CENGAGE

FOUNDATIONS OF COMPUTER SCIENCE

Behrouz Forouzan

FOURTH EDITION

FOUNDATIONS OF COMPUTER SCIENCE 4TH EDITION

BEHROUZ FOROUZAN

CENGAGE

Australia • Brazil • Mexico • Singapore • United Kingdom • United States

CENGAGE

Foundations of Computer Science, **4th Edition**

Behrouz Forouzan

Publisher: Annabel Ainscow

List Manager: Jennifer Grene

Marketing Manager: Anna Reading

Content Project Manager: Phillipa Davidson-Blake

Manufacturing Buyer: Eyvett Davis

Typesetter: SPi Global

Cover Designer: Cyan Design

Cover Image: © Evgeny Turaev/ Shutterstock

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

ISBN: 978-1-4737-5104-0

Cengage Learning EMEA
Cheriton House, North Way,
Andover, Hampshire, SP10 5BE
United Kingdom

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at: **www.cengage.co.uk**

Cengage Learning products are represented in Canada by Nelson Education Ltd.

For your course and learning solutions, visit **www.cengage.co.uk**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Printed in China by RR Donnelley
Print Number: 01 Print Year: 2017

To Ryan, Justin, William, and Benjamin

Contents

Preface

Computers play a large part in our everyday lives and will continue to do so in the future. Computer science is a young discipline that is evolving and progressing. Computer networks have connected people from far-flung points of the globe. Virtual reality is creating three-dimensional images that amaze the eyes. Space exploration owes part of its success to computers. Computer-generated special effects have changed the movie industry. Computers have played important roles in genetics.

Audience

This book is written for both academic and professional audience. The book can be used as a self-study guide for interested professionals. As a textbook, it can be used for a one-semester or one-quarter course. It is designed as the first course in computer science. This book is designed for a CS0 course based on the recommendations of the Association of Computing Machinery (ACM). It covers all areas of computer science in breadth. The book, totally or partially, can also be used in other disciplines where the students need to have a bird's-eye view approach to the computer science.

Changes in the fourth edition

I have made several categories of changes in this edition.

Revised chapters and appendices

Minor changes have been made to almost all the chapters. Two new chapters have been added (Chapters 19 and 20). Some materials have been removed from Chapter 4, expanded and inserted as two new appendices (Appendices I and J).

Organization

The book is made of 20 chapters and 10 appendices.

Chapters

Chapters are intended to provide the basic materials. However, not all chapters are needed for every audience. The professor who teaches the course can decide which chapters to use. We give guidance below.

Appendices

The appendices are intended to provide a quick reference or review of materials needed to understand the concepts discussed in the book. There are ten appendices that can be used by the students for reference and study.

Acronyms

The book contains a list of acronyms for finding the corresponding terms quickly.

Glossary

The book contains an extensive glossary giving full explanations of the terms used in the book.

Pedagogy

Several pedagogical features of this text are designed to make it particularly easy for students to understand the materials.

Visual approach

The book presents highly technical subject matter without complex formulas by using a balance of text and figures. More than 400 figures accompanying the text provide a visual and intuitive opportunity for understanding the material. Figures are particularly important in explaining the relationship between components of a whole. For many students, these concepts are more easily grasped visually than verbally.

Highlighted points

I have repeated important concepts in boxes for quick reference and immediate attention.

Examples and applications

Whenever appropriate, I have included examples that illustrate the concepts introduced in the text.

Algorithms

The inclusion of algorithms in the text helps students with problem solving and programming.

Unified Modeling Language (UML)

Throughout the book I have used UML diagrams to make students familiar with this tool, which is becoming the de facto standard in the industry.

End-of-chapter materials

Each chapter ends with a set of materials that includes the following:

Recommended reading

This section gives a brief list of references relative to the chapter. The references can be used to quickly find the corresponding literature.

Key terms

The new terms used in each chapter are listed at the end of the chapter and their definitions are included in the glossary.

Summary

Each chapter ends with a summary of the material covered by that chapter. The summary consolidates the important learning points in one place for ease of access by students.

Practice set

Each chapter includes a practice set designed to reinforce salient concepts and encourage students to apply them. It consists of three parts: quizzes, questions, and problems.

Quizzes

Quizzes, which are posted on the book website, provide quick concept checking. Students can take these quizzes to check their understanding of the materials. The feedback to the students' responses is given immediately.

Questions

This section contains simple questions about the concepts discussed in the book. Answers to the odd-numbered questions are posted on the book website to be checked by the student.

Problems

This section contains more difficult problems that need a deeper understanding of the materials discussed in the chapter. I strongly recommend that the student trys to solve all of these problems. Answers to the odd-numbered problems are also posted on the book website to be checked by the student.

