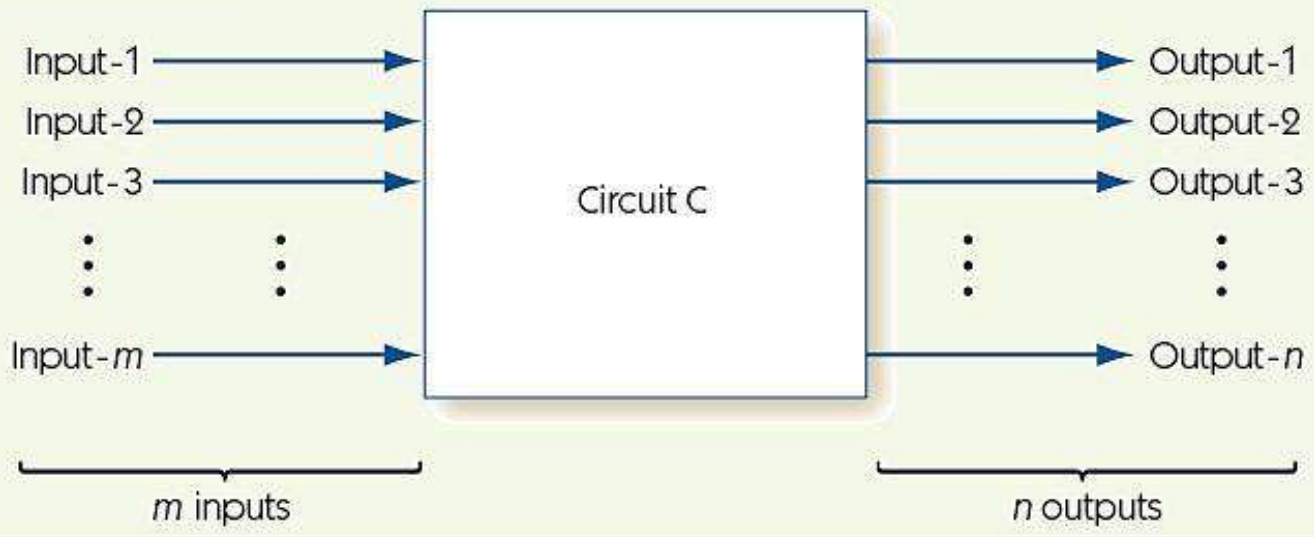


Diagram of a typical computer circuit

Building Computer Circuits (2 of 18)

- To convert a circuit to a Boolean expression
  - Start with output and work backward
    - Find next gate back, convert to Boolean operator
    - Repeat for each input, filling in left and/or right side
- To convert a Boolean expression to a circuit
  - Similar approach
- To build a circuit from desired outcomes
  - Use standard **circuit construction algorithm**
    - E.g., sum-of-products algorithm

Building Computer Circuits (3 of 18)

Example from text

- Build truth table

Inputs			Outputs	
a	*b*	*c*	*Output-1*	*Output-2*
0	0	0	0	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

$2^3 = 8$ input combinations

Building Computer Circuits (4 of 18)

