UNIVERSITY OF LONDON | INTERNATIONAL PROGRAMMES

Introduction to computer systems architecture and programming

J. Matravers

IS1168, 2790168

2011

Undergraduate study in
Economics, Management,
Finance and the Social Sciences

This is an extract from a subject guide for an undergraduate course offered as part of the University of London International Programmes in Economics, Management, Finance and the Social Sciences. Materials for these programmes are developed by academics at the London School of Economics and Political Science (LSE).

For more information, see: www.londoninternational.ac.uk

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

This guide was prepared for the University of London International Programmes by:

J. Matravers, PhD.

This is one of a series of subject guides published by the University. We regret that due to pressure of work the author is unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

University of London International Programmes
Publications Office
Stewart House
32 Russell Square
London WC1B 5DN

Website: www.londoninternational.ac.uk

Published by: University of London

Contents

Introduction

Introduction to computer systems architecture and programming is a '100' course offered on the Economics, Management, Finance and the Social Sciences (EMFSS) suite of programmes.

The computer has become an integral part of our lives. Apart from the computer you use to write your coursework and to communicate with friends, there is the computer technology embedded in your coffee maker that detects how hot to brew your coffee, in your mobile telephone, in the ticket reader that deducts your bus fare directly from your bus pass, in the ATM (automatic teller machine) that disposes your money for the week, etc. The list is huge and is getting longer each day.

However, most of the users of these technologies have little or no knowledge of the history of this phenomenal development and little understanding of how a computer works. For most, the computer remains a black box that magically runs software applications. In this course you will learn how a computer's architecture provides for the computer services we have become so accustomed to using.

Also, those of you who are studying for the **BSc Information systems and management** will be aware that the systems you are studying rely heavily on computer technology, and some insight into this technology is vital, especially if your future career requires you to manage information system development projects that involve the production of software.

This course provides you with an insight into how computers operate. It will do this from two perspectives. It will first explore the area of computer architecture (which means it will look at the underlying components of a computer and the way they carry out processes and instructions). It will also look at the ways computers can interact with each other in a network.

Once you are familiar with the basic way a computer operates, you will learn how you can use programming code to instruct a computer (in other words, how you can write your own software). You will be introduced to the fundamental concepts of programming using Java as an example.

In brief, this course will not only equip you with the skills to develop basic Java applications, it will also introduce you to the areas of computer science that are essential for you to understand the internal processing during program execution.

Aims and objectives

This course presents an up-to-date introduction to computer science and programming. It introduces the foundations of computer architecture together with data representation, manipulation and storage. The use of algorithms for problem-solving is introduced. The course further introduces the concepts of operating systems and computer networks. Against these concepts fundamental programming methods, constructs and concerns will be introduced using the Java programming language.

The objectives of the course are to:

- develop an understanding of the fundamentals of hardware and software technologies that underlie contemporary computer-based information systems

- develop an understanding of the underlying structure and theories of computers and programming
- provide the skills needed to develop algorithms for programming solutions
- provide the skills needed to write simple programs in Java.

Learning outcomes

At the end of the course, and having completed the Essential reading and activities, you should be able to:

- identify the basic elements of hardware and explain their functions and how they fit together to form an architecture
- explain how data is represented, manipulated and stored within a computer system
- identify and explain the functions of operating systems
- explain how computers interact through local and wide area networks
- identify various different types of programming languages and appreciate how they have evolved since the early days of computer programming
- design algorithms to solve basic programming problems
- explain common data types and structures
- explain basic programming structures
- explain the underlying concepts of object-oriented programming
- write simple but effective programs in Java.

How to use this subject guide

The aim of this subject guide is to help you to interpret the syllabus. It outlines what you are expected to know for each area of the syllabus and suggests relevant readings to help you to understand the material.

Throughout this guide I will point you towards reading in two main textbooks. I would recommend that you work through the guide in chapter order, reading the essential texts in parallel. You will find that much of the information you need to learn and understand can also be found on the internet and in other texts, some of which are listed below. I find that different students prefer different writing styles and understand certain texts better than others. Although you must read the Essential reading, I would encourage you to look at other sources, too, where you might find a slightly different explanation/discussion of the same topic helpful.

Each chapter of this subject guide also provides you with examples and activities. It is important that you go through the examples carefully and make sure you have understood them fully. The activities are also an important part of this course. They ensure that you practise the material covered and help you to check whether you have understood important parts of the topic. The examination paper will be set based on the assumption that you have completed all of the activities in this guide.

Having said this, it is important that you appreciate that different topics are not self-contained. There is a degree of overlap between them and you are guided in this respect by the cross-referencing between different chapters. In terms of studying this subject, the chapters of this guide are designed as self-contained courses of study, but for examination purposes you need to have an understanding of the subject as a whole.

At the beginning of each chapter you will find a checklist of your learning outcomes, which is a list of the main points that you should understand once you have covered the material in the guide and the associated readings.

Structure of the guide

This subject guide is divided into two parts. **Part 1** addresses the area of computer architecture and organisation. It also explores how computers communicate with each other through networks. **Part 2** provides an introduction to problem-solving with a computer, algorithms and programming in Java. Accordingly, the guide is structured as follows.

Part 1

Chapter 1 introduces the fundamental computer components and explains how a computer operates.

Chapter 2 explains how data needs to be represented to be stored and processed within a computer system.

Chapter 3 explains how a computer processes data.

Chapter 4 focuses specifically on the role of the operating system.

Chapter 5 serves as a foundation to understanding how computer networks facilitate the communication between computers and other devices.

Part 2

Chapter 6 provides an overview of how programming languages have evolved over the years and introduces the concept of an algorithm.

Chapter 7 introduces the most fundamental features of Java programming. It also helps you to set up your computer to write and run your first Java application.

Chapter 8 introduces essential Java program components as a basis for interactive programs that can perform basic operations.

Chapter 9 provides an introduction to object orientation as a basis for Java programming.

Chapter 10 focuses on decision structures and iteration (loops) in Java.

Chapter 11 focuses on the use of arrays in Java.

Essential reading

You should purchase:

Brookshear, J.G. *Computer Science: An Overview*. (Boston, Mass.: Pearson, 2009) tenth edition [ISBN 9780321544285 (pbk)].[1]

The suggested reading for the Java component of this guide is:

Carrano, F.M. *Imagine! Java: Programming Concepts in Context*. (Boston, Mass.: Pearson, 2011) [ISBN 9780131377158].

Part 2 of this guide introduces you to programming in Java. Please note that numerous books have been written on Java, and you will find the topics introduced in this guide in any standard Java introductory text. Some of these texts are listed in the Further reading section below. You are encouraged to consult several sources on the same topic, because it may well be the case that you will find an author whose writing and explanation style suit you better.

[1] *At the time of publication, there is a new edition of Brookshear due out; details currently available are as follows: Brookshear,* J. Computer Science: An Overview. *11th International edition (with companion website access card) (Pearson, 2011) [ISBN 9780273760238]. Students ahould check the VLE for updates.*

Please note that each chapter of the subject guide commences by identifying the appropriate chapters from these two textbooks. In instances where these textbooks are inadequate or simply do not cover a particular topic, additional or supplementary reading is recommended.

There is one further text which addresses all of the topics in this course (Parts 1 and 2), although not always in as much detail as you might like. Still, it is not an expensive text, and it is recommended that you add it to your personal Essential reading:

Reynolds, C. and P. Tymann *Schaum's Outline of Principles of Computer Science* (Schaum's Outline Series). (New York: McGraw-Hill, 2008) [ISBN 9780071460514].

Detailed reading references in this subject guide refer to the editions of the set textbooks listed above. New editions of one or more of these textbooks may have been published by the time you study this course. You can use a more recent edition of any of the books; use the detailed chapter and section headings and the index to identify relevant readings. Also check the virtual learning environment (VLE) regularly for updated guidance on readings.

Further reading

Please note that as long as you read the Essential reading you are then free to read around the subject area in any text, paper or online resource. You will need to support your learning by reading as widely as possible and by thinking about how these principles apply in the real world. To help you read extensively, you have free access to the VLE and University of London Online Library (see below).

For more detail on the topics of this course I would recommend the texts below.

