# Chapter 4

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

**INVITATION TO**
**Computer Science**

6TH EDITION

# Objectives

After studying this chapter, students will be able to:

- Translate between base-ten and base-two numbers, and represent negative numbers using both sign-magnitude and two's complement representations
- Explain how floating-point numbers, character, 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 computer hardware/circuits

# Objectives (continued)

After studying this chapter, students will be able to:

- Construct circuits using the sum-of-products circuit design algorithm, and analyze simple circuits to determine their truth tables
- Explain how the compare-for-equality (CE) circuit works and its construction from one-bit CE circuits, and do the same for the adder circuit and its one-bit adder parts
- Describe the purpose and workings of multiplexor and decoder control circuits

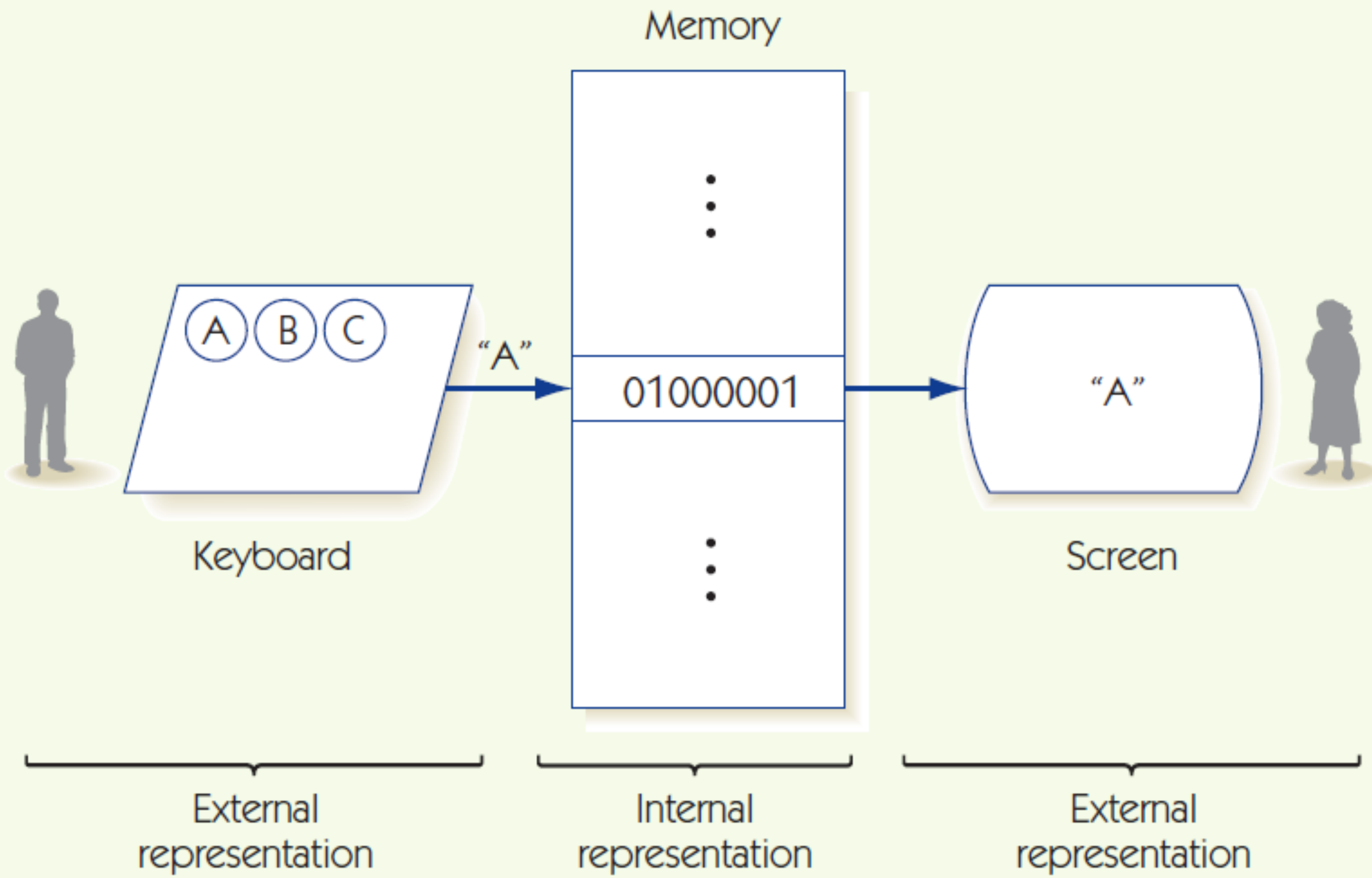
# Introduction

- This chapter is about how computers work
- All computing devices are built on the ideas in this chapter
  - Laptops, desktops
  - Servers, supercomputers
  - Game systems, cell phones, MP3 players
  - Calculators, singing get-well cards
  - Embedded systems, in toys, cars, microwaves, etc.

# The Binary Numbering System

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

**FIGURE 4.1**




Distinction between external and internal representation of information

# The Binary Numbering System (continued)

- 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, 9
  - Each place corresponds to a power of 10
  - $1{,}943 = 1 * 10^3 + 9 * 10^2 + 4 * 10^1 + 3 * 10^0$
- Base two:
  - Uses 2 digits: 0, 1
  - Each place corresponds to a power of 2
  - $1101 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$

## FIGURE 4.2

| Binary | Decimal | Binary | Decimal |
|---|---|---|---|
| 0 | 0 | 10000 | 16 |
| 1 | 1 | 10001 | 17 |
| 10 | 2 | 10010 | 18 |
| 11 | 3 | 10011 | 19 |
| 100 | 4 | 10100 | 20 |
| 101 | 5 | 10101 | 21 |
| 110 | 6 | 10110 | 22 |
| 111 | 7 | 10111 | 23 |
| 1000 | 8 | 11000 | 24 |
| 1001 | 9 | 11001 | 25 |
| 1010 | 10 | 11010 | 26 |
| 1011 | 11 | 11011 | 27 |
| 1100 | 12 | 11100 | 28 |
| 1101 | 13 | 11101 | 29 |
| 1110 | 14 | 11110 | 30 |
| 1111 | 15 | 11111 | 31 |

Binary-to-decimal conversion table

# The Binary Numbering System (continued)

- 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 9:
    - 9/2 = 4 remainder 1, binary number = 1
    - 4/2 = 2 remainder 0, binary number = 01
    - 2/2 = 1 remainder 0, binary number = 001
    - 1/2 = 0 remainder 1, binary number = 1001

# The Binary Numbering System (continued)

- Computers use fixed-length binary numbers for integers, e.g., with 4 bits could represent 0 to 15
- **Arithmetic overflow**: when 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=0 with carry of 1
- Example: 0101 + 0011 = 1000

# The Binary Numbering System (continued)

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

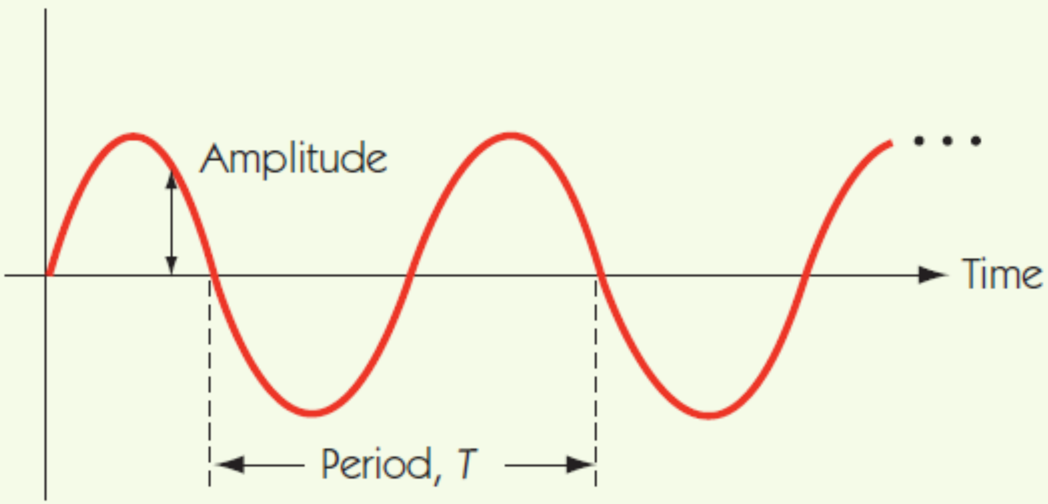
# The Binary Numbering System (continued)

- Floating point numbers use binary scientific notation
  - **Scientific notation**, base 10: $1.35 \times 10^{-5}$
  - 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 (16-bit numbers for each character)