Professor resources

The book contains complete resources for professors who teach the course. They can be downloaded from the book site. They include:

Presentations

The site includes a set of colorful and animated PowerPoint presentations for teaching the course.

Solutions to practice set

Solutions to all questions and problems are provided on the book website for the use of professors who teach the course.

Student resources

The book contains complete student resources on the book website. They include:

Quizzes

There are quizzes at the end of chapters that can be taken by the students. Students are encouraged to take these quizzes to test their general understanding of the materials presented in the corresponding chapter.

Solutions to odd-numbered practice sets

Solutions to all odd-number questions and problems are provided on the book website for the use of students.

How to use the book

The chapters in the book are organized to provide a great deal of flexibility. I suggest the following:

- Materials provided in Chapters 1 to 8 are essential to understand the rest of the book.
- Materials provided in Chapters 9 to 14 can be taught if the time allows. They can be skipped in a quarter system.
- Chapters 15 to 20 can be taught at the discretion of the professor and the majors of students.

Acknowledgments

It is obvious that the development of a book of this scope needs the support of many people.

Peer reviewers

I would like to acknowledge the contributions from peer reviewers to the development of the book. These reviewers are:

Sam Ssemugabi, UNISA	**Steve Maybank, Birbeck College**
Ronald Chikati, Botswana Accountancy College	**Mario Kolberg, University of Stirling**
Alex Dandadzi, University of Limpopo	**Colin Price, University of Worcester**
Tom Verhoeff, Eindhoven University of Technology	**Boris Cogan, London Metropolitan University**
Stefan Gruner, University of Pretoria	**Thomas Mandl, University of Hildesheim**
Harin Sellahwea, University of Buckingham	**Daphne Becker, University of South Africa**
John Newman, University of Wales	**Lubna Fekry Abdulhai and Osama Abulnaja, King Abdulaziz University**
	Katie Atkinson, University of Liverpool

Publisher staff

Special thanks go to the staff of the publisher.

Andrew Ashwin	**Jennifer Grene**
Annabel Ainscow	**Phillipa Davidson-Blake**

Behrouz A. Forouzan
Los Angeles, CA.
January 2018

Trademarks

Throughout the text we have used several trademarks. Rather than insert a trademark symbol with each mention of the trademark name, we acknowledge the trademarks here and state that they are used with no intention of infringing upon them. Other product names, trademarks, and registered trademarks are the property of their respective owners.

CHAPTER 1

Introduction

The phrase *computer science* has a very broad meaning today. However, in this book, we define the phrase as 'issues related to the computer'. This introductory chapter first tries to find out what a computer is, then investigates other issues directly related to computers. We look first at the **Turing model** as a mathematical and philosophical definition of computation. We then show how today's computers are based on the **von Neumann model**. The chapter ends with a brief history of this culture-changing device . . . the computer.

Objectives

After studying this chapter, the student should be able to:

- Define the Turing model of a computer.
- Define the von Neumann model of a computer.
- Describe the three components of a computer: hardware, data, and software.
- List topics related to computer hardware.
- List topics related to data.
- List topics related to software.
- Give a short history of computers.

1.1 TURING MODEL

The idea of a universal computational device was first described by Alan Turing in 1936. He proposed that all computation could be performed by a special kind of a machine, now called a **Turing machine**. Although Turing presented a mathematical description of such a machine, he was more interested in the philosophical definition of computation than in building the actual machine. He based the model on the actions that people perform when involved in computation. He abstracted these actions into a model for a computational machine that has really changed the world.

1.1.1 Data processors

Before discussing the Turing model, let us define a computer as a **data processor**. Using this definition, a computer acts as a black box that accepts input data, processes the data, and creates output data (Figure 1.1). Although this model can define the functionality of a computer today, it is too general. In this model, a pocket calculator is also a computer (which it is, in a literal sense).

Figure 1.1 *A single-purpose computing machine*

Another problem with this model is that it does not specify the type of processing, or whether more than one type of processing is possible. In other words, it is not clear how many types or sets of operations a machine based on this model can perform. Is it a specific-purpose machine or a general-purpose machine?

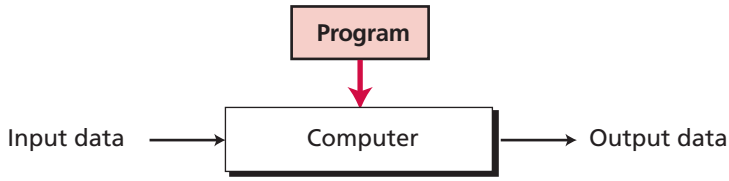
This model could represent a specific-purpose computer (or processor) that is designed to do a single job, such as controlling the temperature of a building or controlling the fuel usage in a car. However, computers, as the term is used today, are *general-purpose* machines. They can do many different types of tasks. This implies that we need to change this model into the Turing model to be able to reflect the actual computers of today.