Computer architecture/organisation

Comer, D.E. *Computer Networks and Internets*. (Upper Saddle River, N.J.: Prentice Hall, 2009) fifth edition [ISBN 9780136061274].

Leiden, C. and M. Wilensky *TCP/IP For Dummies*. (Hoboken, N.J.: John Wiley & Sons, 2009) sixth edition [ISBN 9780470450604].

Patterson, D.A. and J.L. Hennessy *Computer Organization and Design: the Hardware/software Interface*. (Burlington, Mass.: Elsevier Morgan Kaufmann, 2008) fourth edition [ISBN 9780123744937 (pbk)].

Stallings, W. *Computer Organization and Architecture, Designing for Performance*. (Boston, Mass.: Pearson, 2010) [ISBN 9780135064177].

Tanenbaum, A. *Structured Computer Organization*. (Upper Saddle River, N.J.: Pearson Prentice Hall, 2010) fifth edition [ISBN: 9780135094051(pbk)].

Tanenbaum, A. *Computer Networks*. (Upper Saddle River, N.J.: Pearson, 2011) fifth international edition [ISBN 9780132553179].

Programming in Java

Bell, D. and M. Parr *Java for Students*. (Harlow: Prentice Hall, 2010) sixth edition [ISBN 9780273731221(pbk)].

Deitel, H.M. and P.J. Deitel *Java: How to Program*. (Upper Saddle River, N.J.; London: Pearson, 2010) eighth international edition [ISBN 9780131364837 (pbk)].

Downey, A.B. *How to Think Like a Computer Scientist, Java Version*, Version 4.1 (Free Software Foundation, 2008) http://greenteapress.com/thinkapjava/thinkapjava.pdf

Horstmann, C.S. *Java Concepts*. (Hoboken, N.J.: John Wiley & Sons, 2010) sixth edition [ISBN 9780470509470 (pbk)].

Liang, Y.D. *Introduction to Java Programming*. (Harlow: Pearson, 2010) brief version, eighth edition [ISBN 9780132473118 (pbk)].

Journals

The topics introduced as part of this subject touch upon many areas of computer science, and there are many journals that discuss current research issues. If you are interested in leading-edge research you might want to look at the following sources.

www.acm.org/

The Association for Computing Machinery (ACM) is known as one of the largest international computing societies. It offers a wide range of publications in all areas of computer science.

www.computer.org/

As another leading computer society the Institute of Electrical and Electronics Engineers (IEEE) Computer Society offers a wide range of publications that are relevant to the subject matter introduced here. You will find when you study for this subject that the IEEE also plays a dominant role in the setting of standards in the field of computer science.

Web resources

www.howstuffworks.com/

This website gives introductory explanations of numerous technologies and might be helpful as an easy-to-understand introduction to computer technologies.

www.wired.com/

This website provides you with daily news on the latest (computer) technologies.

http://foldoc.org/

The Free Online Dictionary of Computing provides brief introductory definitions of terms.

www.computerhistory.org/

This website gives you a good historical overview with illustrative photographs of the computer as it evolved from its early beginnings.

http://download.oracle.com/javase/

In the second part of this subject guide you learn how to program in Java. The first public version of Java was released by Sun Microsoft. Sun Microsoft was acquired by Oracle in early 2010. A main source of software downloads, tutorials and information on Java will, therefore, be the Oracle website.

www.wikipedia.org/

Finally, you might want to use Wikipedia to help you clarify a certain concept or terminology. Keep in mind, however, that this encyclopaedia is written by the people who use it, and that you should remain critical of the material you read. You should not use Wikipedia as a replacement for textbooks or peer-reviewed journal articles.

Unless otherwise stated, all websites in this subject guide were accessed in April 2011. We cannot guarantee, however, that they will stay current and you may need to perform an internet search to find the relevant pages.

Online study resources

In addition to the subject guide and the Essential reading, it is crucial that you take advantage of the study resources that are available online for this course, including the VLE and the Online Library.

You can access the VLE, the Online Library and your University of London email account via the Student Portal at:
http://my.londoninternational.ac.uk

You should have received your login details for the Student Portal with your official offer, which was emailed to the address that you gave on your application form. You have probably already logged in to the Student Portal in order to register! As soon as you registered, you will automatically have been granted access to the VLE, Online Library and your fully functional University of London email account.

If you forget your login details at any point, please email uolia.support@london.ac.uk quoting your student number.

The VLE

The VLE, which complements this subject guide, has been designed to enhance your learning experience, providing additional support and a sense of community. It forms an important part of your study experience with the University of London and you should access it regularly.

The VLE provides a range of resources for EMFSS courses.

- Self-testing activities: Doing these allows you to test your own understanding of subject material.
- Electronic study materials: The printed materials that you receive from the University of London are available to download, including updated reading lists and references.
- Past examination papers and *Examiners' commentaries*: These provide advice on how each examination question might best be answered.
- A student discussion forum: This is an open space for you to discuss interests and experiences, seek support from your peers, work collaboratively to solve problems and discuss subject material.
- Videos: There are recorded academic introductions to the subject, interviews and debates and, for some courses, audio-visual tutorials and conclusions.
- Recorded lectures: For some courses, where appropriate, the sessions from previous years' study weekends have been recorded and made available.
- Study skills: Expert advice on preparing for examinations and developing your digital literacy skills.
- Feedback forms.

Some of these resources are available for certain courses only, but we are expanding our provision all the time and you should check the VLE regularly for updates.

Making use of the Online Library

The Online Library contains a huge array of journal articles and other resources to help you read widely and extensively.

To access the majority of resources via the Online Library you will either need to use your University of London Student Portal login details, or you

will be required to register and use an Athens login:
http://tinyurl.com/ollathens

The easiest way to locate relevant content and journal articles in the Online Library is to use the **Summon** search engine.

If you are having trouble finding an article listed in a reading list, try removing any punctuation from the title, such as single quotation marks, question marks and colons.

For further advice, please see the online help pages:
www.external.shl.lon.ac.uk/summon/about.php

Examination structure

Important: the information and advice given here are based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you to always check both the current *Regulations* for relevant information about the examination, and the VLE where you should be advised of any forthcoming changes. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

Remember, it is important to check the VLE for:

- up-to-date information on examination and assessment arrangements for this course
- where available, past examination papers and *Examiners' commentaries* for the course which give advice on how each question might best be answered.

The examination paper for this course is three hours in duration. The paper will be divided into **two** sections comprising four questions each. You are expected to answer two questions from each section (i.e. a total of four questions). You should allow an equal amount of time for each question and attempt all parts or aspects of a question.

The Examiner attempts to ensure that all of the topics covered in the syllabus and subject guide are examined. Some questions could cover more than one topic from the syllabus since the different topics are not self-contained. A Sample examination paper appears as an appendix to this guide.

Examination advice

Before you start to answer an examination question you should take a few minutes to read through all of the questions carefully. You can then choose the four questions you think you will answer best. You must make sure that you choose two questions from each section. You will not get any credit for answering more than two questions in each section.

There is a tendency for students to pick out a key phrase that they recognise and then to write everything that they associate with this phrase. You must make sure that the answer presented actually addresses the question posed. Also, you must write your answers clearly and legibly. Your answer has to demonstrate that you have understood the material.

You may be asked to provide one or more examples to illustrate your argument. Keep in mind that this should be viewed as an opportunity for you to demonstrate that you have understood and can apply the material learnt.

In the examination you may be asked to write Java program code. Such a question requires you to demonstrate that you can create a Java program by identifying the correct algorithm and knowing the relevant Java programming concepts. It is not a test that checks whether the syntax of your Java programming code is perfect.

However, keep in mind that proper programming practice is essential for you to establish the ability to understand how algorithms are developed and implemented in Java and hence become a proficient programmer. The examination will be based on the assumption that you have done all the activities in this subject guide.

Also, it is essential that you include comments in which you explain why you have chosen certain programming constructs and how they fulfil a particular task.

Each of the examination questions carries equal marks, and the available time should be divided between the questions accordingly. If a question consists of sub-parts, the marks allocated to each part should be reflected in the amount of time and effort that is put into answering the sub-part of the question.