# The Binary Numbering System (continued)

- 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 periods per time unit

**FIGURE 4.4**

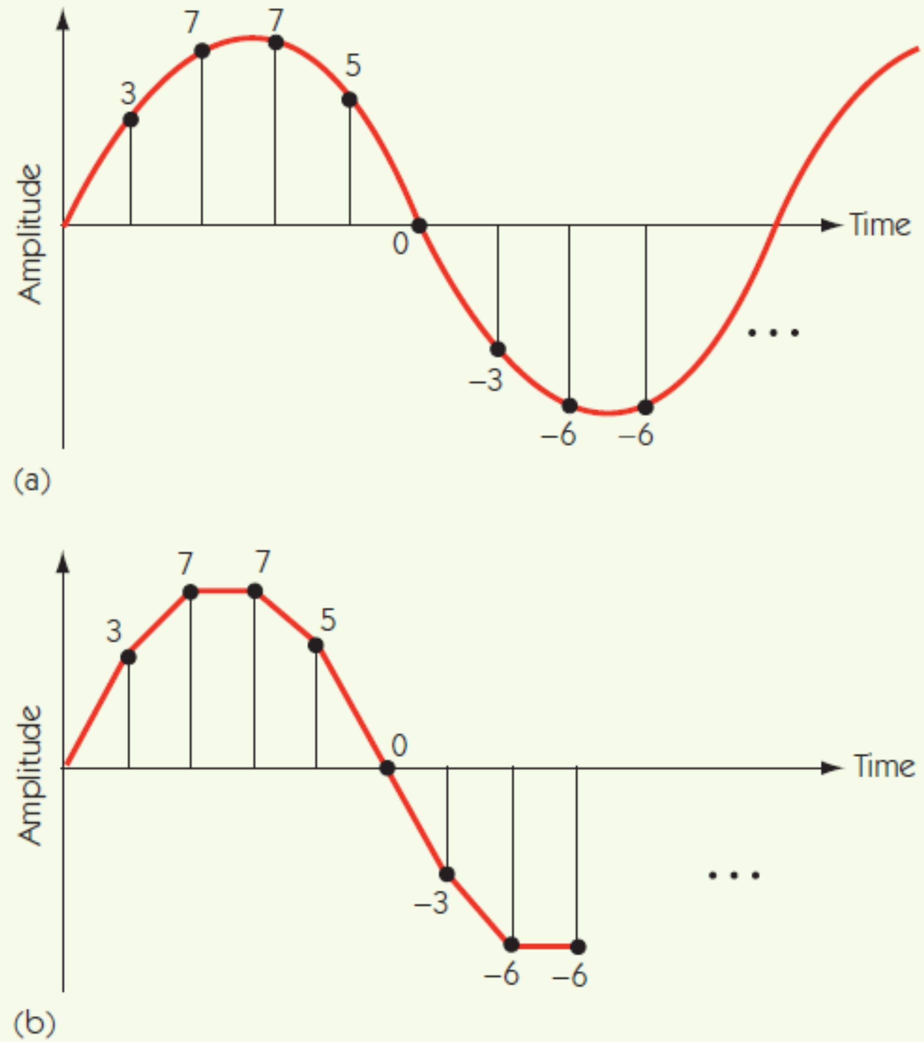



Example of sound represented as a waveform

# The Binary Numbering System (continued)

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

FIGURE 4.5

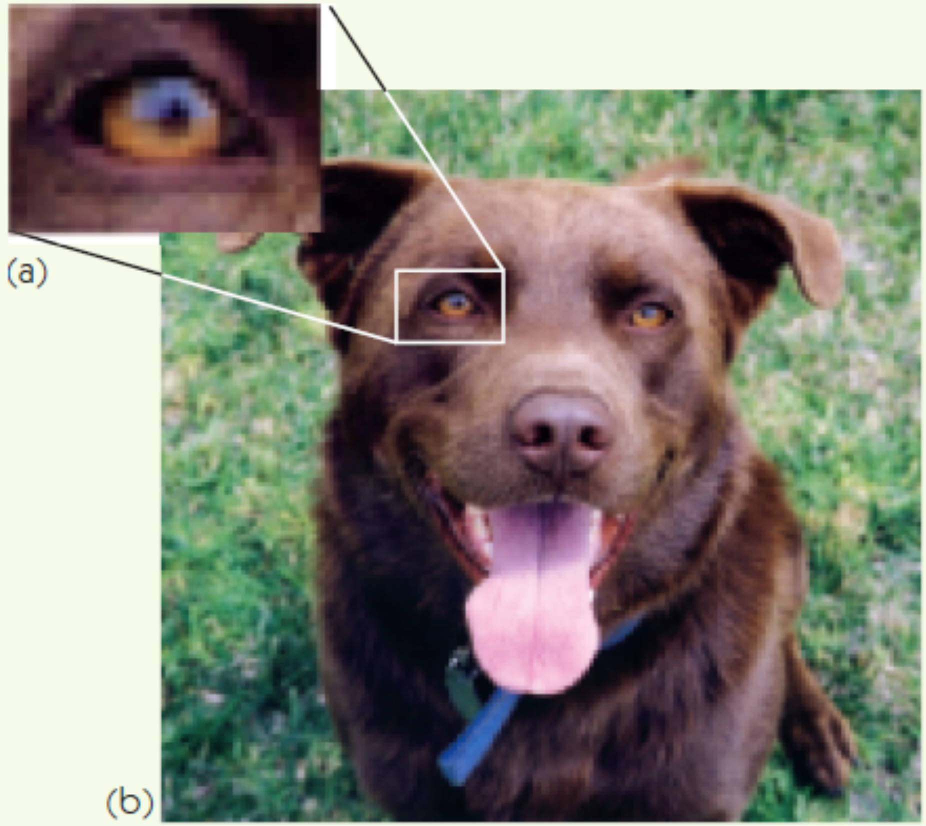



Digitization of an analog signal
(a) Sampling the original signal
(b) Recreating the signal from the sampled values

# The Binary Numbering System (continued)

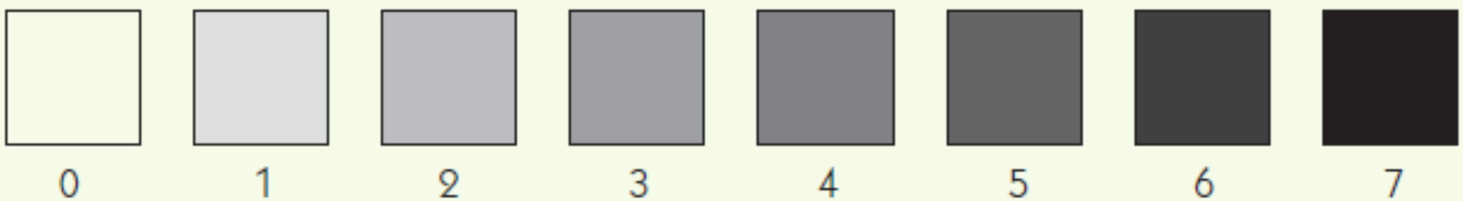
- Image sampling: record color or intensity at fixed, discrete intervals in two dimensions
- Pixels: individual recorded samples
- **RGB encoding scheme**:
  - Colors are combinations of red, green, and blue
  - One byte each for red, green, and blue
- **Raster graphics** store picture as two-d grid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

FIGURE 4.6




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

FIGURE 4.7




An eight-level gray scale

# The Binary Numbering System (continued)

- Data size: how much to store:
  - 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