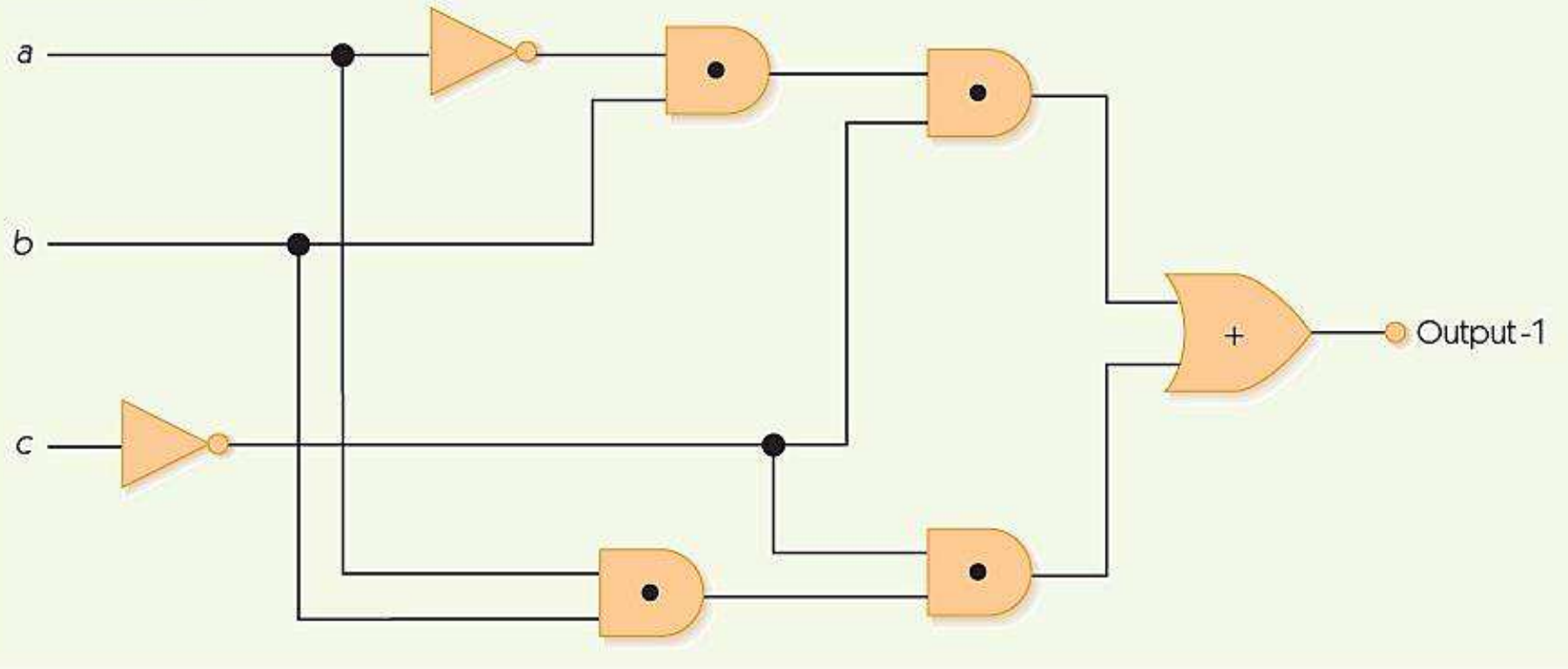
Example from text

- Find true rows for Output-1

Inputs				
a	*b*	*c*	*Output-1*	
0	0	0	0	
0	0	1	0	
0	1	0	1	← case 1
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	← case 2
1	1	1	0	

Building Computer Circuits (5 of 18)

FIGURE 4.24



Circuit diagram for the output labeled Output-1

Building Computer Circuits (6 of 18)

FIGURE 4.25

1. Construct the truth table describing the behavior of the desired circuit
2. While there is still an output column in the truth table, do Steps 3 through 6
3. Select an output column
4. Subexpression construction using AND and NOT gates
5. Subexpression combination using OR gates
6. Circuit diagram production
7. Done

The sum-of-products circuit construction algorithm

Building Computer Circuits (7 of 18)

Compare-for-equality (CE) circuit

- Input is two unsigned binary numbers.
- Output is 1 if inputs are identical and 0 otherwise.
- Start with 1-bit version (1-CE) and build general version from that.

Building Computer Circuits (8 of 18)

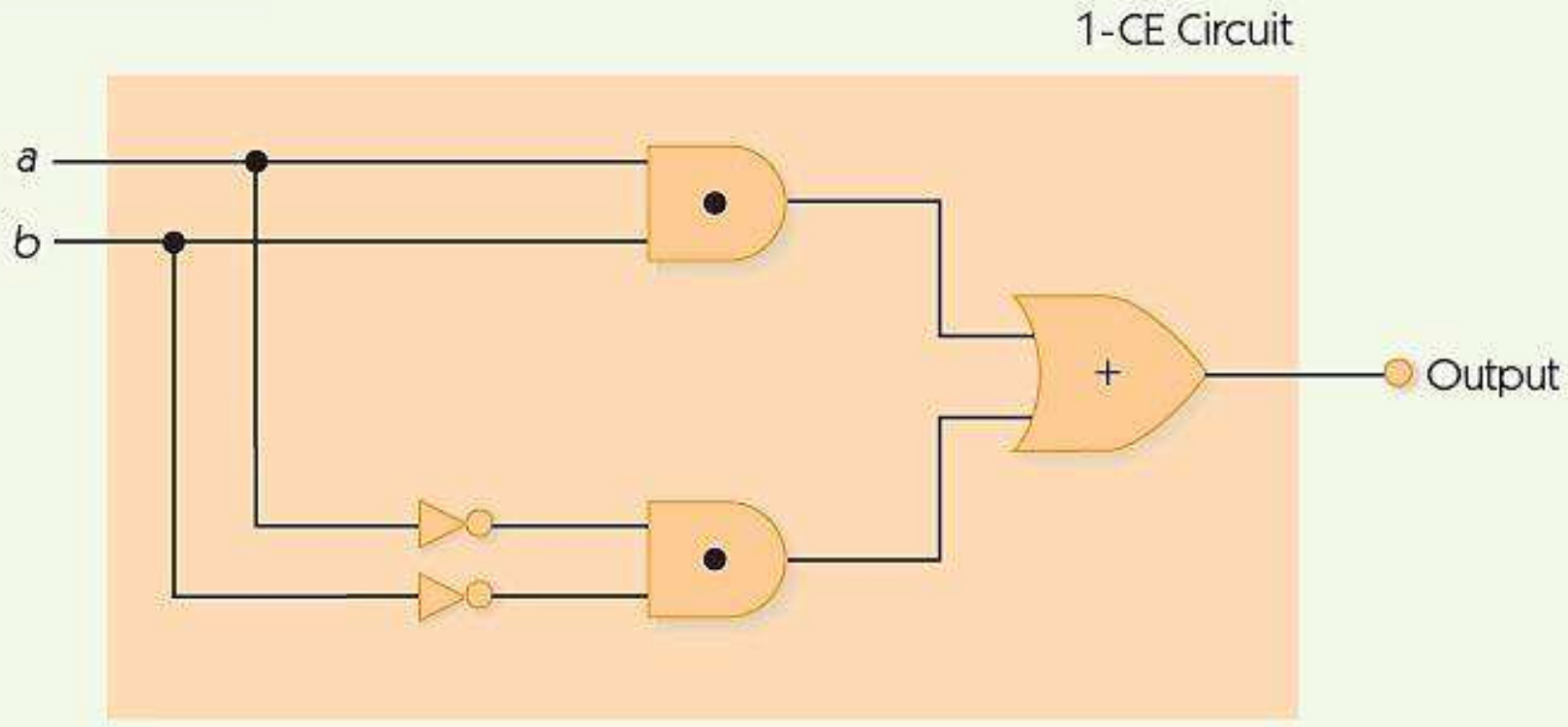
- 1-CE circuit: compare two input bits for equality
- Truth table

a	*b*	*Output*
0	0	1 ← case 1 (both numbers equal to 0)
0	1	0
1	0	0
1	1	1 ← case 2 (both numbers equal to 1)