Recommended study time

This subject guide is divided into several chapters according to the topics that are addressed. Please note that the chapters are of various lengths, but that the length of a chapter does not necessarily indicate the amount of time you should spend on the topic. Also, it is not the case that each chapter requires exactly the same time for reading and study.

You will find that studying for Part 1 of the guide is quite different from the work you will have to do in Part 2. Part 2 is a lot more practical, and it is essential that you apply the theory that is presented by writing programming code. The activities will help you to do so, and if you struggle the first time you do an activity, you should repeat it until you feel competent.

It is impossible to give an exact amount of time you should spend on each chapter, especially because different students require different amounts of time to study a particular topic. The time to stop studying a particular topic is when you know the topic thoroughly.

It is essential that you plan your time carefully. Ensure that you allocate time every week, and stick to your schedule right from the beginning and throughout the year, so that your studies are a continuous process. Although you should allow for some additional time for the revision in the weeks leading up to the examination, you will not be successful if you leave the majority of your work until the last few weeks of the year. For the Java programming in particular, you will need continuous practice.

A **very rough indication** of how an average student might divide his/her time between chapters is given below, but please keep in mind that this is merely a vague guideline, and that you will have to distribute your time according to your abilities.

Chapter	Approximate percentage of your time
1	5%
2	10%
3	5%
4	10%
5	10%
6	5%
7	5%
8	15%
9	15%
10	15%
11	5%

Syllabus

The following topics are covered.

Computer architecture and organisation

- The origins of computer science
- Elements of a computer
- Von Neumann architecture
- Data representation
- The binary system.

Operating systems

- Operating system architecture
- Memory management
- Process scheduling
- Semaphores and deadlocks.

Networking

- Network fundamentals
- The TCP/IP reference model
- Internet protocols
- The world wide web.

Problem-solving and programming concepts

- Programming language generations
- Algorithms and pseudocode
- The object-oriented programming paradigm.

Introducing programming with Java

- Structure and components of a Java program
- Input and output
- Objects, attributes, methods
- Arithmetic and Boolean expressions
- Variables and constants, data types
- Pre-defined Java classes
- Control structures
- Arrays.

Part 1

Notes

Chapter 1: Introduction to computer systems architecture

Aim of the chapter

This chapter gives you an overview of what constitute the main components of computer systems architecture. It also provides you with an historical overview of how the idea of the computer first emerged and how it has evolved from its early beginnings to the powerful technologies we have available today.

Learning outcomes

After completing this chapter, and the Essential reading and activities, you should be able to:

- describe the central ideas underlying the discipline of computer architecture and organisation
- outline the advancement of computer technology over the past few decades
- explain the basic structure and functioning of a computer.

Essential reading

Brookshear, J.G. *Computer Science: An Overview*. (Boston, Mass.: Pearson, 2009) Chapter 0, section 0.2; Chapter 2, section 2.1.

Reynolds, C. and P. Tymann *Schaum's Outline of Principles of Computer Science* (Schaum's Outline Series). (New York: McGraw-Hill, 2008) Chapter 1.

Further reading

ACM and IEEE Computer Society, *Computer Science Curriculum 2008: An Interim Revision of CS 2001*, December 2008, www.acm.org//education/curricula/ComputerScience2008.pdf

Stallings, W. *Computer Organization and Architecture, Designing for Performance*. (Boston, Mass.: Pearson, 2010).

Tanenbaum, A. *Structured Computer Organization*. (Upper Saddle River, N.J.: Pearson Prentice Hall, 2010).

Websites

Computer History Museum: www.computerhistory.org

Information on John Napier: www.johnnapier.com/

Video clip on the ENIAC: http://news.cnet.com/1606-2_3-29770.html

Bletchley Park: www.bletchleypark.org.uk/

Kopplin, John *An Illustrated History of Computers*, 2002: www.computersciencelab.com/ComputerHistory/History.htm

Introduction

The aim of this course is to provide you with an insight into how a computer works. We know that computers are devices that carry out the instructions they are given. Part 2 of this course teaches you fundamental

programming skills, and you may argue that as long as you understand these fundamentals you will be able to instruct a computer. However, this course offers much more than programming skills. It also provides you with the insight into how the computer processes the instructions it receives in the form of program code. As such this course provides a basis for the ability to optimise both the way a computer system is programmed and the way it is designed.

To this effect, Chapter 1 first presents some historical information for you to get an overview of how the first idea of the computer came about and how it evolved into the kind of computer we use today. This chapter then continues to provide an introduction into the fundamental components of a computer's architecture.

An historical overview of computer science

This section provides an overview of the historical development of the computer. In no way does it claim to be exhaustive, but it offers some initial insights into the main stages of the development of computer structure and computer processes.

Early contributions

The date of the use of the first computer is open to debate as it depends on how we define a computer. If we base this overview on the search for the first apparatus that was used within some form of calculation, we are taken back to at least the ancient Greeks and Romans who are said to have used the **abacus** to help them with additions and subtractions (Brookshear 2009). However, the abacus is simply a memory support. The position of the beads merely helps the user to remember (or store) values used in the calculation.

Since the invention of the abacus, the work of the Scottish mathematician and philosopher **John Napier** (1550–1617) is often viewed as a significant contribution towards the development of increasingly sophisticated calculation devices. John Napier invented the idea of logarithms (remember from your mathematics lessons at school: if $x = y^z$ then $\log_y x = z$) and used this concept to develop a device (known as **Napier's bones**) that managed to reduce the complexity of multiplication and division into the more simple operations of addition and subtraction. He did this by taking advantage of the fact that if a number is expressed in exponential form, multiplication, for example, can be carried out by adding the exponents (for example, $10^2 \times 10^4 = 10^{(2+4)}$, which is a simplified calculation of $100 \times 10{,}000$). For more detail see www.johnnapier.com/

The first mechanical computing machines

The development of the first **mechanical** computing machine is generally considered to be the next major step in the advancement of computing devices, and it is the French mathematician and philosopher **Blaise Pascal** (1623–1662) who is credited with this invention. His calculator, also known as the **Pascaline**, was able to add or subtract two decimal numbers with the help of mechanical gears. The Pascaline was later developed further by the German mathematician and philosopher **Gottfried Wilhelm von Leibniz** (1646–1716), who added multiplication and division to this calculator.

The English mathematician **Charles Babbage** (1791–1871) is known for the development of his **Difference Engine** and the **Analytical Engine**. In his Difference Engine Babbage attempted to create a machine (the size

of a small room!) that was able to perform calculations without having to attempt multiplication and division. Unfortunately, the project (funded by the British government) could not be completed before funding ran out. Still, plans and components of the Difference Engine have survived, and Babbage continued to explore his ideas in the development of the Analytical Engine. The Analytical Engine is the first machine that was able to receive instructions in the form of holes in paper cards (punch cards) and hence is considered as the first **programmable** machine. Also, the Analytical Engine was the first machine to support conditional program execution (i.e. whether some particular part of the program code was executed could be made dependent on whether during execution some condition was found to be true; you will learn more about conditional statements and their significance in Part 2 of this guide).

Relays and vacuum tubes

Although the underlying (mechanical) technologies of the machines above were developed further in the early 1900s, it was hard to develop such machines in a financially viable manner. However, advances in the field of electronics soon opened up new possibilities with the development of **relays** and **vacuum tubes** (see Brookshear 2009).

A relay is a mechanical switch that is controlled by electromagnets. A vacuum tube is an electronic device that can also be used as a switch. Both switches are used to switch electrical currents on and off. Relays have the advantage of being reliable and relatively cheap. Vacuum tubes have the advantage of being much faster than relays, but they consume much more energy and have a high heat emission.

Famous examples of the first computers that took advantage of these advances include the **Z3**, the **Mark 1**, the **Colossus** and the **ENIAC** (Kopplin 2002). They used circuits that contained relays or vacuum tubes, and punched cards were used for input and for storage.

Konrad Zuse is well known for the development of his **Z3** computer (after a couple of earlier models) using electronically controlled mechanical relays. Z3 was finished in Nazi Germany in complete isolation in 1941, and as a consequence engineers in the US and UK remained unaware of Zuse's work.