## FIGURE 4.8

| 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) |

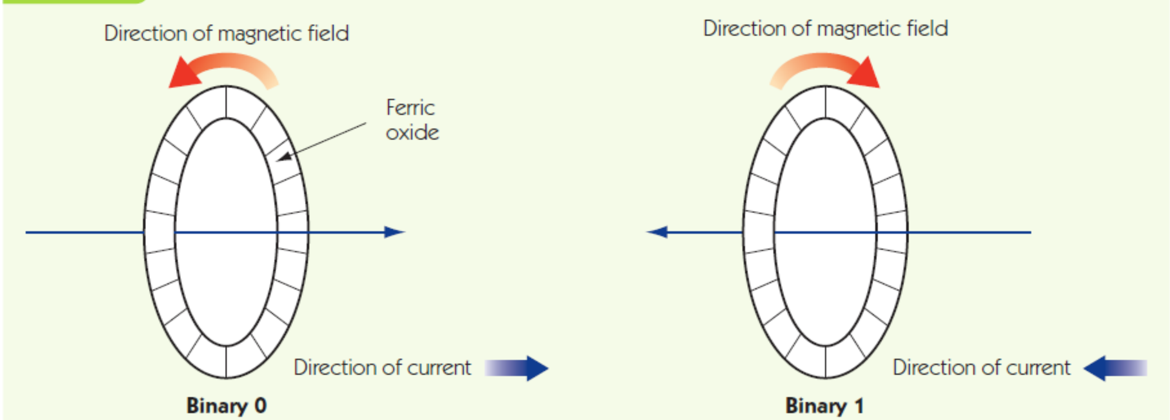
Using variable-length code sets

(a) Fixed length
(b) Variable length

# The Binary Numbering System (continued)

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

FIGURE 4.9

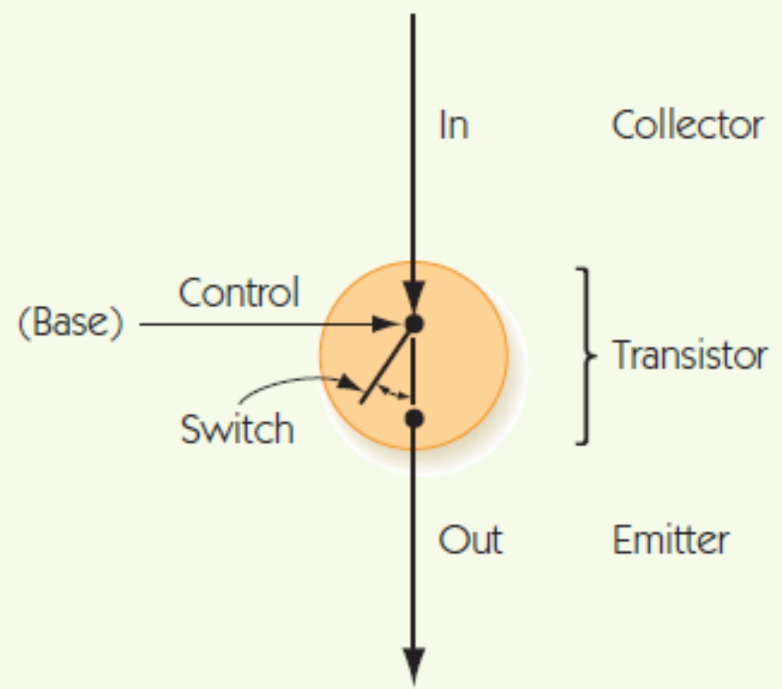



Using magnetic cores to represent binary values

# The Binary Numbering System (continued)

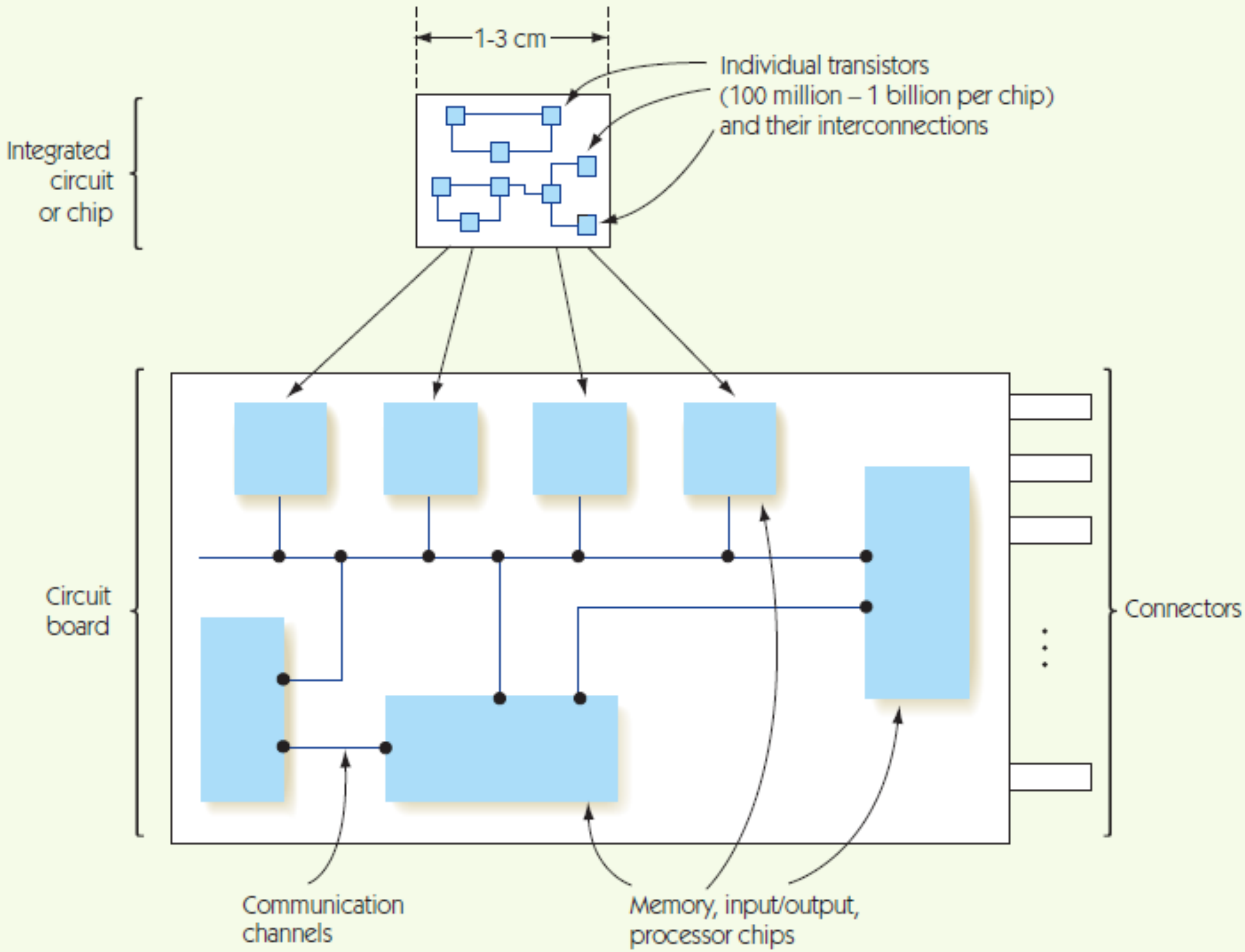
- **Transistors**
  - Solid-state switches
  - Change on/off when given power on control line
  - Extremely small (billions per chip)
  - Enable computers that work with **gigabytes** of data

FIGURE 4.11




Simplified model of a transistor

FIGURE 4.10




Relationships among transistors, chips, and circuit boards

# Boolean Logic and Gates

- **Boolean logic:** rules for manipulating true/false
- Boolean expressions can be converted to circuits
- **Hardware design/logic design** pertains to the design and construction of new circuits
- Note that 1/0 of binary representations 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 (continued)

- **Truth tables** lay out true/false values for Boolean expressions, for each possible true/false input
- Example: (a • b) + (a • ~b)

| **a** | **b** | **~b** | **(a • b)** | **(a • ~b)** | **(a • b) + (a • ~b)** |
|---|---|---|---|---|---|
| true | true | false | true | false | true |
| true | false | true | false | true | true |
| false | true | false | false | false | false |
| false | false | true | false | false | false |

FIGURE 4.12

| Inputs | | Output |
|---|---|---|
| | | a AND b |
| a | b | (also written $a \cdot b$) |
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

Truth table for the AND operation

FIGURE 4.13

| Inputs | | Output |
|---|---|---|
| | | a OR b |
| a | b | (also written $a + b$) |
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

Truth table for the OR operation

FIGURE 4.14

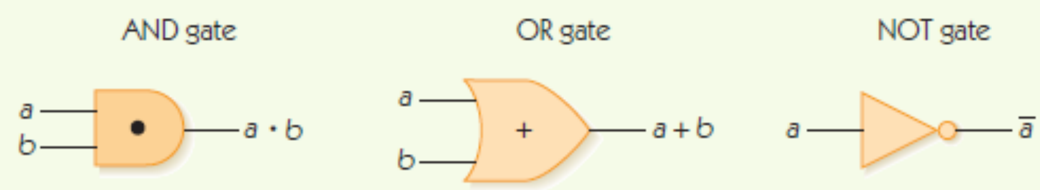
| Input | Output |
|---|---|
| | NOT a |
| a | (also written $\bar{a}$) |
| False | True |
| True | False |