- Boolean expression: (a • b) + (~a • ~b)

Building Computer Circuits (9 of 18)

FIGURE 4.27



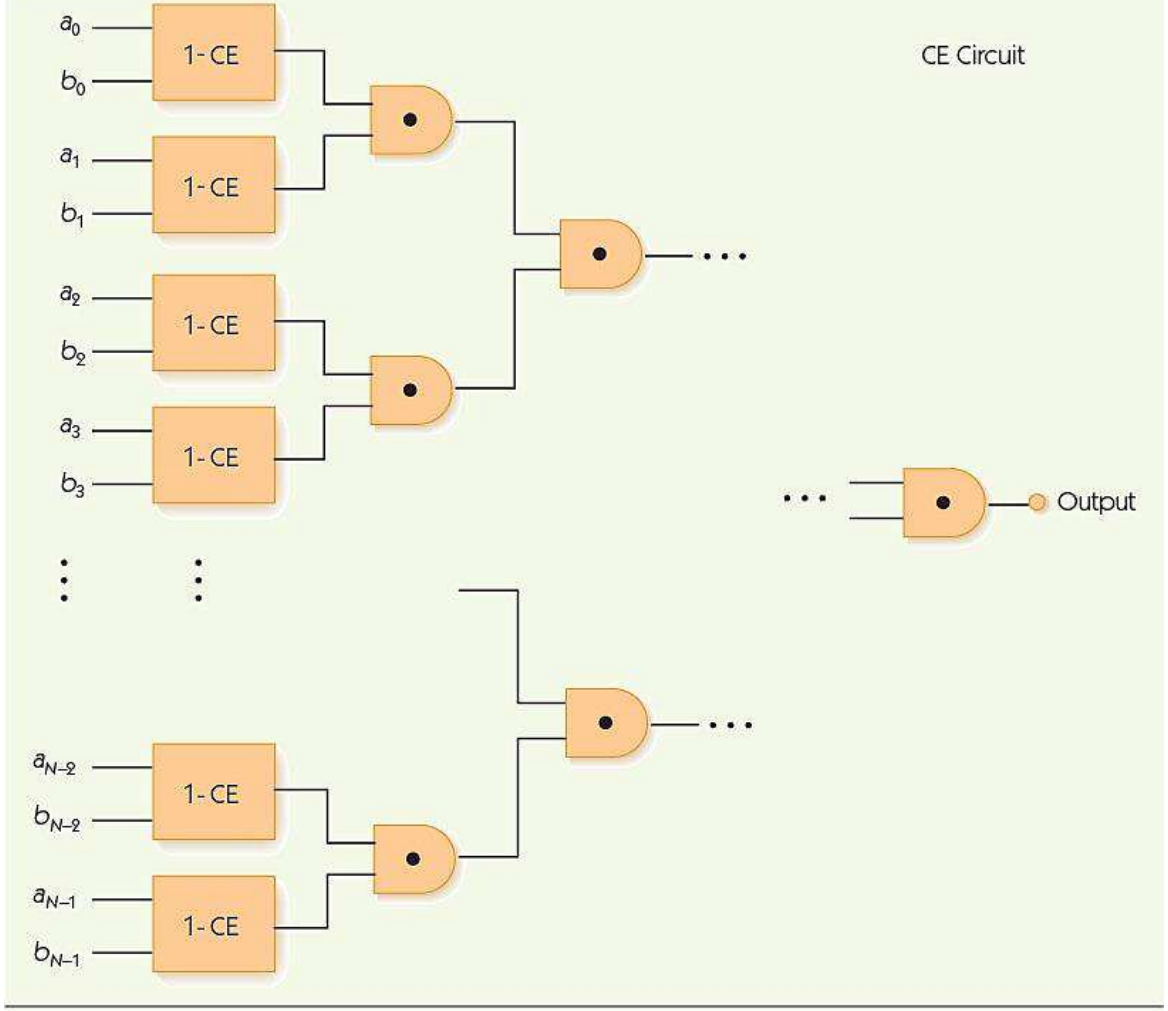
One-bit compare-for-equality circuit

Building Computer Circuits (10 of 18)

- N-bit CE circuit
- Input: $a_0a_2\ldots a_{n-1}$ and $b_0b_2\ldots b_{n-1}$, where a_i and b_i are individual bits
- Pair up corresponding bits: a_0 with b_0, a_1 with b_1, etc.
- Run a 1-CE circuit on each pair
- AND the results

Building Computer Circuits (11 of 18)

FIGURE 4.28



N-bit compare-for-equality circuit

Building Computer Circuits (12 of 18)

Full adder circuit

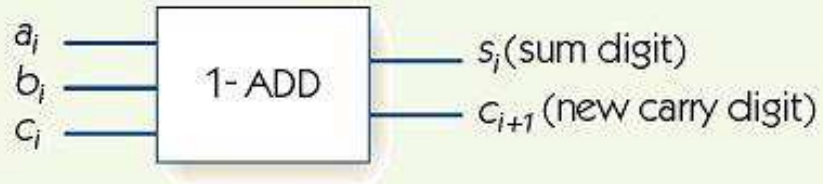
- Input is two unsigned N-bit numbers
- Output is one unsigned N-bit number, the result of adding inputs together
- Example

				1	
	0	0	1	0	1
+	0	1	0	0	1
	0	1	1	1	0

- Start with 1-bit adder (1-ADD)

Building Computer Circuits (13 of 18)

FIGURE 4.29



Inputs			Outputs	
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The 1-ADD circuit and truth table

Building Computer Circuits (14 of 18)

- Sum digit, s_i, has the Boolean expression

  $(\sim a_i \bullet \sim b_i \bullet c_i) + (\sim a_i \bullet b_i \bullet \sim c_i) +$