The **Mark 1** computer, which was developed at Harvard University in partnership with IBM in 1944, is known as another early famous success of a programmable machine that used electronically controlled relays. With a length of 16 m and a height 2.4 m, the Mark 1 was enormous in size. It was initially used by the Navy and later in other fields to solve repetitive calculations. It was operational for 15 years (see www-03.ibm.com/ibm/history/). (Note: The Mark 2 is known for the first use of the term 'computer bug'. The unexpected behaviour of the machine was traced down to a moth which had been trapped in a relay. This moth was the first computer bug! To see a photograph of its recording visit: http://commons.wikimedia.org/wiki/File:H96566k.jpg)

The use of vacuum tubes was the next step in the advances of computer technology. The **Colossus** computer and the **ENIAC** (Electronic Numerical Integrator and Computer) are generally quoted as the first computers of this era and hence as the first **electronic** computers. The Colossus was used at Bletchley Park to support the decoding of German messages during the Second World War (www.bletchleypark.org.uk).

The ENIAC was built by the University of Pennsylvania to address the needs of the Army's Ballistics Research Laboratory during the Second

World War for the accurate calculation of the firing tables used to aim their artillery. However, the ENIAC was only completed in 1946, and as a consequence its first task was to help with some complex calculations that were required to assess the feasibility of the hydrogen bomb. Hence the ENIAC is often described as the first **general-purpose electronic** computer and is often chosen as the main representative of vacuum tube computers or as what we now refer to as **first-generation computers** (Stallings 2010). For an interesting video clip on the ENIAC visit http://news.cnet.com/1606-2_3-29770.html

From transistors to integrated circuits

Since the development of first-generation computers, the further development of the computer has been pushed forward by advances in technology. **Second-generation computers** emerged with the development of the **transistor** in the 1950s. The transistor replaced the vacuum tube. Made from silicon, major advantages of the transistor included its smaller size, lower price and lower heat emission.

The invention of the **integrated circuit** in 1958 marks the era of **third-generation computers**. Rather than having to treat components such as transistors, resistors and conductors as separate components that have to be connected, the advances in micro-electronics allowed the production of an entire circuit from one tiny piece of silicon.

Beyond the third generation, there are varying views of how or whether computer hardware advancement can be divided into further generations, but some attempts have been made based on the advances of integrated circuit technology (Stallings 2010).

A major milestone in the development of computers since the 1940s includes the development of the first **desktop computers**. Several (successful) attempts were made by computer companies and individuals. However, the most famous remains the development of the **PC** (**personal computer**), which was first developed by IBM in 1981 with software by Microsoft. We still use the term PC today to refer to any computer, from any manufacturer, that has evolved from IBM's original desktop computer (Brookshear 2009).

Since the early beginnings of the desktop computer as a tool that was useful in disciplines beyond mathematical calculations, and accessible to a more general user group, computer technology has penetrated all aspects of life. Today most of us use a PC on a daily basis, and computer technology is integrated in our telephones, entertainment technologies, our cars, etc.

In order to gain a better understanding of how computer technology works, the remainder of this chapter introduces you to the areas of computer architecture that are essential for you to understand the internal processes of a computer.

Reading

Now read Brookshear (2009) Chapter 0, section 0.2 and Reynolds and Tymann (2008) Chapter 1.

Activities

1. On the internet find pictures and/or video clips of the early computing machines mentioned in this chapter (and others). Get a sense of their size, the materials/components that were used to build them, the way they were operated and how they have evolved over the past few centuries. You may want to look at the sources below,

but you are also encouraged to find others yourself.

Kopplin, John, *An Illustrated History of Computers*, 2002, www.computersciencelab.com/ComputerHistory/History.htm

The website of Bletchley Park: www.bletchleypark.org.uk/content/machines.rhtm

2. Among other things, Pascal is famous for the idea of using gears in his Pascaline. Data was represented through 'gear positioning'. Explain what this means.
3. Name two inventions that have had a significant influence on the development of the computer as we know it today and explain their significance.

What is computer architecture?

Since the term 'computer architecture' constitutes part of the title of this subject guide it deserves some attention – if only to clarify its meaning in the context of this text.

If you look through the computer science literature it seems impossible to find a universally accepted definition of the term computer architecture. Experts often disagree about the exact differentiation between the terms **computer architecture** and **computer organisation** (Stallings 2010).

However, the Association for Computing Machinery (ACM) in conjunction with other computer societies and professionals in the field of computer science has proposed curriculum recommendations for computer science that also include the subject of computer architecture (see www.acm.org/education/curricula/ComputerScience2008.pdf). Although a clear distinction between computer architecture and computer organisation is not made, a list of topics that ought to be covered in an introductory course on 'Architecture and organisation' is suggested. Part of the curriculum reads:

> In this introduction the term 'architecture' is taken to include instruction set architecture (the programmer's abstraction of a computer), organization or microarchitecture (the internal implementation of a computer at the register and functional unit level), and system architecture (the organization of the computer at the cache, and bus level).

Based on this proposal, this text uses the term 'computer architecture' to encompass all aspects of a computer you should know about in order to understand how a computer executes a program. Hence, here computer architecture includes the following areas:

- the fundamental physical components that constitute a computer system (the hardware)
- the kind of instructions/language the computer can understand
- the underlying computer technology that manipulates these instructions (sometimes referred to as microarchitecture).

However, please be aware that although this subject guide draws these areas of research together under the heading of 'computer architecture', other authors may treat (part of) some of these areas under the heading of 'computer organisation'. So in your reading look out for relevant materials under both headings: computer architecture **and** computer organisation.

The Von Neumann architecture

Let us now first look at the general structure of a computer system. The Von Neumann architecture describes the structure on which most of today's computers are built, and its original idea is associated with the ENIAC computer discussed above. Although the ENIAC computer could be programmed, entering programs or changing them was a difficult procedure as it needed to be programmed manually by turning switches and plugging and unplugging cables. The mathematician **John von Neumann** (1903–1957) is generally credited with the idea of storing the program code together with the stored data to overcome the inconveniences of 'external' programming.

In 1945 the concept of the 'stored program' was first suggested as the basis for the design of the **EDVAC** (Electronic Discrete Variable Computer), and in 1946 the **IAS** computer was developed at the Princeton Institute for Advanced Studies. This machine was only completed in 1952, but its structure is still considered to be the original basis for today's computers' architecture. You might think this is obvious: of course programs are stored with the 'remaining' data. However, try to understand that in the early days of computing a strict distinction was made between the data that was stored and the 'processing methods' (programs) that were applied to manipulate the data. The realisation that instructions could be encoded and stored just like 'ordinary' data was a major innovation (Brookshear 2009).

Computers based on the **stored program** concept or the **Von Neumann architecture** store both their program code (i.e. instructions) and the data that is required for (or may result from) the computation in the computer's memory. During computation, program instructions are retrieved from the memory and executed one after the other. The component in a computer that controls this computation is known as the **central processing unit** (CPU, or nowadays often just referred to as the processor). The CPU consists of:

- the arithmetic logic unit, which performs operations, such as addition and subtraction, on the data
- the control unit, which coordinates the computer's activities
- the registers, which are data storage cells that are used for the temporary storage of data; for example, the data that is required in a calculation that is carried out.

The connection between the CPU and the memory is known as a bus. Graphical representations of this fundamental underlying computer architecture can be found in most computer science textbooks (e.g. Brookshear 2009, Figure. 2.1).

Note that the underlying idea here is that the Von Neumann architecture presents a **logical** description of the stored program computer. The exact way in which this architecture is implemented was not Von Neumann's concern. The only thing that was relevant was that the technology used met his functional specification.

The stored program concept also has its shortcomings. The separation of CPU and the memory that stores both data and program code required by the CPU means that the bus struggles to provide all the data fast enough to take advantage of the full capacity/speed at which today's CPUs can operate. Both CPU speed and computer memory size have increased dramatically over the past, and the associated demand for ever faster data

traffic between CPU and memory has caused the Von Neumann bottleneck to remain a main concern among computer engineers.

Reading

Now read Brookshear (2009) Chapter 2 section 2.1.

Activities

1. Find out what cache memory is and explain how it may alleviate the Von Neumann bottleneck.
2. The Harvard architecture is often discussed as an alternative to the Von Neumann architecture. With the help of the literature available to you, find out what the main components of the Harvard architecture are and compare them with the Von Neumann architecture.