Truth table for the NOT operation

# Boolean Logic and Gates (continued)

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

FIGURE 4.15




| 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 |

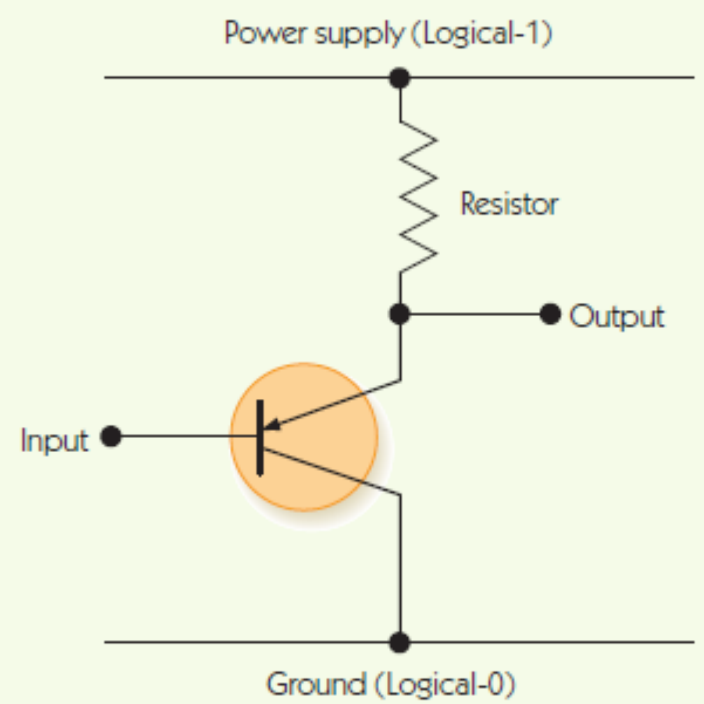
| a | ā |
|---|---|
| 0 | 1 |
| 1 | 0 |

The three basic gates and their symbols

# Boolean Logic and Gates (continued)

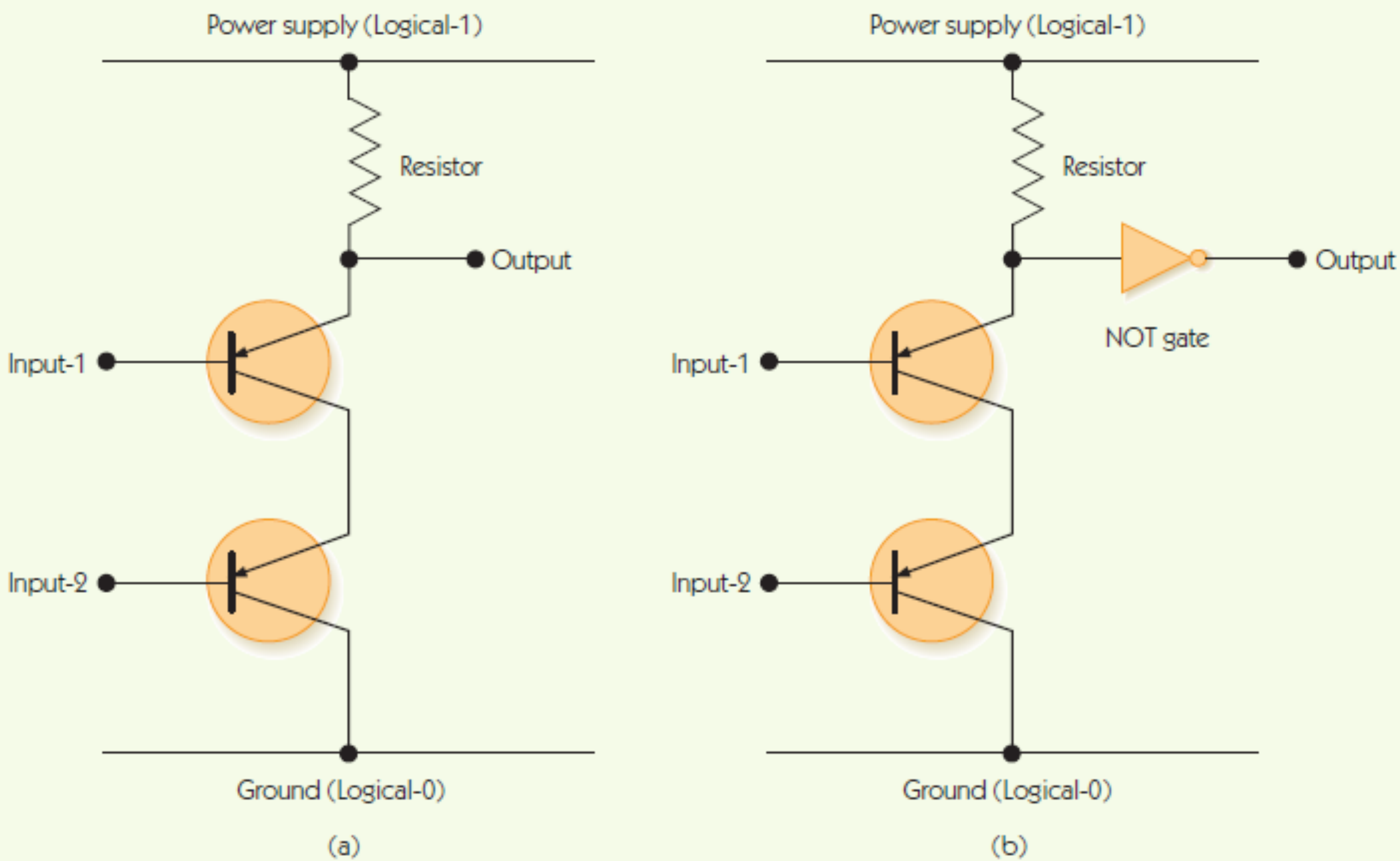
- Gates are built from transistors
- NOT gate: 1 transistor
- AND gate: 3 transistors
- OR gate: 3 transistors
- NAND and NOR: 2 transistors

FIGURE 4.16



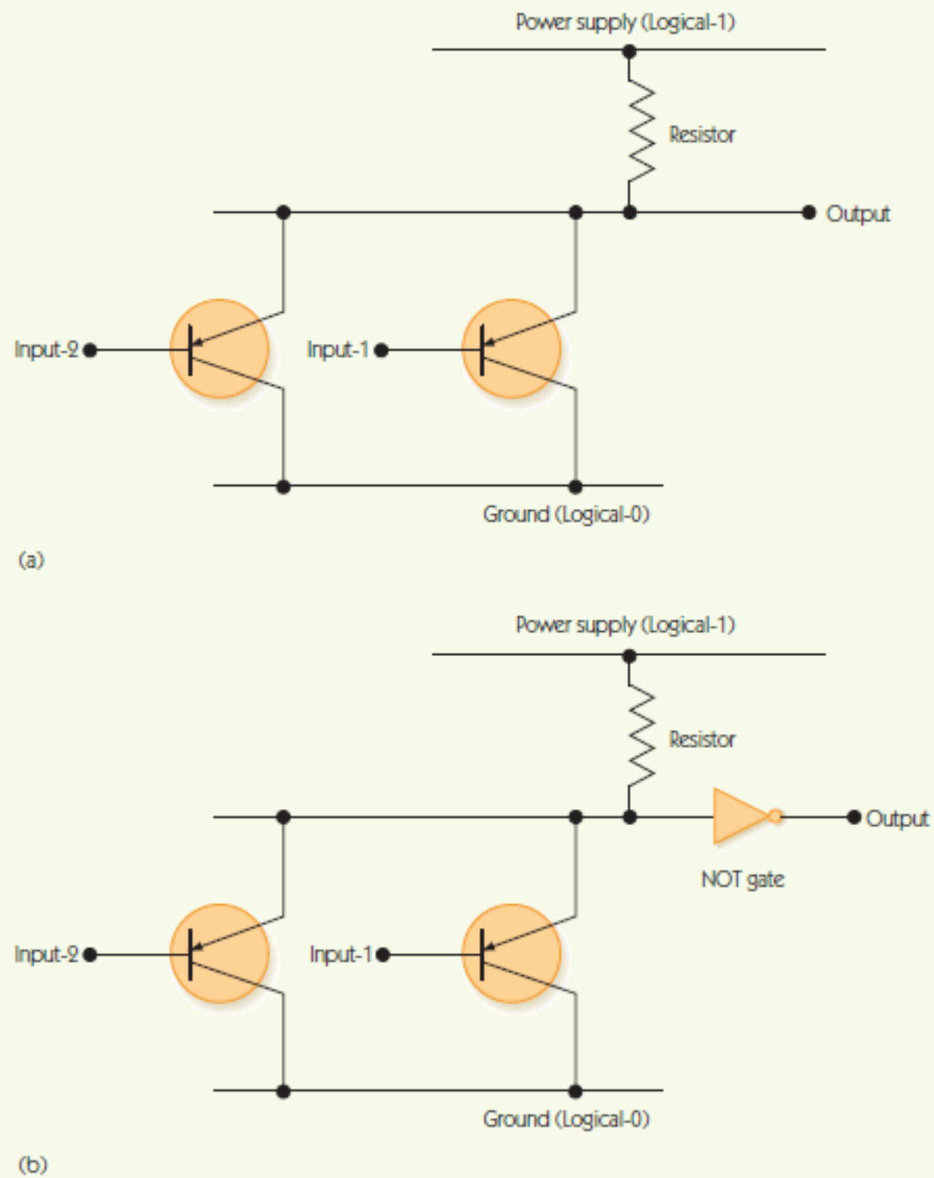

Construction of a NOT gate

FIGURE 4.17




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

**FIGURE 4.18**




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

# Building Computer Circuits

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

FIGURE 4.19

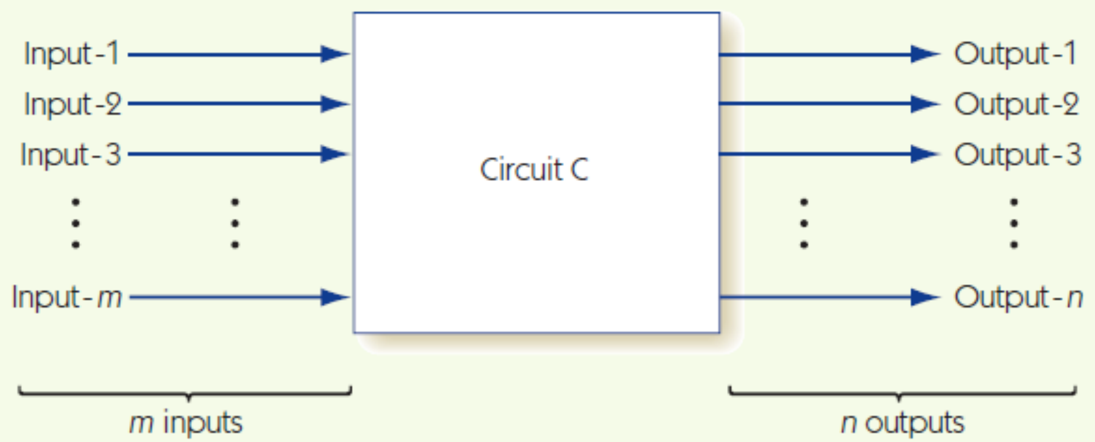



Diagram of a typical computer circuit

# Building Computer Circuits (continued)

- To convert a circuit to a Boolean expression:
  - Start with output and work backwards
    - 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 (continued)

Example from text

- Build truth table:

| a | b | c | Output1 | Output2 |
|---|---|---|---|---|
| 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 |