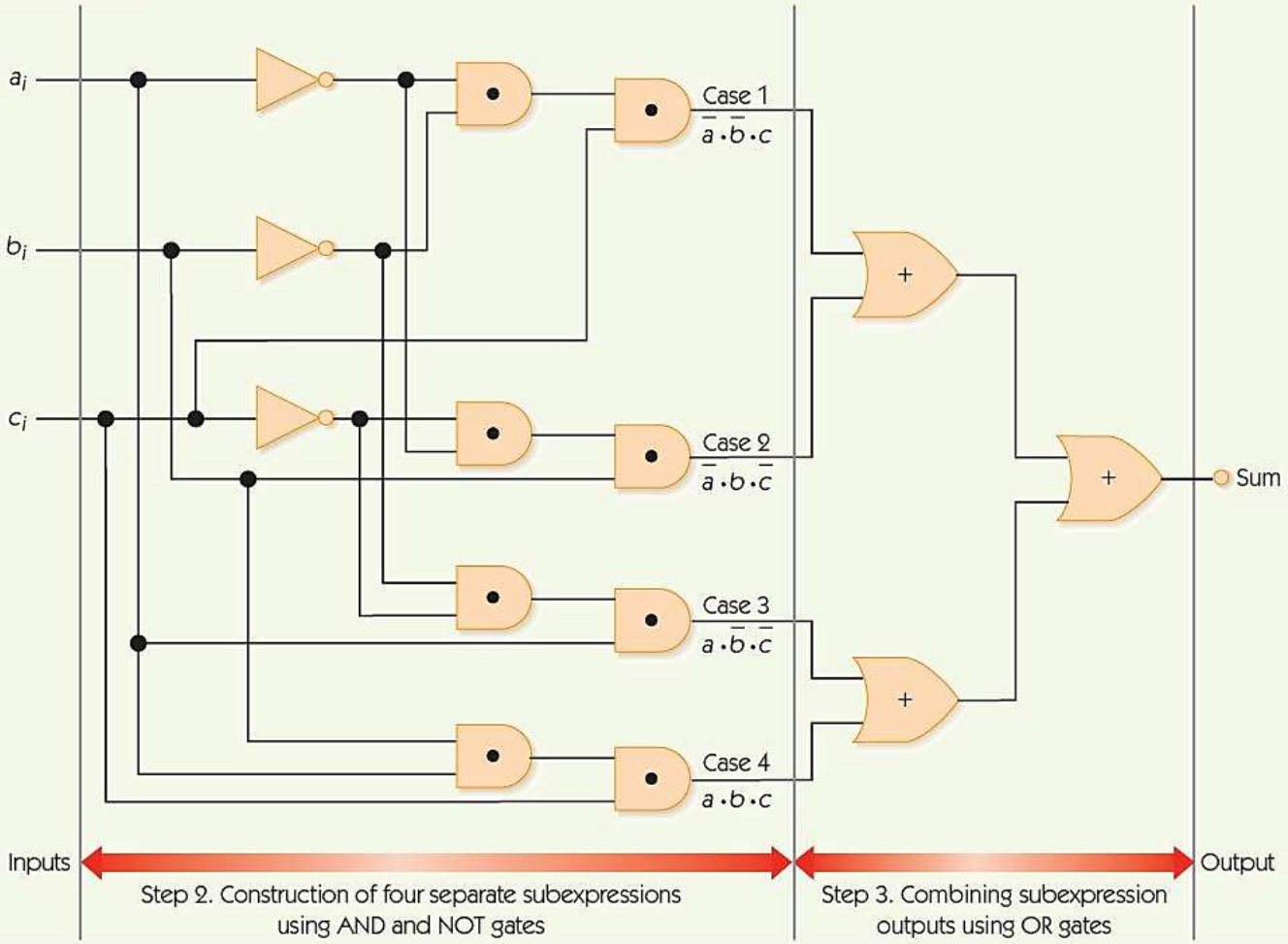
  $(a_i \bullet \sim b_i \bullet \sim c_i) + (a_i \bullet b_i \bullet c_i)$
- Carry digit, c_{i+1}, has the Boolean expression

  $(\sim a_i \bullet b_i \bullet c_i) + (a_i \bullet \sim b_i \bullet c_i) +$

  $(a_i \bullet b_i \bullet \sim c_i) + (a_i \bullet b_i \bullet c_i)$

Building Computer Circuits (15 of 18)