A reminder of your learning outcomes

After completing this chapter, and the Essential reading and activities, you should be able to:

- describe the central ideas underlying the discipline of computer architecture and organisation
- outline the advancement of computer technology over the past few decades
- explain the basic structure and functioning of a computer.

Sample examination question

1. The Von Neumann architecture presents a logical view of a computer that still forms the basis of most computers as we know them today.
   a. Explain what the von Neumann architecture is and, with the help of an example, explain how it processes data. (10 marks)
   b. Apart from the above, there have been many other mathematical and technical breakthroughs that have had an impact on the way computers have advanced since the early twentieth century. Write a short essay in which you specify the two advances that – in your view – have been most significant and argue why they have been so important. (15 marks)

Notes

Chapter 2: Data representation

Aim of the chapter

This chapter explains how data can be represented in binary form so it can be stored in and processed by a computer.

Learning outcomes

After completing this chapter, and the Essential reading and activities, you should be able to:

- explain how textual and numeric information can be represented in binary form
- perform basic calculations within the binary system
- explain the fundamental operations within Boolean logic and how logic gates can be used to perform these operations within a digital circuit.

Essential reading

Brookshear, J.G. *Computer Science: An Overview*. (Boston, Mass.: Pearson, 2009) Chapter 1, sections 1.1–1.2, 1.4–1.6 and 1.8.

Reynolds, C. and P. Tymann *Schaum's Outline of Principles of Computer Science* (Schaum's Outline Series). (New York: McGraw-Hill, 2008) relevant sections of Chapter 3.

Further reading

Patterson, D.A. and J.L. Hennessy *Computer Organization and Design: the Hardware/software Interface.* (Burlington, Mass.: Elsevier Morgan Kaufmann, 2008).

Stallings, W. *Computer Organization and Architecture, Designing for Performance*. (Boston, Mass.: Pearson, 2010).

Tanenbaum, A. *Structured Computer Organization*. (Upper Saddle River, N.J.: Pearson Prentice Hall, 2010).

Websites

Some information about George Boole: www.kerryr.net/pioneers/gallery/boole.htm

A helpful introduction into the binary system: Wright, Christine R. and Samuel A. Rebelsky *The binary system*: www.cs.grinnell.edu/~rebelsky/Courses/CS152/97F/Readings/student-binary.html

Useful information and illustrations of Boolean logic and logic gates can be found at a website by Ken Bigelow: www.play-hookey.com/digital/

Introduction

The previous chapter has given you an overview of how a computer operates. It has shown that, in order to operate, the computer needs to store and process data. This chapter is concerned with the way data needs to be represented so it can be stored in a computer's memory and processed as part of computer operations.

Bits and bytes

At the most fundamental (and slightly simplified) level, today's computers only recognise two states: either there is an electric current running through a circuit or not. As a consequence, all information that is represented within a computer needs to be represented with the help of these two states. Rather than referring to a state as 'running current' or 'no running current', computer science uses a 1 and a 0 for the two different possibilities. Hence the most basic unit of information is the binary digit (referred to as a bit of information). A bit of information can contain either a 1 or a 0. A collection of eight bits of information is generally referred to as one byte, and memory size is generally measured in the number bytes of information a computer can store. The memory size of our early computers was measured in 2^{10} bytes. Since $2^{10} = 1024$, this was rounded down to 1000, and 1024 bytes were actually referred to (not quite correctly) as one kilobyte. Accordingly a machine with a capacity of 16 kilobytes had 16,384 bytes. We now talk about memory size in terms of:

- Megabyte: 1 megabyte = 2^{20} bytes =1,048,576 bytes
- Gigabyte: 1 gigabyte = 2^{30} bytes = 1,073,741,824 bytes.

Now that we know that all information that we feed into a computer needs to be translated into a code of 0s and 1s, i.e. into binary digits (bits), the following sections explain how text and numbers can be encoded in this way.

Representing text

A common way to represent text is to agree on a unique code for every symbol (e.g. for every letter, punctuation mark or number) that needs to be presented. Each code consists of a fixed-length sequence of bits (but obviously the sequence is different for each code). A word can then be 'written' by determining the code for each letter and stringing them together. The most well-known code that is used in this way is the **ASCII** (American Standard Code for Information Interchange). It uses 8-bit strings to represent the English alphabet. You can find an overview of the ASCII code in Appendix A of Brookshear (2009) or you can look it up on the internet. You should then be able to encode simple text, as in the following example.

01010111	01100001	01101011	01100101	00100000	01110101	01110000	00100001
W	a	k	e		u	p	!

The files we refer to as text files generally contain long sequences of code (mostly ASCII) as described above. A simple text editor can translate the code and make it readable to us (and the other way around).

The programming language Java, which you will be introduced to in Part 2 of this guide, uses the Unicode standard for the representation of characters. Unicode provides 16 bits for the representation of a character (rather than 8), and therefore the Unicode character set is much larger than the ASCII set. The Unicode character set, therefore, also includes characters from alphabets other than English (e.g. Chinese and Greek).

Note that word-processor files on the other hand are more complex. Encoding schemes such as ASCII and Unicode only address the encoding of the text. They are not suitable for the formatting information a word processor provides.

Activity

Work out the approximate memory space (in bytes) that is required to store an A4 journal that is 70 pages long.

Representing images

Images are also encoded using bit patterns. Generally an image is divided into many small picture elements (**pixels**), and the appearance of each pixel is then encoded in binary form. The collection of these encoded pixels is known as the **bitmap** of the image.

The method that is used to encode the individual pixels of a bitmap varies. A simple black and white image, for example, can be easily represented using a single bit for each pixel. More sophisticated black-and-white pictures with varying shades of grey may use as 8-bit sequences (a byte) for each pixel to represent the different shades.

A colour image usually uses three bytes to represent a single pixel. For example, computers frequently use a combination of red, green and blue light to represent a wide spectrum of colour (the **RGB colour model**). Accordingly, the colour of a pixel can be represented using three bytes, each representing the intensity of the colours red, green and blue.

Files that store a bitmap image can be rather large, and various compression methods have been developed to reduce their size. **Graphic Interchange Format** (**GIF**), for example, is one such method that reduces the size of a bitmap file by reducing the number of colours that can be assigned to a pixel to 256, and the **Joint Photographic Experts Group** (**JPEG**) developed a compression method which is commonly used to compress photographs.

Representing sound

In order to store sound on your computer, the analogue sound signal needs to be converted into a digital format. Typically the amplitude of the sound wave is checked and recorded at regular time intervals. These values can then be stored in binary form and used to re-construct the initial wave at a later stage. The sampling frequency used when recording a CD is 44,100 samples per second (44.1 kHz sample rate), and the sample data is recorded as a 16-bit sequence (32 bits for stereo recordings).

You will probably also have come across **MIDI** (**Musical Instrument Digital Interface**) files. This is an alternative approach to recording music that is frequently used in the music synthesisers used for video games or annotation on websites. MIDI files generally require less memory space, because they store music in the form of parameters that describe the music, such as, for example, the note to be played, the volume, the tempo, and so on. This usually requires less storage space than sampling at a rate of 44.1 kHz.

The **Motion Picture Expert Group** (**MPEG**) has developed various standards to compress both audio and video files. **MP3** (which is short for MPEG layer 3), for example, is such a system that was developed to compress audio.

Please note that although you are not expected to be familiar with the details of the methods used to compress image and sound files, you should be aware that these methods exist and that they are frequently used not just to reduce storage space, but also to ensure that these files can be transmitted more easily via different transmission media, for example, the internet.

Activity

If the sampling frequency for a stereo CD recording is 44,100 samples per second, how much storage space is required for a 30-minute stereo recording?

Representing numbers – the binary system

When you look at an ASCII (or Unicode) table, you will find that it also includes codes for the representation of numbers. However, this is not an ideal way of representing numbers if they are used within a calculation. Also, within ASCII we would be using 8 bits to represent a single-digit number, and the largest number we could store would be 9. However, if we represent all numbers as binary numbers (rather than decimal) we can actually represent the numbers 0 to 127 with 8 bits. We also have the advantage that a number is translated into a different numeric system (i.e. the binary system) where it remains a number and where hence mathematical calculations can still be carried out (in ASCII a number is merely a symbol in the same way as a letter is a symbol).