# Building Computer Circuits (continued)

Example from text

- Find true rows for Output1

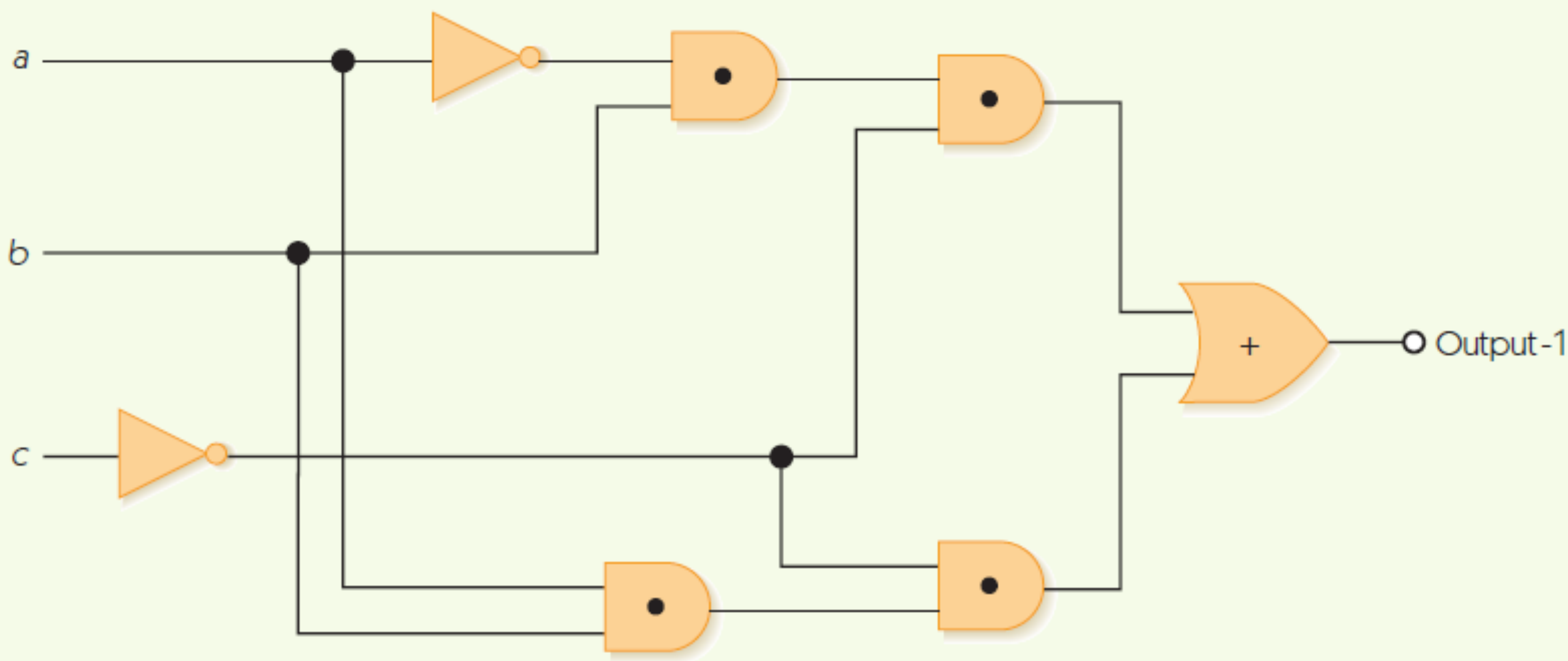
| a | b | c | Output1 | Output2 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| **0** | **1** | **0** | **1** | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| **1** | **1** | **0** | **1** | 1 |
| 1 | 1 | 1 | 0 | 0 |

# Building Computer Circuits (continued)

Example from text

- For each true row, AND input sto make 1
  - $a = 0, b = 1, c = 0: (\sim a \bullet b \bullet \sim c)$
  - $a = 1, b = 1, c = 0: (a \bullet b \bullet \sim c)$
- Combine row subexpressions with OR
  - $(\sim a \bullet b \bullet \sim c) + (a \bullet b \bullet \sim c)$
- Build circuit from expression
- (and repeat for other output)

FIGURE 4.20




Circuit diagram for the output labeled Output-1

**FIGURE 4.21**

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 (continued)

**Compare-for-equality (CE) circuit**

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

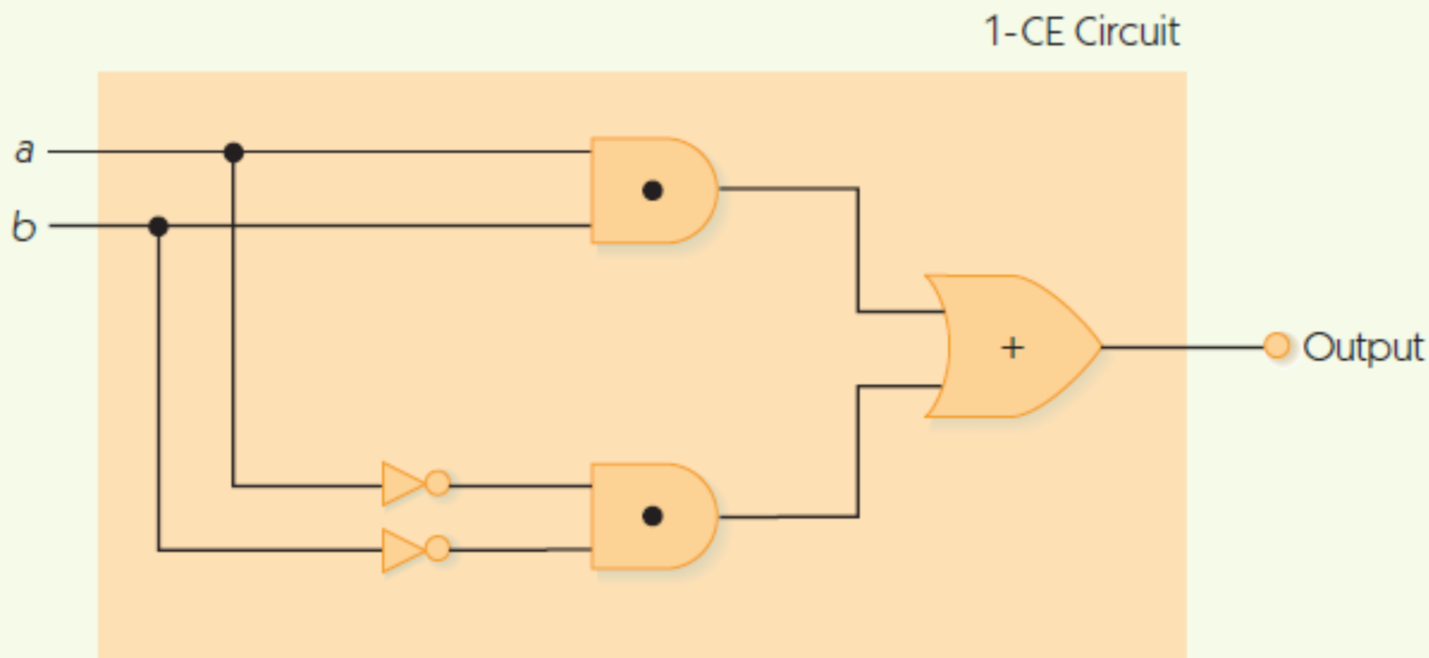
# Building Computer Circuits (continued)