1.1.2 Programmable data processors

The Turing model is a better model for a general-purpose computer. This model adds an extra element to the specific computing machine: the *program*. A **program** is a set of instructions that tells the computer what to do with data. Figure 1.2 shows the Turing model.

In the Turing model, the **output data** depends on the combination of two factors: the **input data** and the program. With the same input data, we can generate different output if we change the program. Similarly, with the same program, we can generate different

Figure 1.2 *A computer based on the Turing model: programmable data processor*

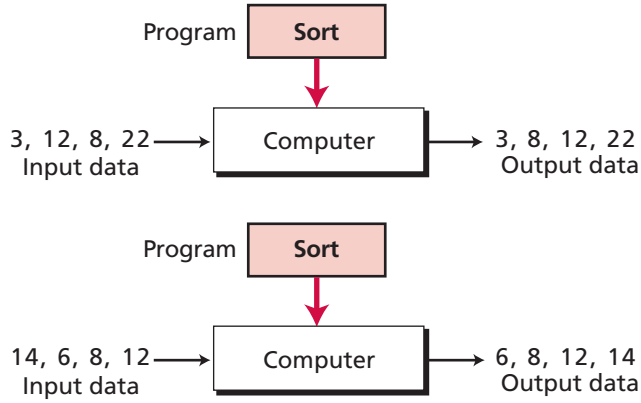


outputs if we change the input data. Finally, if the input data and the program remain the same, the output should be the same. Let us look at three cases.

Same program, different input data

Figure 1.3 shows the same sorting program with different input data. Although the program is the same, the outputs are different, because different input data is processed.

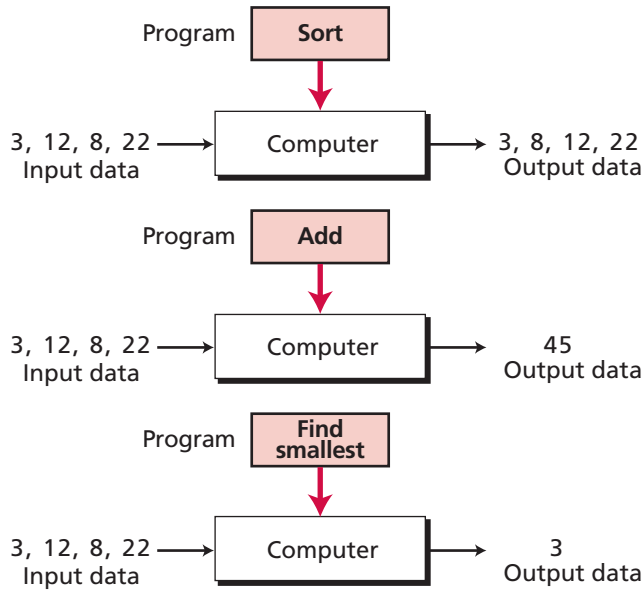
Figure 1.3 *The same program, different data*



Same input data, different programs

Figure 1.4 shows the same input data with different programs. Each program makes the computer perform different operations on the input data. The first program sorts the data, the second adds the data, and the third finds the smallest number.

Figure 1.4 *The same data, different programs*



Same input data, same program

We expect the same result each time if both input data and the program are the same, of course. In other words, when the same program is run with the same input data, we expect the same output.

1.1.3 The universal Turing machine

A *universal Turing machine*, a machine that can do any computation if the appropriate program is provided, was the first description of a modern computer. It can be proved that a very powerful computer and a universal Turing machine can compute the same thing. We need only provide the data and the program—the description of how to do the computation—to either machine. In fact, a universal Turing machine is capable of computing anything that is computable.

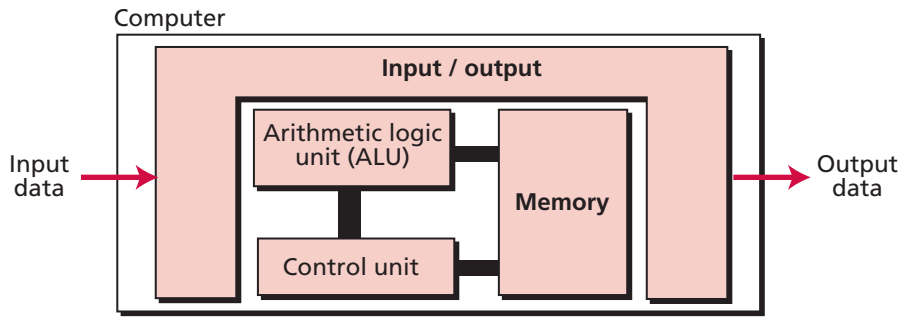
1.2 VON NEUMANN MODEL

Computers built on the Turing universal machine store data in their memory. Around 1944–1945, John von Neumann proposed that, since program and data are logically the same, programs should also be stored in the memory of a computer.