Binary numbers are represented using bit sequences. The representation follows the same simple principle as the decimal system. So you need to remind yourself of these underlying principles:

The decimal system uses 10 as a base, and the 10 digits available are 0, 1, 2, 3, … , 9. Depending on its position in the whole number, each digit represents the amount of 1s, 10s, 100s, 1000s, etc. (i.e. 10^0s, 10^1s, 10^2s, 10^3s, etc.). For example, the decimal number 9348 represents $9\times10^3 + 3\times10^2 + 4\times10^1 + 8\times10^0$.

The binary system uses 2 as its base with the digits 0 and 1. Let's look at 110100 as an example of a binary number (in order to avoid confusion the base of a number may be indicated using the following convention: 110100_2):

$110100_2 = 0\times2^0 + 0\times2^1 + 1\times2^2 + 0\times2^3 + 1\times2^4 + 1\times2^5 = 0 + 0 + 4 + 0 + 16 + 32 = 52_{10}$

In summary, what you need to be able to understand is that the value that is represented by a digit depends on the digit's position within the number and the base of the numeric system used. With every move of a digit to the left the value represented increases by a power of the base. For the decimal system this means digits – from right to left – need to be multiplied by 1, 10, 100, 1000, etc. For the binary system, digits – again from right (the least significant bit) to left (the most significant bit) – need to be multiplied by 1, 2, 4, 8, 16, etc. to calculate their value.

As part of the explanation above you have been shown how a binary number can be converted into a decimal number. You also need to be able to convert a decimal number into a binary number. This can be done by:

- dividing the decimal number by 2 and recording the quotient and the remainder
- continuing to do this with every resultant quotient until the quotient is 0.

The backwards sequence of the recorded remainders is then the binary representation of the original decimal number.

Let us illustrate this with our example of 52:

52 : 2 = 26 remainder 0

26 : 2 = 13 remainder 0

13 : 2 = 6 remainder 1

6 : 2 = 3 remainder 0

3 : 2 = 1 remainder 1

1 : 2 = 0 remainder 1

Hence $52_{10} = 110100_2$

You should now have an understanding of the underlying ideas of the binary system and how to convert from decimal into binary and vice versa.

Binary addition

The next step is to learn how simple mathematical calculations are carried out within the binary system. We will start with the addition of two positive binary numbers, and again it is a matter of mapping the procedure we are familiar with within the decimal system onto the binary system. An example is probably the best way to explain how it works. Let us add 52 and 49.

In the binary system we would calculate by adding the units, noting the overflow under the 10s, then adding the 10s and noting the overflow under the 100s, and finally adding the 100s (which in this case only consist of the overflow).

		100s	10s	1s
			5	2
	+		4	9
Overflow:		1	1	
Result:		1	0	1

We will now do the same addition in the binary system. We already know that $52_{10} = 110100_2$, and you should be able to work out now (Try it!) that $49_{10} = 110001_2$. We will follow the same approach as in the decimal system above. All you need to think about is what causes an overflow when adding binary digits:

0+0 = 0

1+0 = 0+1 = 1

1+1 = 10, i.e. there is an overflow of 1

1+1+1 = 11, i.e. there is an overflow of 1

		64s	32s	16s	8s	4s	2s	1s
			1	1	0	1	0	0
	+		1	1	0	0	0	1
Overflow:		1	1					
Result:		1	1	0	0	1	0	1

If you now check our result of 1100101_2 by converting it into a decimal number, you should find that $101_{10} = 1100101_2$.

Activities

1. Convert the following numbers into decimal numbers:
   - 111111_2
   - 101001_2

2. Convert the following numbers into binary numbers and add them. Convert the result back into decimal to check your result.
   - 999_{10}
   - 256_{10}

Binary subtraction

In order to subtract one binary number from another we can again use the procedures we are familiar with from the decimal system. For example: $10011_2 - 1111_2$

		64s	32s	16s	8s	4s	2s	1s
				1	0	0	1	1
	–				1	1	1	1
Borrowed:				1	1			
Result:				0	0	1	0	0

So we start from the least significant bits on the right where we deduct 1 from 1 and get a result of 0. The next column is also straightforward. In the third and fourth columns we need to borrow 1 from the column on the left.

However, it is quite difficult for a computer to carry out subtraction in this (so familiar to us) way. A more computer-friendly approach that has been proposed relies on the idea of turning the number to be subtracted into its negative counterpart (referred to as its **two's complement**) and then adding this negative using ordinary addition. (Please note that although you should be aware of the concept of representing the two's complement of a binary number, a detailed knowledge of how the two's complement can be determined is beyond the material covered in this course.)

A further well-known method that can be used to represent a negative binary number is the **sign and magnitude** method. This method follows the concept used within mathematics where negative numbers are represented by using the minus sign as a prefix. As the binary numbers used with computers do not have any extra symbols, an easy way to represent a negative number is to agree that the left-most bit is just the equivalent of a $+/-$ sign, and the remainder of the number indicates the value, i.e. the **magnitude** of the number:

- 0 indicates that the number is positive
- 1 indicates that the number is negative.

For example, in an 8-bit representation of binary numbers 00001100_2 would represent 12_{10} and 10001100_2 would be $-12_{10.}$

Although the sign and magnitude method has been used in early computers and may still be used within the representation of floating-point numbers, the advantage of the two's complement is evident: The two's complement representation allows us (or more importantly the computer) to carry out subtraction through a combination of negation and addition. This means that a computer's electronic circuits that are designed to carry out addition can also be used for subtraction. You will learn more about these circuits in the next section.

Reading

Now read Brookshear (2009) Chapter 1, sections 1.2, 1.4–1.6 and 1.8.

Activities

1. Translate the calculation below into binary format and solve it. Convert the result back into decimal to check your result. $579_{10} - 432_{10}$
2. What would 11111011 represent in the sign and magnitude representation?

Boolean logic

The sections above have established that the data that is stored and processed in a computer needs to be in binary format. You have been introduced to the binary system, you know how to translate decimal numbers into binary numbers, and you should be familiar with how the most fundamental calculations can be carried out in the binary system.

If we now look at the computer technology that processes this binary information, we need to consider the gate as the most basic building block of the computer. The gate is an electronic circuit that normally consists of a number of transistors. These transistors are arranged in such a way that the gate performs a logic operation. The logic operation that is carried out by these gates is known as **Boolean logic**, developed by the English mathematician **George Boole** (1815–1864). This section introduces you to the fundamental ideas of Boolean logic (or Boolean algebra).

Boolean logic considers two values only. These are referred to as TRUE and FALSE or 1 and 0 respectively (it does not really matter what you call them – what is important is that there are two different values only). You are familiar with the basic arithmetic operations like add, subtract, multiply and divide; Boolean logic has the following four basic operators:

- AND – results in TRUE if both operands are TRUE
- OR – results in TRUE if at least one of the operands is TRUE
- XOR – results in TRUE if one (and only one) of the operands is TRUE (exclusive OR)
- NOT – reverses the value of the operand.

The tables below (often referred to as Boolean truth tables) summarise the results of the Boolean operations for the main operators. A Boolean operator uses one or more Boolean inputs and determines one (only one) Boolean output.

AND operator		
A (input)	B (input)	C (output)
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Table 2.1: The Boolean operation AND.

OR operator		
A (input)	B (input)	C (output)
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Table 2.2: The Boolean operation OR.

XOR operator		
A (input)	B (input)	C (output)
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Table 2.3: The Boolean operation XOR.

NOT operator	
A (input)	C (output)
TRUE	FALSE
FALSE	TRUE

Table 2.4: The Boolean operation NOT.

Note: You can also think of A and B as statements that can be TRUE or FALSE. For example, A could be the TRUE statement '5 > 3'. However, these statements may also contain variables, e.g. B might be the statement 'X is a prime number' which is TRUE or FALSE depending on the value of X. You will use these kinds of statements in some of your programming techniques later.

Let us now consider the gate again. A gate has one or more inputs and produces exactly one output. Input and output can be at one of two different voltage levels, which in turn represent 1 and 0 (or TRUE and FALSE). In this way logic gates can perform logical operations; that is, you can think of them as boxes which receive multiple inputs and produce one output according to the truth tables above (Brookshear 2009). Figure 2.1 gives you the standard gate representations for the Boolean operations introduced in this chapter.