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

| a | b | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

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

## FIGURE 4.22

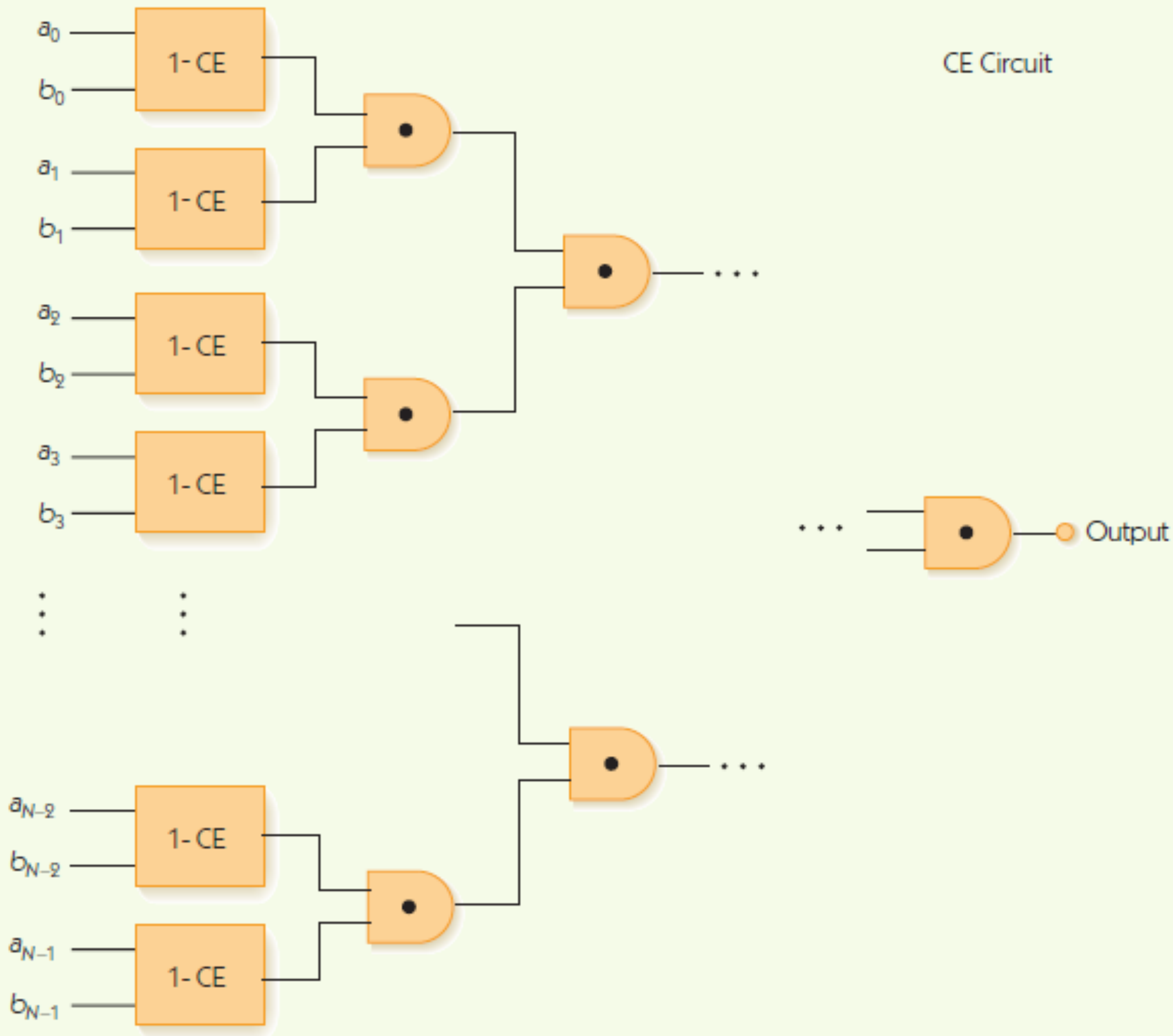



One-bit compare-for-equality circuit

# Building Computer Circuits (continued)

- 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

FIGURE 4.23




*N*-bit compare-for-equality circuit

# Building Computer Circuits (continued)

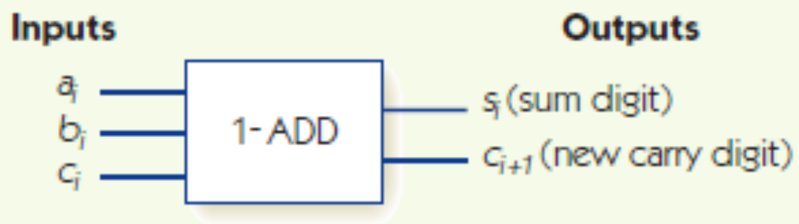
**Full adder circuit**

- 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 one-bit adder (1-ADD)

**FIGURE 4.24**




| 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 (continued)

- Sum digit, $s_i$, has Boolean expression:

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

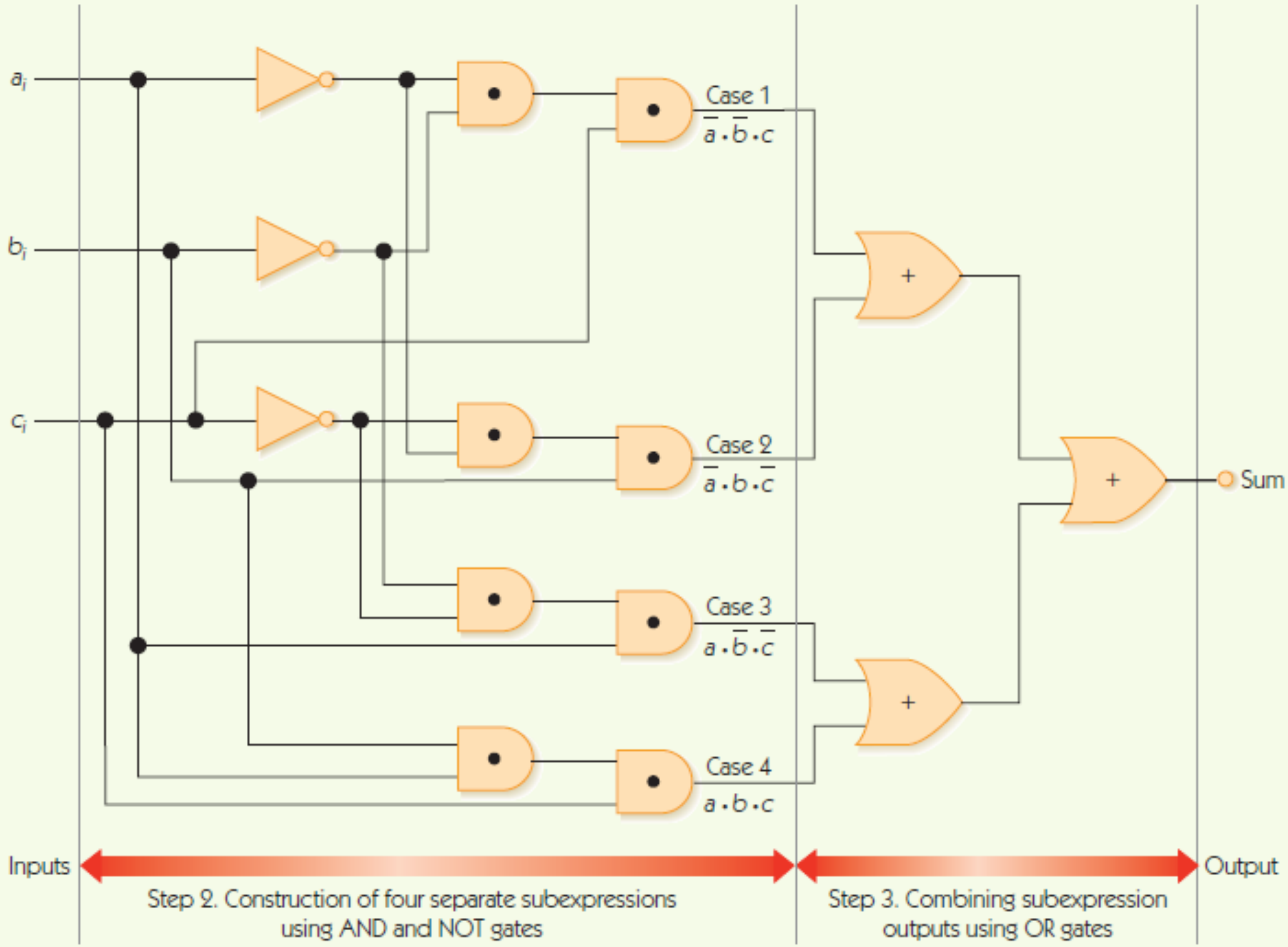
  $(a_i \bullet \sim b_i \bullet \sim c_i) + (a_i \bullet b_i \bullet c_i)$

- Carry digit, $c_{i+1}$, has Boolean expression:

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

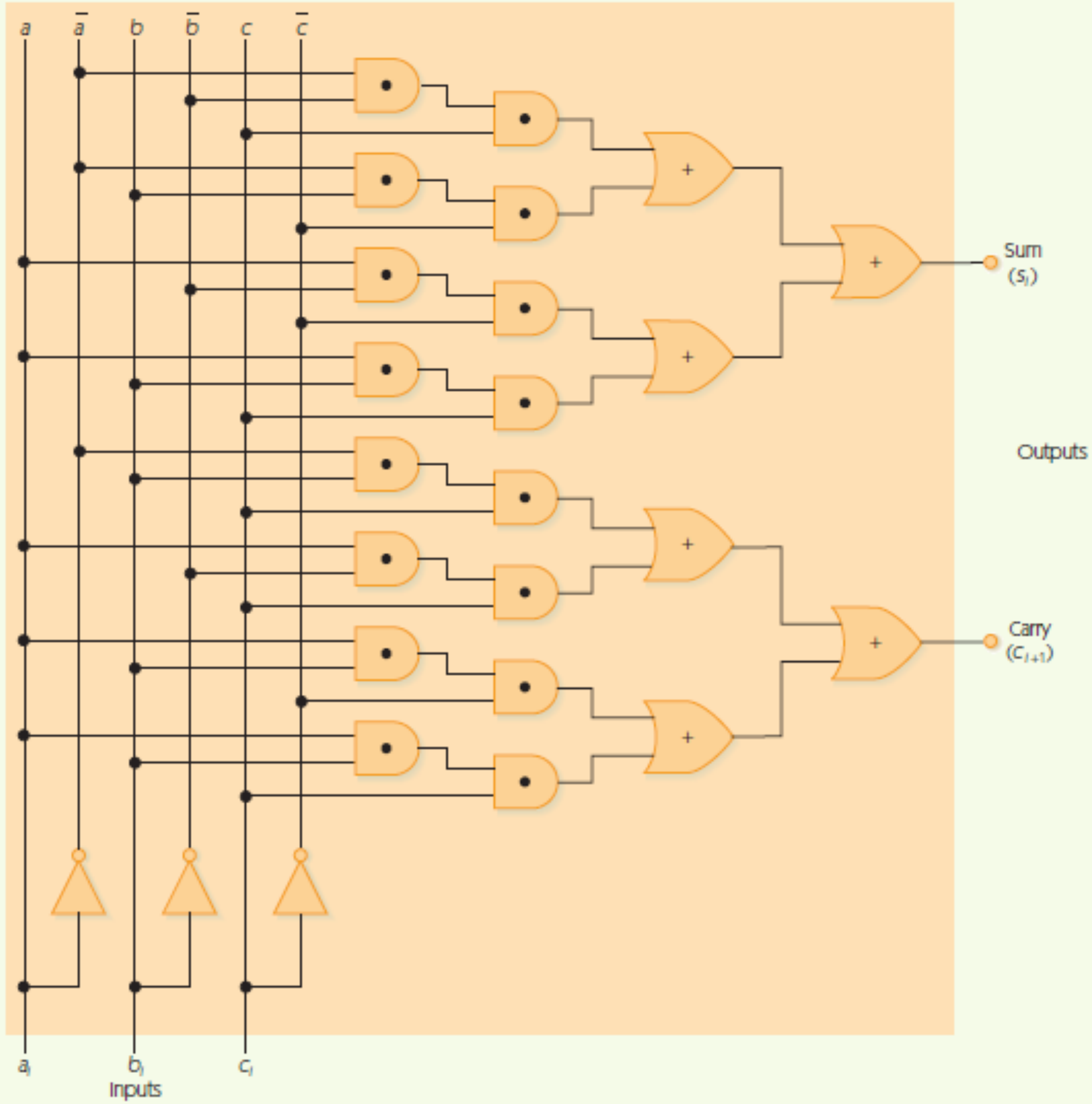
  $(a_i \bullet b_i \bullet \sim c_i) + (a_i \bullet b_i \bullet c_i)$