1.2.1 Four subsystems

Computers built on the von Neumann model divide the computer hardware into four subsystems: memory, arithmetic logic unit, control unit, and input/output (Figure 1.5).

Figure 1.5 *The Von Neumann model*



Memory

Memory is the storage area. This is where programs and data are stored during processing. We discuss the reasons for storing programs and data later in the chapter.

Arithmetic logic unit

The **arithmetic logic unit (ALU)** is where calculation and **logical operations** take place. For a computer to act as a data processor, it must be able to do arithmetic operations on data (such as adding a list of numbers). It should also be able to do logical operations on data, as we will see in Chapter 4.

Control unit

The **control unit** controls the operations of the **memory**, ALU, and the input/output subsystem.

Input / output

The **input subsystem** accepts input data and the program from outside the computer, while the **output subsystem** sends the result of processing to the outside world. The definition of the input/output subsystem is very broad: it also includes secondary storage devices such as disk or tape that stores data and programs for processing. When a disk stores data that results from processing, it is considered an output device: when it reads data from the disk, it is considered an input device.

1.2.2 The stored program concept

The von Neumann model states that the program must be stored in memory. This is totally different from the architecture of early computers in which only the data was stored in memory: the programs for their task were implemented by manipulating a set of switches or by changing the wiring system.

The memory of modern computers hosts both a program and its corresponding data. This implies that both the data and programs should have the same format, because they are stored in memory. In fact, they are stored as *binary* patterns in memory—a sequence of 0s and 1s.

1.2.3 Sequential execution of instructions

A program in the von Neumann model is made of a finite number of **instructions**. In this model, the control unit fetches one instruction from memory, decodes it, then executes it. In other words, the instructions are executed one after another. Of course, one instruction may request the control unit to jump to some previous or following instruction, but this does not mean that the instructions are not executed sequentially. Sequential execution of a program was the initial requirement of a computer based on the von Neumann model. Today's computers execute programs in the order that is the most efficient.

1.3 COMPUTER COMPONENTS

We can think of a computer as being made up of three components: computer hardware, data, and computer software.

1.3.1 Computer hardware

Computer hardware today has four components under the von Neumann model, although we can have different types of memory, different types of input/output subsystems, and so on. We discuss computer hardware in more detail in Chapter 5.

1.3.2 Data

The von Neumann model clearly defines a computer as a data processing machine that accepts the input data, processes it, and outputs the result.

Storing data

The von Neumann model does not define how data must be stored in a computer. If a computer is an electronic device, the best way to store data is in the form of an electrical signal, specifically its presence or absence. This implies that a computer can store data in one of two states.

Obviously, the data we use in daily life is not just in one of two states. For example, our numbering system uses digits that can take one of ten states (0 to 9). We cannot (as yet) store this type of information in a computer: it needs to be changed to another system that uses only two states (0 and 1). We also need to be able to process other types of data (text, image, audio, video). These also cannot be stored in a computer directly, but need to be changed to the appropriate form (0s and 1s).

In Chapter 3, we will learn how to store different types of data as a binary pattern, a sequence of 0s and 1s. In Chapter 4, we show how data is manipulated, as a binary pattern, inside a computer.

Organizing data

Although data should be stored only in one form inside a computer, a binary pattern, data outside a computer can take many forms. In addition, computers (and the notion of data processing) have created a new field of study known as *data organization*, which asks the question: can we organize our data into different entities and formats before storing them inside a computer? Today, data is not treated as a flat sequence of information. Instead, data is organized into small units, small units are organized into larger units, and so on. We will look at data from this point of view in Chapters 11–14.

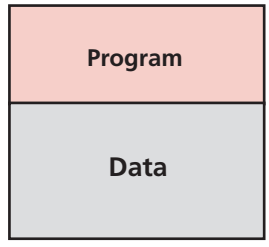
1.3.3 Computer software

The main feature of the Turing or von Neumann models is the concept of the *program*. Although early computers did not store the program in the computer's memory, they did use the concept of programs. *Programming* those early computers meant changing the wiring systems or turning a set of switches on or off. Programming was therefore a task done by an operator or engineer before the actual data processing began.