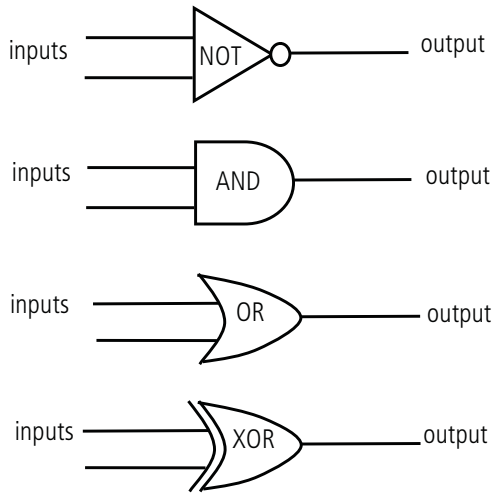

Figure 2.1: Gate symbols for Boolean operators.

Several gates (i.e. permutations of NOT, AND, OR, XOR) can then be arranged to create circuits that can perform logical and also arithmetic operations like adding two numbers. For example, if we combine an XOR gate with an AND gate as shown below, we get what is referred to as a **half adder**. The half adder adds the two binary digits A and B. Different combinations of inputs A and B result in the sum of inputs A and B and the overflow:

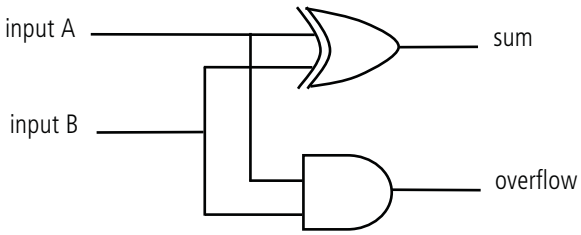

Figure 2.2: Combining the XOR operator and the AND operator to make a half adder.

Input A	Input B	Sum	Overflow
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

Table 2.5: The truth table for a half adder.

The reason why this is 'only' a half adder is that it only adds two one-digit binary numbers, but in order to add multi-digit numbers, an adder would have to be able to recognise and add an overflow of a lower-order bit addition. It is possible to extend this circuit to create a full adder.

You now know how text and numbers can be represented within a computer. You have had an introduction to digital circuit design and to the way gates can implement Boolean logic within a computer. When a computer executes a program (i.e. performs tasks like a calculation or editing some text), it needs to manipulate this data. This manipulation also involves moving data between different locations. The next chapter looks at further detail of the computer architecture to explore how such data processes may be carried out.

Reading

Now read Brookshear (2009) Chapter 1, section 1.1 and Reynolds and Tymann (2008), relevant sections of Chapter 3.

Activity

Consult the literature or the internet to work out how a full adder could be created (for example, visit www.play-hookey.com/digital/adder.html). What would the truth table for a full adder look like?

Note that the purpose of this activity is for you to gain a better understanding of the purpose and functionality of different gates and the capabilities that lie in arranging them in electric circuits. There is no need for you to learn the components of a particular circuit, like a full adder of a half adder, by heart. What is important is that you understand the underlying ideas.

A reminder of your learning outcomes

After completing this chapter, and the Essential reading and activities, you should be able to:

- explain how textual and numeric information can be represented in binary form
- perform basic calculations within the binary system
- explain the fundamental operations within Boolean logic and how logic gates can be used to perform these operations within a digital circuit.

Sample examination question

1. a. Negative binary numbers can be represented using the sign and magnitude method or the two's complement. Outline the underlying ideas of these two methods and explain their advantages and disadvantages. (10 marks)

   b. 'Gates are the fundamental building blocks from which a computer is built.'

   Explain what a gate is and in what way it is fundamental to the functioning of a computer. (15 marks)

Chapter 3: Data manipulation

Aims of the chapter

This chapter explains how a computer carries out instructions to manipulate data. A further aim of this chapter is to introduce you to the concepts underlying the communication among distinct computer components and between the computer and its peripheral devices.

Learning outcomes

After completing this chapter and the Essential reading and activities, you should be able to:

- demonstrate an understanding of the concept of using machine language to program a computer
- explain how a computer's CPU executes a computer program
- describe the fundamentals of how computer components communicate with each other and with peripheral devices.

Essential reading

Brookshear, J.G. *Computer Science: An Overview*. (Boston, Mass.: Pearson, 2009) Chapter 2, sections 2.1–2.3 and 2.5.

Reynolds, C. and P. Tymann *Schaum's Outline of Principles of Computer Science* (Schaum's Outline Series). (New York: McGraw-Hill, 2008) relevant sections of Chapter 3.

Further reading

Patterson, D.A. and J.L. Hennessy *Computer Organization and Design: the Hardware/software Interface.* (Burlington, Mass.: Elsevier Morgan Kaufmann, 2008).

Stallings, W. *Computer Organization and Architecture, Designing for Performance*. (Boston, Mass.: Pearson, 2010).

Tanenbaum, A. *Structured Computer Organization*. (Upper Saddle River, N.J.: Pearson Prentice Hall, 2010).

Introduction

Chapter 1 has provided you with an overview of how computers have evolved since people first attempted to use technology to help them with their calculations. It has also introduced the Von Neumann architecture as a fundamental basis for computer operation. Chapter 2 has then given you an introduction into how data needs to be represented in binary form for a computer to store it, and it has also shown you that (arithmetic) operations can be performed upon such binary data.

When a computer runs a program it needs to carry out a sequence of instructions (which may involve carrying out arithmetic operations, but also others). Like its (other) data these instructions also have to be encoded in a binary format. Chapter 3 looks at how instructions are represented in a binary format and how a CPU recognises and performs such instructions.

Machine language

At this point, you should recall the underlying idea and the main components of the Von Neumann architecture as introduced in Chapter 1: The computer stores the **program code** with the **data** in the main memory, and the CPU controls the execution of this code by first loading (fetching) the code for an instruction from the memory into the registers of the CPU, decoding the instruction, and then executing it. Hence program execution involves the repetition of these three steps: fetching, decoding, executing. The processing required for a single instruction is sometimes also referred to as an **instruction cycle**.

CPUs are designed to understand a fixed number of instructions. These instructions (that are fetched, decoded and executed) need to be represented as bit patterns in order for them to be stored and for the CPU to understand them (in the same way as data needs to be represented in binary form). The collection of instructions a CPU can understand is known as the CPU's **instruction set**. The low-level representation of a CPU's instruction set is known as the **machine language** of a computer.

Obviously, it is hard for us to write instructions in binary form directly. A much more readable form of machine language is what we refer to as **assembly language**. Assembly language uses mnemonic (memorable) code words to describe an instruction. For example, an assembly language may express the moving of the contents of register 1 to register 2 as MOV R1, R2. It is then the task of an **assembler** to translate the assembly language code into the machine code 0100000000010010.

The next section gives you an insight into the kind of instructions that form part of a typical instruction set.

Machine instructions – an overview

At the most fundamental level, the actions that are performed by the CPU are rather limited, and hence the instruction set is surprisingly small. The list below summarises the kind of instructions that constitute an instruction set together with concrete examples (using mnemonic representation) for each kind (Stallings 2010).

- Processor–memory transfer instructions: data may be transferred from the CPU to memory or vice versa.
  Example: MOVE
- Processor–input/output (I/O) transfer: data may be transferred from the CPU to external devices (e.g. printer, monitor) or received from external input devices (e.g. keyboard, scanner).
  Examples: READ, WRITE
- Arithmetic or logic operations: these actions involve the common arithmetic operations (e.g. addition) and Boolean operations.
  Examples: ADD, AND
- Control flow operations: these control the sequence of the execution of the program, e.g. they may determine the next location in memory from which an instruction is being fetched.
  Example: BRANCH

Program execution

In order to gain a better understanding of how machine language works, let us now look at an example of how its instructions can be combined to carry out certain tasks. For this purpose we will borrow the machine language and CPU architecture proposed in Appendix C of Brookshear (2009).

Our computer has the following (see figure below).