FIGURE 4.25




Sum output for the 1-ADD circuit

FIGURE 4.26

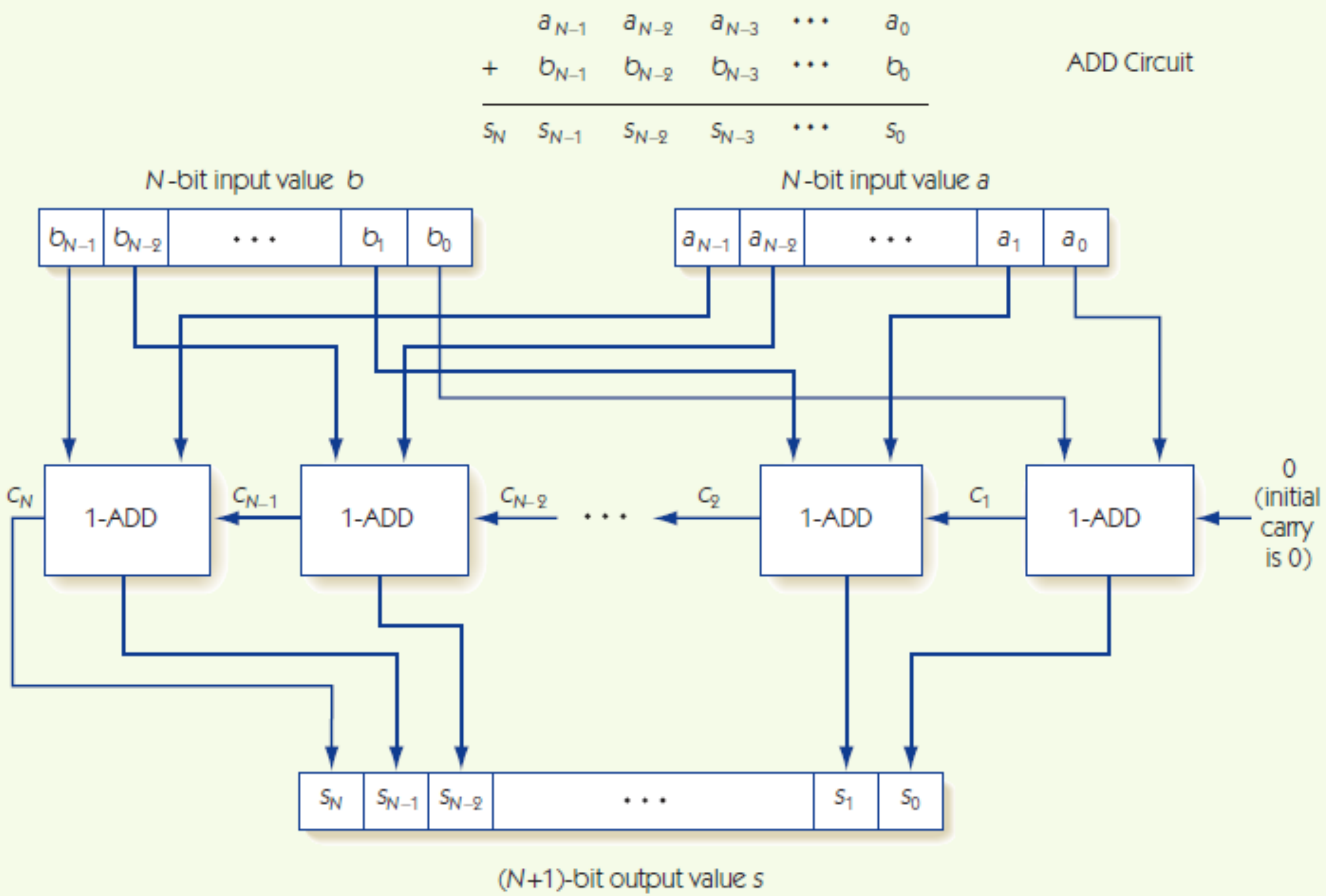



Complete 1-ADD circuit for 1-bit binary addition

# Building Computer Circuits (continued)

- 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

FIGURE 4.27

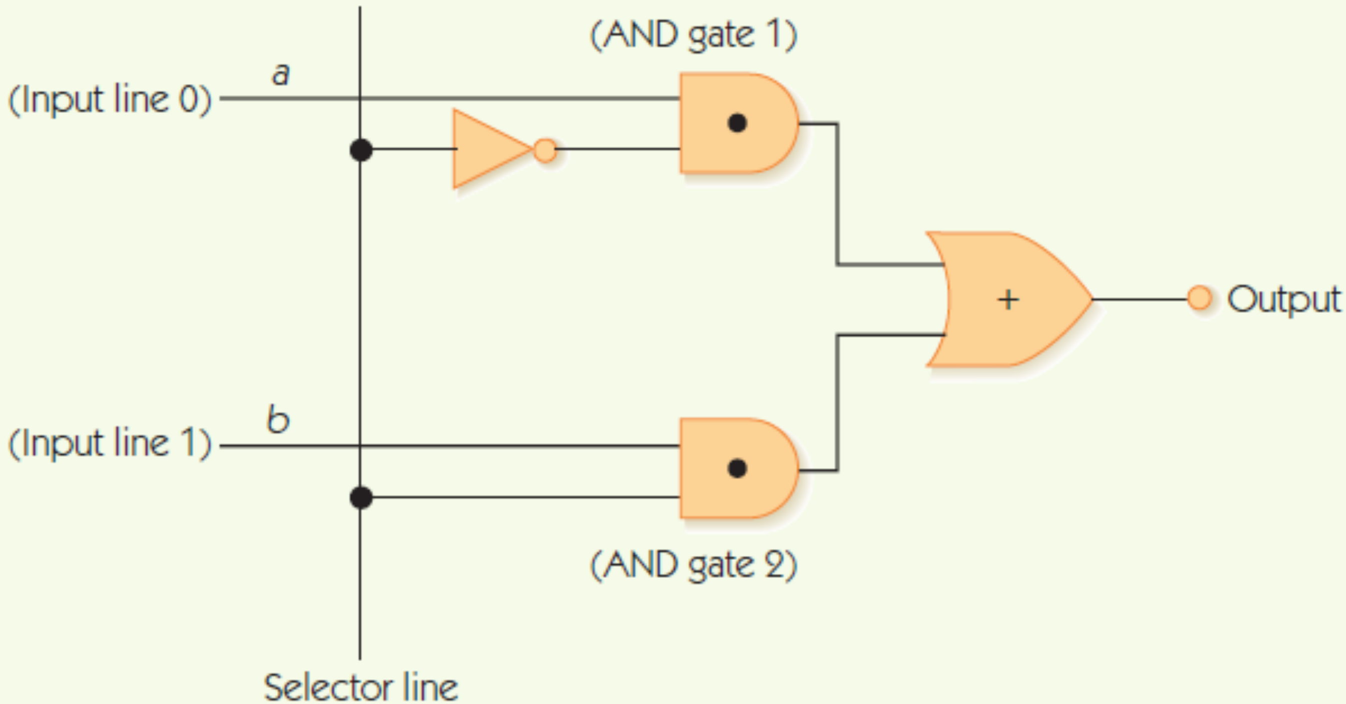



The complete full adder ADD circuit

# Control Circuits

- **Control circuits** make decisions, determine order of operations, select data values
- **Multiplexor** 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