Programs must be stored

In the von Neumann model programs are stored in the computer's memory. Not only do we need memory to hold data, but we also need memory to hold the program (Figure 1.6).

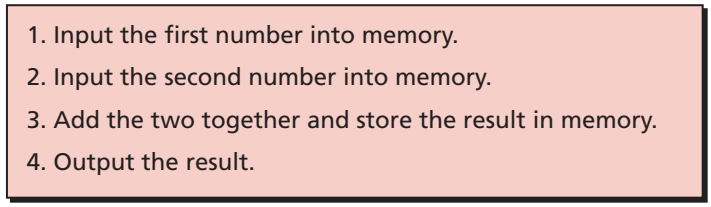
Figure 1.6 *Program and data in memory*



A sequence of instructions

Another requirement of the model is that the program must consist of a sequence of instructions. Each instruction operates on one or more data items. Thus, an instruction can change the effect of a previous instruction. For example, Figure 1.7 shows a program that inputs two numbers, adds them, and prints the result. This program consists of four individual instructions.

Figure 1.7 *A program made of instructions*



We might ask why a program must be composed of instructions. The answer is reusability. Today, computers do millions of tasks. If the program for each task was an independent entity without anything in common with other programs, programming would be difficult. The Turing and von Neumann models make programming easier by defining the different instructions that can be used by computers. A programmer can then combine these instructions to make any number of programs. Each program can be a different combination of different instructions.

Algorithms

The requirement for a program to consist of a sequence of instructions made programming possible, but it brought another dimension to using a computer. A programmer must not only learn the task performed by each instruction, but also learn how to combine these instructions to do a particular task. Looking at this issue differently, a programmer must first solve the problem in a step-by-step manner, then try to find the appropriate instruction (or series of instructions) to implement those steps. This step-by-step solution is called an **algorithm**. Algorithms play a very important role in computer science and are discussed in Chapter 8.

Languages

At the beginning of the computer age there was only one computer language, *machine language*. Programmers wrote instructions (using binary patterns) to solve a problem. However, as programs became larger, writing long programs using these patterns became tedious. Computer scientists came up with the idea of using symbols to represent binary patterns, just as people use symbols (words) for commands in daily life. Of course, the symbols used in daily life are different from those used in computers. So the concept of **computer languages** was born. A natural language such as English is rich and has many rules to combine words correctly: a computer language, on the other hand, has a more limited number of symbols and also a limited number of words. We will study computer languages in Chapter 9.

Software engineering

Something that was not defined in the von Neumann model is **software engineering**, which is the design and writing of **structured programs**. Today it is not acceptable just to write a program that does a task: the program must follow strict rules and principles. We discuss these principles, collectively known as *software engineering*, in Chapter 10.

Operating systems

During the evolution of computers, scientists became aware that there was a series of instructions common to all programs. For example, instructions to tell a computer where to receive data and where to send data are needed by almost all programs. It is more efficient to write these instructions only once for the use of all programs. Thus the concept of the **operating system** emerged. An operating system originally worked as a manager to facilitate access to the computer's components by a program, although today operating systems do much more. We will learn about them in Chapter 7.

1.4 HISTORY

In this section we briefly review the history of computing and computers. We divide this history into three periods.

1.4.1 Mechanical machines (before 1930)

During this period, several computing machines were invented that bear little resemblance to the modern concept of a computer.

- In the seventeenth century, Blaise Pascal, a French mathematician and philosopher, invented Pascaline, a mechanical calculator for addition and subtraction operations. In the twentieth century, when Niklaus Wirth invented a structured programming language, he called it Pascal to honor the inventor of the first mechanical calculator.
- In the late seventeenth century, German mathematician Gottfried Leibniz invented a more sophisticated mechanical calculator that could do multiplication and division as well as addition and subtraction. It was called the Leibniz Wheel.
- The first machine that used the idea of storage and programming was the Jacquard loom, invented by Joseph-Marie Jacquard at the beginning of the nineteenth century. The loom used punched cards (like a stored program) to control the raising of the warp threads in the manufacture of textiles.
- In 1823, Charles Babbage invented the Difference Engine, which could do more than simple arithmetic operations—it could solve polynomial equations, too. Later, he invented a machine called the Analytical Engine that, to some extent, parallels the idea of modern computers. It had four components: a mill (corresponding to a modern ALU), a store (memory), an operator (control unit), and output (input/output).
- In 1890, Herman Hollerith, working at the US Census Bureau, designed and built a programmer machine that could automatically read, tally, and sort data stored on punched cards.