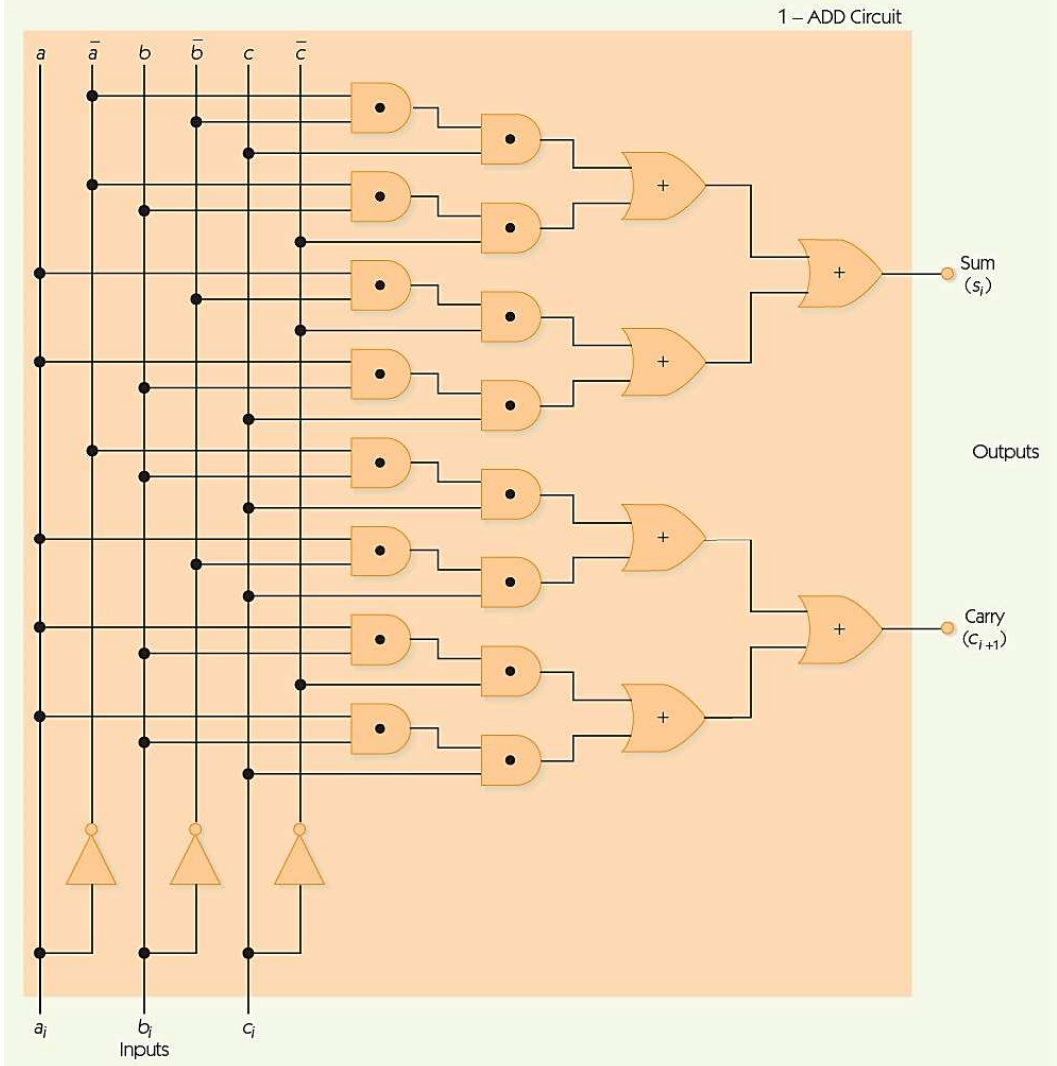
FIGURE 4.30



Sum output for the 1-ADD circuit

Building Computer Circuits (16 of 18)

FIGURE 4.31



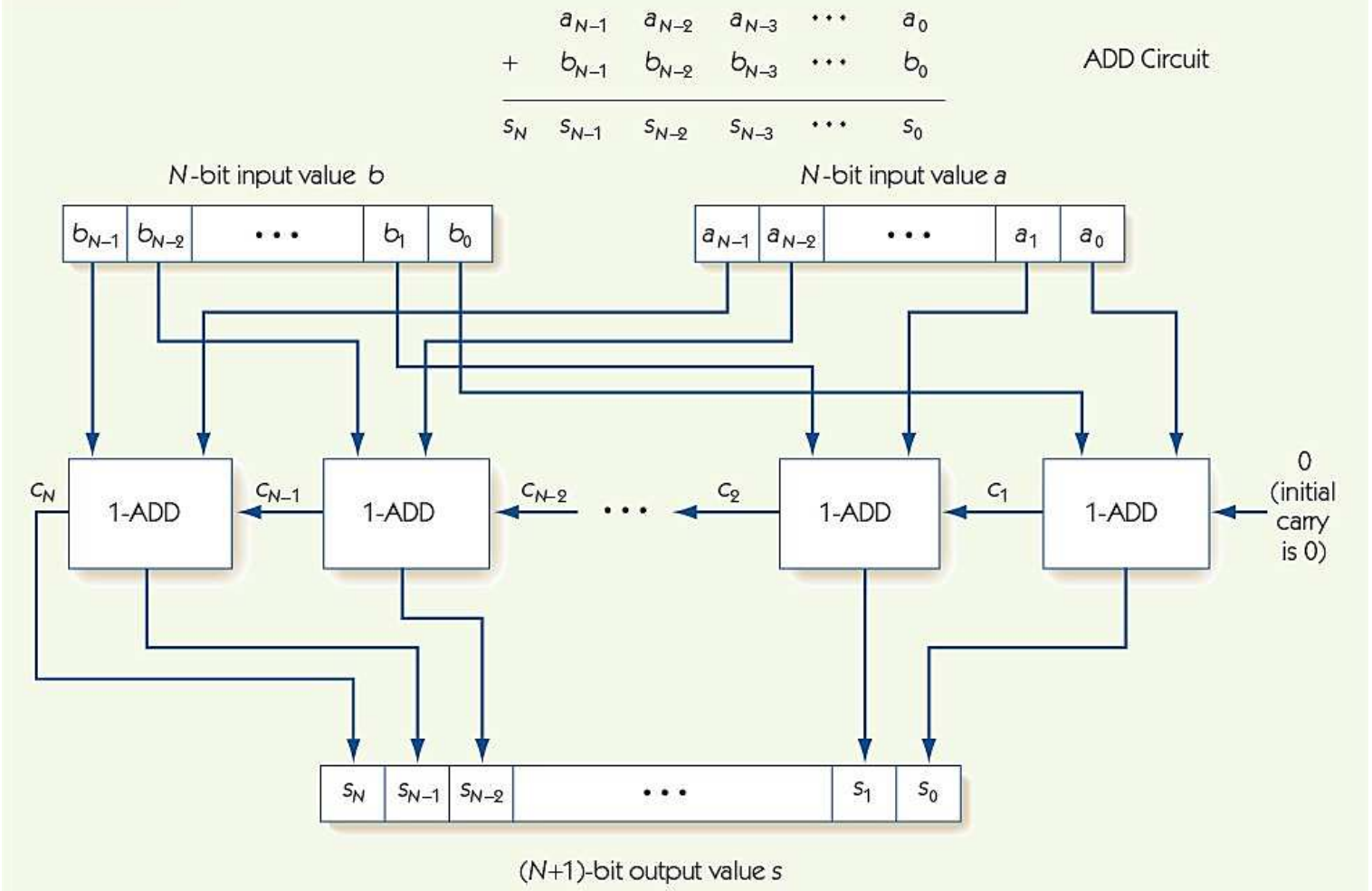
Complete 1-ADD circuit for 1-bit binary addition