- A CPU with:

  a program counter (PC), which is a special register that holds the address of the instruction that needs to be fetched next

  an instruction register (IR) that holds the fetched instruction

  16 general-purpose registers for the temporary storage of the data that is relevant to the current instruction

- A main memory with 256 memory cells (0–255) of a size of 8 bits each.

Note that the 16 general-purpose registers are numbered 0 to F, and the memory addresses range from 00 to FF using the hexadecimal system. The hexadecimal representation is usually used to make long strings of bits more readable. In the hexadecimal system the numbers 10–15 are represented by the letters A–F. This means, for example: $1001_2 = 9_{10} = 9_{16}$ and $1010_2 = 10_{10} = A_{16}$. The convention is that four bits are grouped together and represented by a single hexadecimal digit. Therefore, $1111\ 1111_2 = 256_{10}$ is represented here as FF.

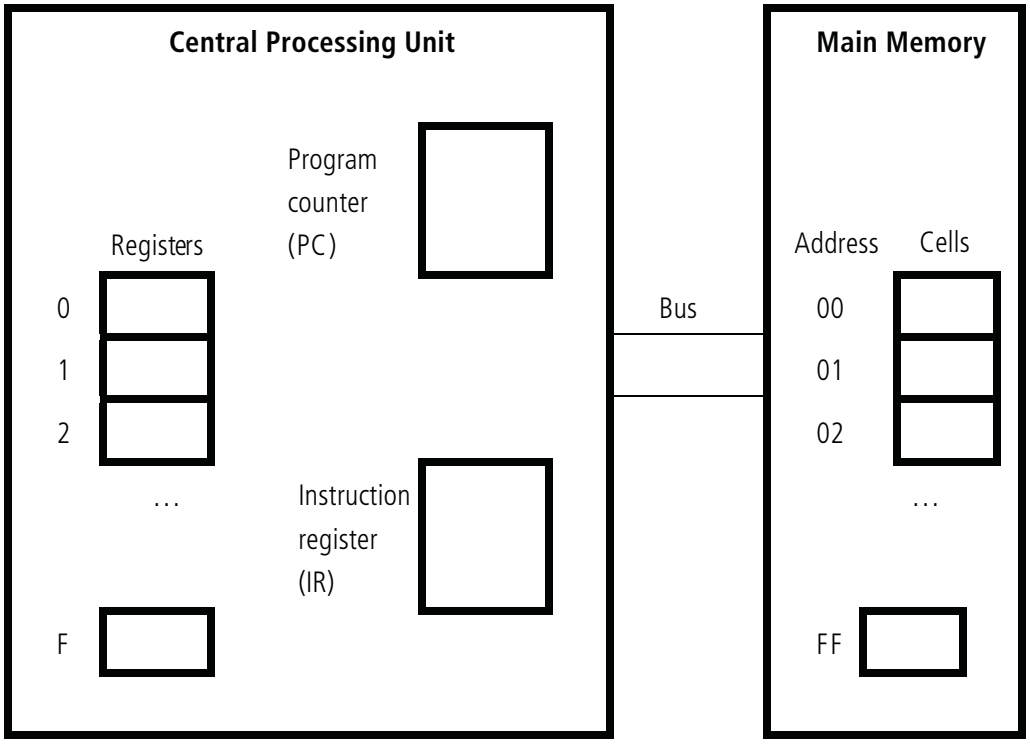

Figure 3.1: A computer architecture

We use the representation of the basic architecture above (adapted from Brookshear 2009) to look at what happens within the computer when program code is executed.

Following the stored program concept, all instructions (as part of the program code) are stored as a list in the main memory (together with the data). The CPU executes these instructions within its instruction cycle as follows.

In a first step, an instruction is fetched from the main memory. The program counter (PC) stores the address of where the next instruction needs to be fetched from, and the fetched instruction is loaded into the instruction register (IR). The PC automatically counts forward (unless an instruction demands a jump in the program sequence, in which case the instruction causes the CPU to set the PC to the new (diverted) address).

The instruction is then decoded and executed by the processor. This may involve actions such as initiating a certain circuitry in the arithmetic/logic unit or sending signals to the main memory to load data into a certain register.

Once the execution of the instruction has been completed, the next cycle starts again with fetching the next instruction.

Reading

Now read Brookshear (2009) Chapter 2, Sections 2.1–2.3.

Communication with peripherals – the controller

The section above has investigated the communication within the CPU and between the CPU and its main memory. Obviously, communication also takes place between the computer and many other input/output (I/O) devices (peripherals) (for example, the screen, the keyboard, the printer, an external hard drive, a digital camera, etc.). Each of these peripherals is linked to the computer's bus through a **controller**.

The controller generally constitutes a small computer in itself with its own functions and temporary memory (buffer) for the data that is sent or received. The controller is entirely devoted to managing the communication with the peripherals via the bus. Communication is supported by use of interrupts (Reynolds and Tymann 2008). When a program calls for output, for example, the relevant data is moved into the buffer of the controller and the controller is instructed to start the output operation. After completion, the controller sends an **interrupt** to notify the main computer that it is ready for further output/action, and the original program can send further data.

The development of standards has led to controllers such as the USB (universal serial bus) controllers and FireWire that allow the use of a single controller for a wide range of different devices (Brookshear 2009).

Some of these controllers communicate with the CPU as if they were memory addresses (i.e. data can be LOADED and STORED via the controller). Obviously, in order for this not to interfere with communication with the main memory, a set of addresses is reserved for the controller. This approach is referred to as **memory-mapped I/O** (Reynolds and Tymann 2008). Alternatively, a computer may have special instructions that direct the transfer of the data to and from the controller (I/O instructions, such as 'STORE').

Today's computers generally have controllers that can access main memory directly without the need for any intervention by the CPU. This is referred to as **direct memory access** and has the advantage that efficiency is increased as the CPU can continue its computations while data is moved in or out of memory. Here the complications of the Von Neumann bottleneck introduced in Chapter 1 become obvious again: The use of the bus needs to be optimised for the transactions between the CPU, the controller and the main memory.

Parallel versus serial communication

Two different kinds of communication are used for the transaction of data: **parallel communication** and **serial communication** (Brookshear 2009). Within parallel communication data is sent simultaneously in several parallel channels. An 8-bit parallel data link, like a bus, transmits one byte of data simultaneously. Within serial communication only one

channel is used, through which all data is sent sequentially. If we assume that both communication links operate at the same speed, it is obvious that the parallel link can transfer significantly more data in the same amount of time (the speed is measured in bits per second (bps, but also Kbps, Mbps and Gbps)). However, the enormous advancement of technologies over the past years has led to high-speed serial links, and serial communication is now more commonly used than parallel communication. Just think of the connections on your computer. Your USB (universal serial bus) or FireWire port are typical examples of serial ports used to link to external devices today. The typical Ethernet connection used within local area networks also is a serial link. For greater distances we often use telephone lines as a serial medium to transfer digital data. DSL (digital subscriber line) services, for example, use telephone lines for high-speed internet connections by making use of frequencies that are not audible to us. The lower frequencies can then still be used for voice communication as we know it.

You should now have a good understanding of what computer architecture constitutes and how a computer understands and executes program code. You should also be able to explain the fundamentals of how data is transferred between computer components and between the computer and external devices. The next chapter looks more closely at how a computer's internal operations are coordinated by its operating system.

Reading

Now read Brookshear (2009) Chapter 2, section 2.5 and Reynolds and Tymann (2008), relevant sections of Chapter 3.

Activity

The choice between serial communication and parallel communication depends on a number of factors. With the help of the relevant literature find out what these factors are and how they influence this choice.

A reminder of your learning outcomes

After completing this chapter, and the Essential reading and activities, you should be able to:

- demonstrate an understanding of the concept of using machine language to program a computer
- explain how a computer's CPU executes a computer program
- describe the fundamentals of how computer components communicate with each other and with peripheral devices

Sample examination question

1. a. Explain the difference between data and program. (5 marks)

   b. The stored program concept of the Von Neumann architecture entails the storage of both data and program within the same memory. Also, both data and program are encoded in binary form. Discuss the advantages and disadvantages of this design. (10 marks)

   c. Apart from internal data transfer between computer components, the computer may have to communicate with its peripherals. Explain how this process is managed. (10 marks)

Notes