**FIGURE 4.28**

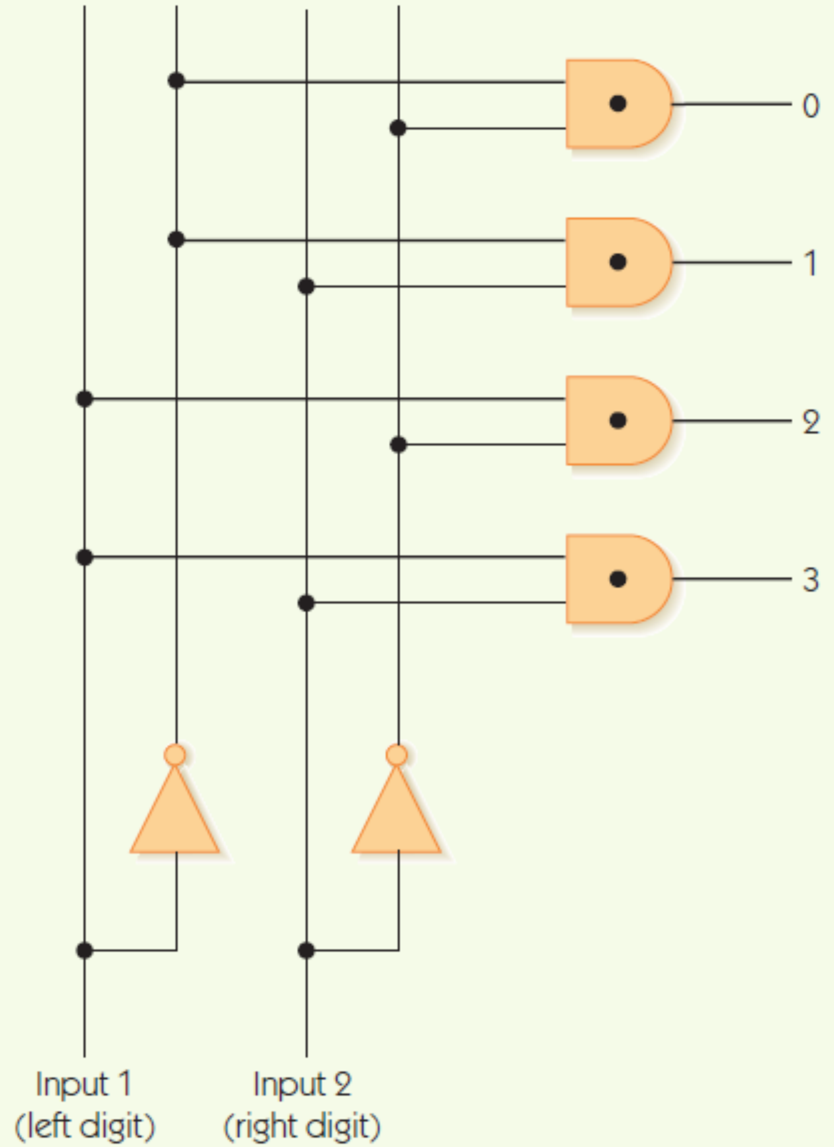



A two-input multiplexor circuit

# Control Circuits (continued)

- **Decoder** sends a signal out 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

FIGURE 4.29

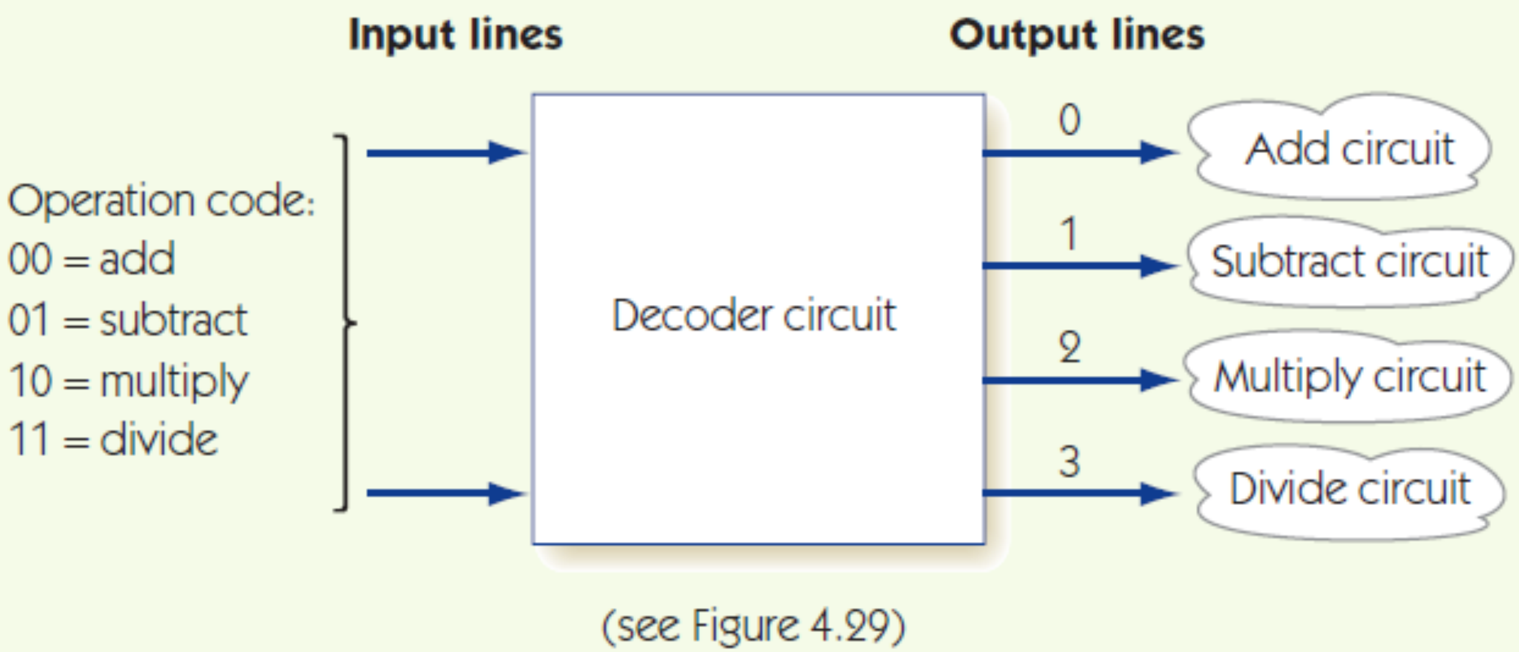



A 2-to-4 decoder circuit

# Control Circuits (continued)

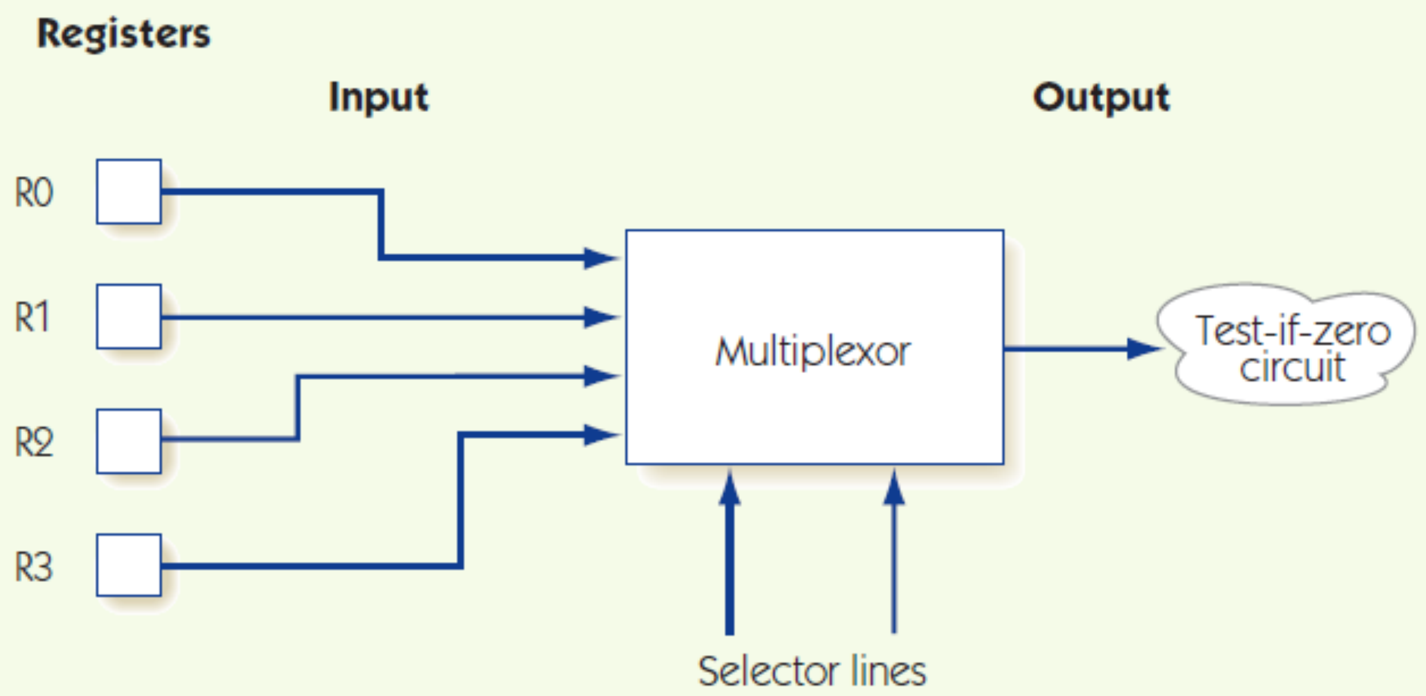
- Decoder circuit uses
  - To select a single arithmetic instruction, given a code for that instruction
  - Code activates one output line, that line activates corresponding arithmetic circuit
- Multiplexor circuit uses
  - To choose one data value from among a set, based on selector pattern
  - Many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

**FIGURE 4.30**




Example of the use of a decoder circuit

FIGURE 4.31




Example of the use of a multiplexor circuit

# Summary

- 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
- Boolean logic describes how to build and manipulate expressions that are true/false
- We can build logic gates that act like Boolean operators using transistors
- Circuits may be built from logic gates: circuits correspond to Boolean expressions

# Summary

- 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 (multiplexor and decoder circuits)