Building Computer Circuits (17 of 18)

- N-bit adder circuit
- Input: $a_0a_2\ldots a_{n-1}$ and $b_0b_2\ldots b_{n-1}$, where a_i and b_i are individual bits
- a_0 and b_0 are least significant digits: ones place
- Pair up corresponding bits: a_0 with b_0, a_1 with b_1, etc.
- Run 1-ADD on a_0 and b_0, with fixed carry in $c_0 = 0$
- Feed carry out c_1 to next 1-ADD and repeat

Building Computer Circuits (18 of 18)

FIGURE 4.32



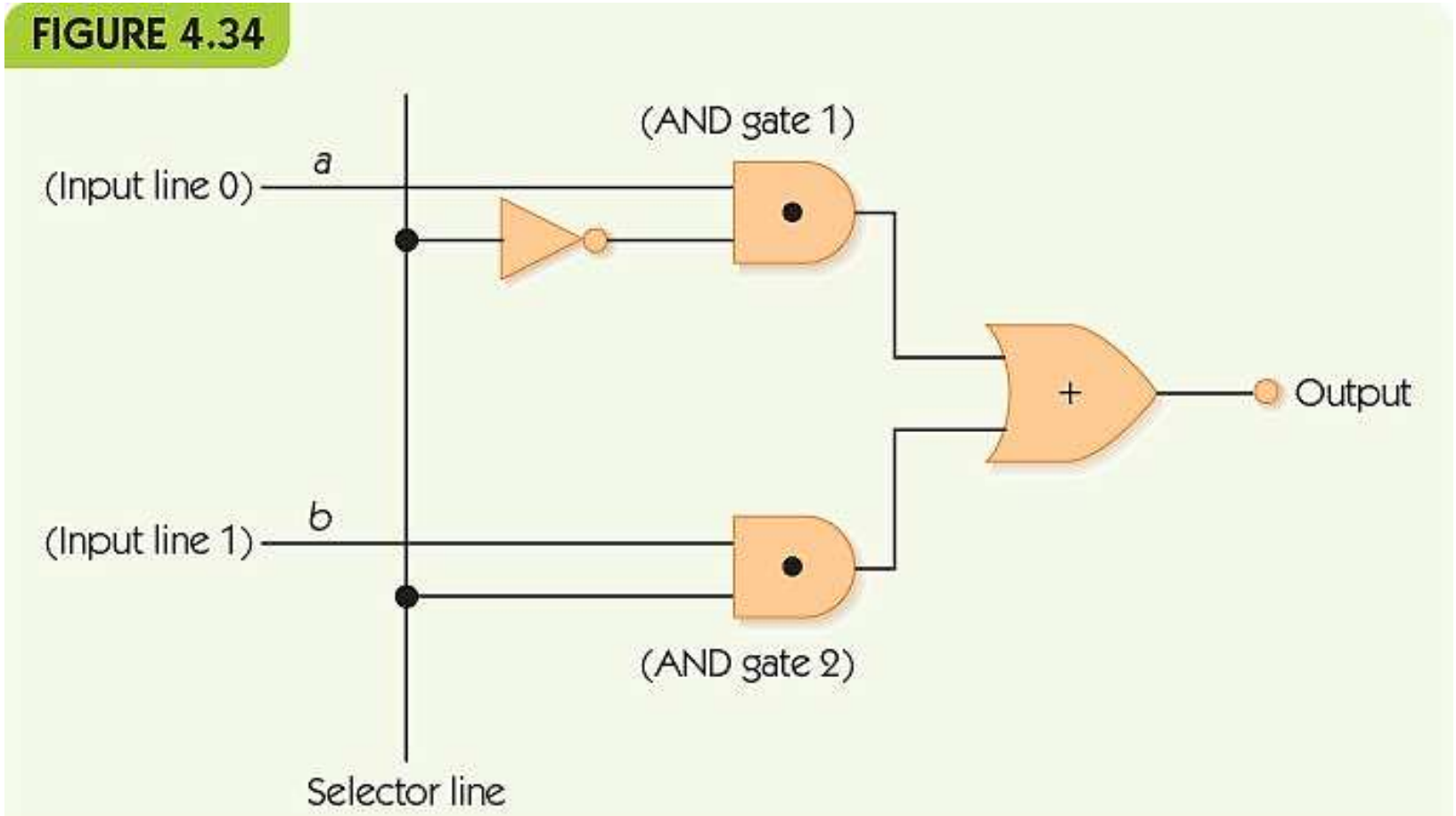
The complete full adder ADD circuit

Control Circuits (1 of 7)

