Chapter 4: The Building Blocks: Binary Numbers, Boolean Logic, and Gates

In this chapter, you will learn about:
- The binary numbering system
- Boolean logic and gates
- Building computer circuits
- Control circuits

Introduction
- Chapter 4 focuses on hardware design (also called logic design)
- How to represent and store information inside a computer
- How to use the principles of symbolic logic to design gates
- How to use gates to construct circuits that perform operations such as adding and comparing numbers, and fetching instructions

The Binary Numbering System
- A computer's internal storage techniques are different from the way people represent information in daily lives
- Information inside a digital computer is stored as a collection of binary data

Binary Representation of Numeric and Textual Information
- Binary numbering system
  - Base-2
  - Built from ones and zeros
  - Each position is a power of 2
    \[1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\]
- Decimal numbering system
  - Base-10
  - Each position is a power of 10
    \[3052 = 3 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 2 \times 10^0\]
Representing integers
- Decimal integers are converted to binary integers
  - Given k bits, the largest unsigned integer is $2^k - 1$
    - Given 4 bits, the largest is $2^4 - 1 = 15$
  - Signed integers must also represent the sign (positive or negative)

Representing real numbers
- Real numbers may be put into binary scientific notation: $a \times 2^b$
  - Example: $101.11 \times 2^0$
- Number then normalized so that first significant digit is immediately to the right of the binary point
  - Example: $10111 \times 2^3$
- Mantissa and exponent then stored

Characters are mapped onto binary numbers
- ASCII code set
  - 8 bits per character; 256 character codes
- UNICODE code set
  - 16 bits per character; 65,536 character codes
- Text strings are sequences of characters in some encoding

Multimedia data is sampled to store a digital form, with or without detectable differences
Representing sound data
- Sound data must be digitized for storage in a computer
- Digitizing means periodic sampling of amplitude values

From samples, original sound may be approximated
To improve the approximation:
  - Sample more frequently
  - Use more bits for each sample value
Binary Representation of Sound and Images (continued)

- Representing image data
  - Images are sampled by reading color and intensity values at even intervals across the image
  - Each sampled point is a pixel
  - Image quality depends on number of bits at each pixel

The Reliability of Binary Representation

- Electronic devices are most reliable in a bistable environment
- Bistable environment
  - Distinguishing only two electronic states
  - Current flowing or not
  - Direction of flow
- Computers are bistable: hence binary representations

Binary Storage Devices

- Magnetic core
  - Historic device for computer memory
  - Tiny magnetized rings: flow of current sets the direction of magnetic field
  - Binary values 0 and 1 are represented using the direction of the magnetic field

Binary Storage Devices (continued)

- Transistors
  - Solid-state switches: either permits or blocks current flow
  - A control input causes state change
  - Constructed from semiconductors

Figure 4.9
Using Magnetic Cores to Represent Binary Values

Figure 4.11
Simplified Model of a Transistor
Boolean Logic and Gates: Boolean Logic

- Boolean logic describes operations on true/false values
- True/false maps easily onto bistable environment
- Boolean logic operations on electronic signals may be built out of transistors and other electronic devices

Boolean Logic (continued)

- Boolean operations
  - a AND b
    - True only when a is true and b is true
  - a OR b
    - True when either a is true or b is true, or both are true
  - NOT a
    - True when a is false, and vice versa

Boolean Logic (continued)

- Boolean expressions
  - Constructed by combining together Boolean operations
    - Example: (a AND b) OR ((NOT b) AND (NOT a))
  - Truth tables capture the output/value of a Boolean expression
    - A column for each input plus the output
    - A row for each combination of input values

Example:
(a AND b) OR ((NOT b) AND (NOT a))

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

Gates

- Gates
  - Hardware devices built from transistors to mimic Boolean logic
- AND gate
  - Two input lines, one output line
  - Outputs a 1 when both inputs are 1

Gates (continued)

- OR gate
  - Two input lines, one output line
  - Outputs a 1 when either input is 1
- NOT gate
  - One input line, one output line
  - Outputs a 1 when input is 0 and vice versa
Gates (continued)

- Abstraction in hardware design
- Map hardware devices to Boolean logic
- Design more complex devices in terms of logic, not electronics
- Conversion from logic to hardware design may be automated

Building Computer Circuits: Introduction

- A circuit is a collection of logic gates:
  - Transforms a set of binary inputs into a set of binary outputs
  - Values of the outputs depend only on the current values of the inputs
  - Combinational circuits have no cycles in them (no outputs feed back into their own inputs)

A Circuit Construction Algorithm

- Sum-of-products algorithm is one way to design circuits:
  - Truth table to Boolean expression to gate layout

Sum-of-Products Algorithm for Constructing Circuits

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
A Circuit Construction Algorithm (continued)
- Sum-of-products algorithm
  - Truth table captures every input/output possible for circuit
  - Repeat process for each output line
    - Build a Boolean expression using AND and NOT for each 1 of the output line
    - Combine together all the expressions with ORs
    - Build circuit from whole Boolean expression

Examples Of Circuit Design And Construction
- Compare-for-equality circuit
- Addition circuit
- Both circuits can be built using the sum-of-products algorithm

A Compare-for-equality Circuit
- Compare-for-equality circuit
  - CE compares two unsigned binary integers for equality
  - Built by combining together 1-bit comparison circuits (1-CE)
  - Integers are equal if corresponding bits are equal (AND together 1-CD circuits for each pair of bits)

A Compare-for-equality Circuit (continued)
- 1-CE circuit truth table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

A Compare-for-equality Circuit (continued)
- 1-CE Boolean expression
  - First case: (NOT a) AND (NOT b)
  - Second case: a AND b
  - Combined: (NOT a) AND (NOT b)) OR (a AND b)
An Addition Circuit

- Addition circuit
  - Adds two unsigned binary integers, setting output bits and an overflow
  - Built from 1-bit adders (1-ADD)
  - Starting with rightmost bits, each pair produces
    - A value for that order
    - A carry bit for next place to the left

---

1-ADD truth table

- Input
  - One bit from each input integer
  - One carry bit (always zero for rightmost bit)
- Output
  - One bit for output place value
  - One “carry” bit

---

Control Circuits

- Do not perform computations
- Choose order of operations or select among data values
- Major types of controls circuits
  - Multiplexors
    - Select one of inputs to send to output
  - Decoders
    - Sends a 1 on one output line, based on what input line indicates

---

Multiplexor form

- $2^n$ regular input lines
- $N$ selector input lines
- 1 output line

Multiplexor purpose

- Given a code number for some input, selects that input to pass along to its output
- Used to choose the right input value to send to a computational circuit
Control Circuits (continued)

- **Decoder purpose**
  - Given a number code for some operation, trigger just that operation to take place
  - Numbers might be codes for arithmetic: add, subtract, etc.
  - Decoder signals which operation takes place next

Summary

- Digital computers use binary representations of data: numbers, text, multimedia
- Binary values create a bistable environment, making computers reliable
- Boolean logic maps easily onto electronic hardware
- Circuits are constructed using Boolean expressions as an abstraction
- Computational and control circuits may be built from Boolean gates