- **Control circuits** make decisions, determine order of operations, select data values
- **Multiplexer** selects one from among many inputs
  - 2^N input lines
  - N selector lines
  - 1 output line
- Each input line corresponds to a unique pattern on selector lines
- That input value is passed to output

Control Circuits (2 of 7)

FIGURE 4.34



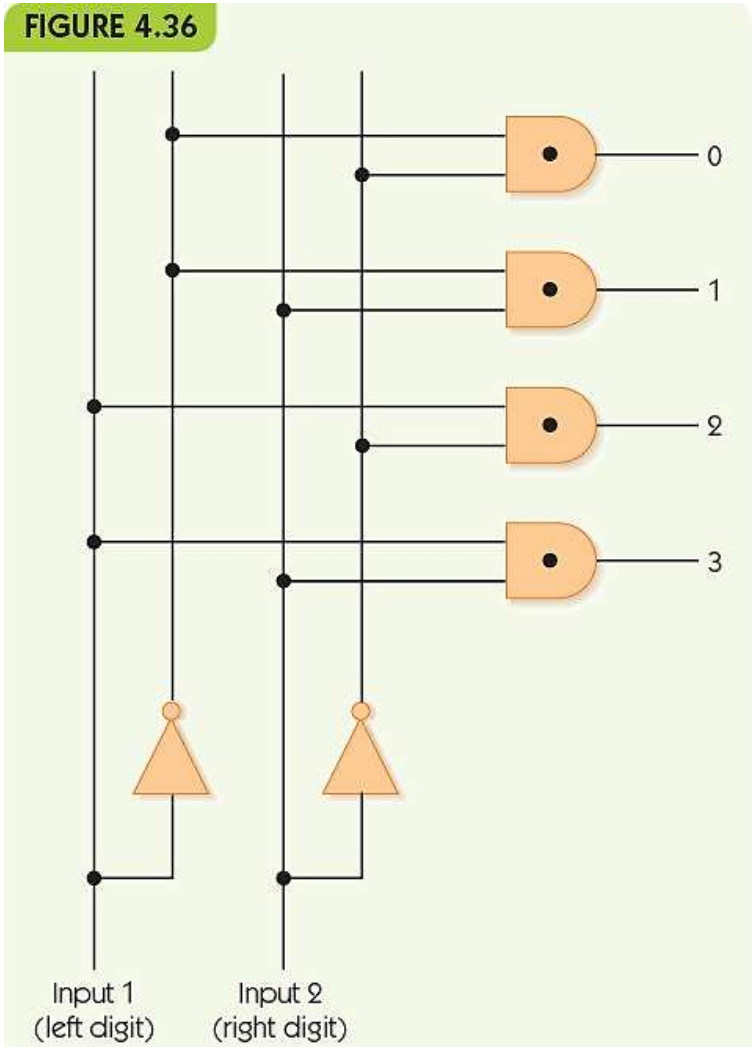
A two-input multiplexor circuit

Control Circuits (3 of 7)

- **Decoder** sends a signal out to only one output chosen by its input
  - N input lines
  - 2^N output lines
- Each output line corresponds to a unique pattern on input lines
- Only the chosen output line produces 1, all others output 0

Control Circuits (4 of 7)

FIGURE 4.36



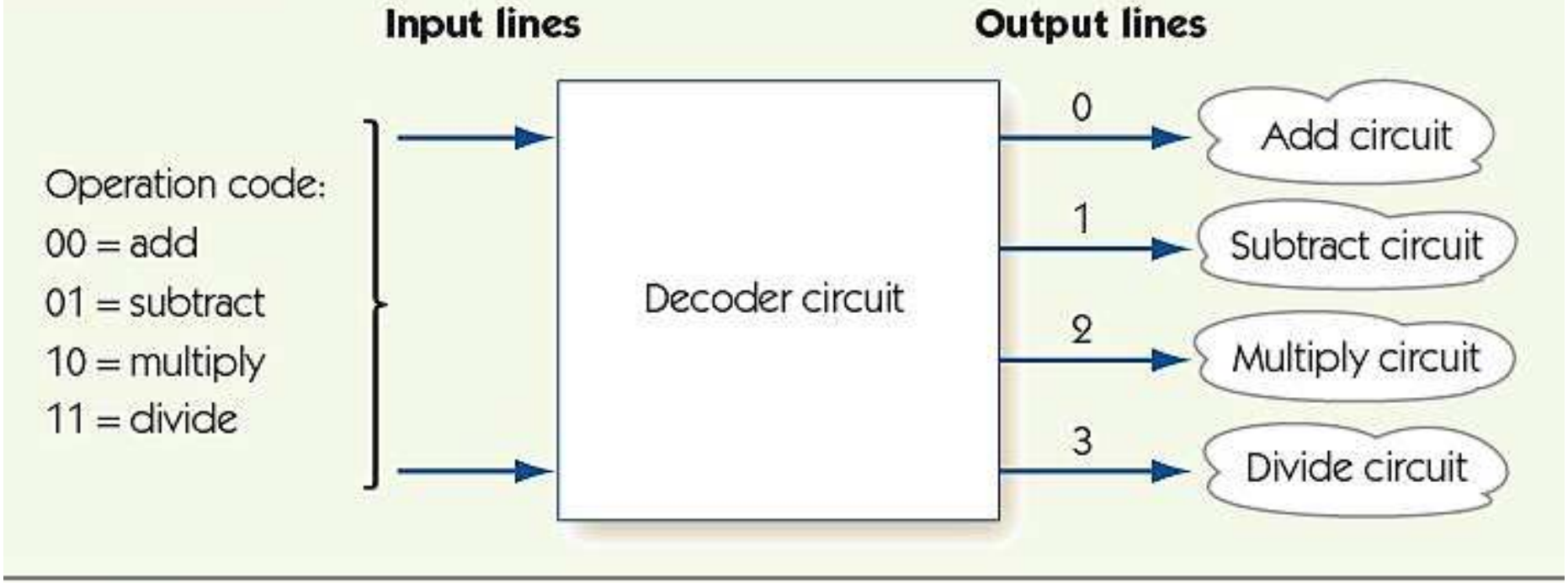
A 2-to-4 decoder circuit

Control Circuits (5 of 7)

- Decoder circuit uses
  - To select a single arithmetic instruction, given a code for that instruction
  - Code activates one output line; that line activates corresponding arithmetic circuit
- Multiplexer circuit uses
  - To choose one data value from among a set, based on selector pattern
  - Many data values flow into the multiplexer, only the selected one comes out

Control Circuits (6 of 7)

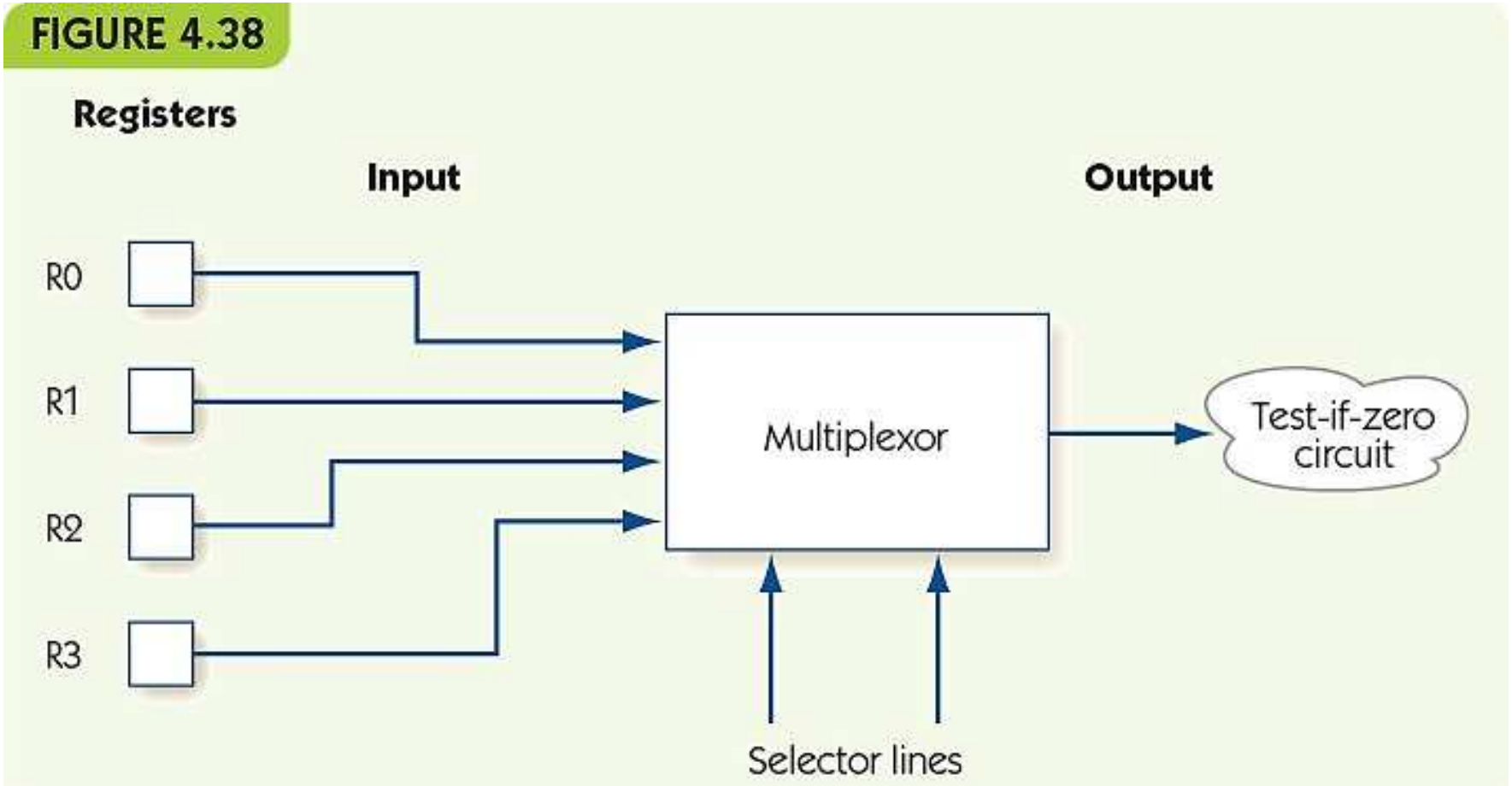
FIGURE 4.37



Example of the use of a decoder circuit

Control Circuits (7 of 7)

FIGURE 4.38



Example of the use of a multiplexor circuit

Summary (1 of 2)

- Computers use binary representations because they maximize reliability for electronic systems.
- Many kinds of data may be represented at least in an approximate digital form using binary values.
- Data can be compressed.
  - Lossy: Some information is lost in the process.
  - Lossless: No information is lost in the process.
- Boolean logic describes how to build and manipulate expressions that are true/false.

Summary (2 of 2)

- We can build logic gates that act like Boolean operators using transistors
- Circuits may be built from logic gates; circuits correspond to Boolean expressions
- Sum-of-products is a circuit design algorithm: takes a specification and ends with a circuit
- We can build circuits for basic algorithmic tasks
  - Comparisons (compare-for-equality circuit)
  - Arithmetic (adder circuit)
  - Control (multiplexer and decoder circuits)