Tutorial Letter 102/3/2022

Computer Systems: Fundamental Concepts

COS1521

Semesters 1 and 2

School of Computing

IMPORTANT INFORMATION:

This tutorial letter contains important information about your module.

Note: This is a blended online module and therefore it is available on myUnisa. However, in order to support you in your learning process, you will receive this tutorial letter and the study guide in printed format.

BARCODE

Define tomorrow.

UNISA university of south africa

CONTENTS

Page

NB: You should read each unit in this letter as you study the corresponding unit in the textbook. However, you are strongly advised to read the content of this latter **from page 24** onwards since it contains simplified methods of dealing with:

1. Number systems
2. Boolean algebra
3. Creating logic circuits
4. Creating and simplifying Karnaugh maps
5. Minterms
6. Etc…

Please go through these pages **before** you contact the lecturer for further assistance. All the content in these pages (and the rest of pages) is **examinable**.

PREFACE

This tutorial letter consists of two parts. In this letter, we provide the purpose, outcomes, assessment criteria and ranges for this module, information regarding the study material covered in the prescribed book, explanations on the study material covered in Appendix E and additional exercises with solutions and explanations.

Part I of this tutorial letter contains the purpose, outcomes and ranges for this module (similar to what is given is Tutorial letter 101, and notes on the study material for this module that is covered in the prescribed book, namely,

Author: Forouzan, Behrouz
Title: *Foundations of Computer Science*
Edition: 4th
Year: 2017
ISBN-10: 1-4737-5104-7
ISBN-13: 978-1-4737-5104-0

We refer to the prescribed book as **Forouzan** in the study material. If you are still using the 3rd edition, please note that page number references will be different.

The preface in **Forouzan** provides information about the composition of the book. At the end of each chapter the authors provide a list of key terms and a summary that should be handy for identifying and revising all the important concepts discussed in the chapter. A list of review questions as well as multiple-choice questions are also provided that should be used for testing your understanding of the study material covered in each chapter. Solutions to odd-numbered review questions are provided at a URL which will be provided to you on myUnisa or via email. It will be under:

http://www.cengage.co.uk

Tutorial Letter 101 contains information regarding this tutorial letter and the chapters in the prescribed book that you need to study in order to prepare for the assignments that should be done. Also look in the Tutorial Letter 101 for sections/subsections in the textbook which are excluded for this module from the prescribed chapters.

Chapters 12, 15-18 and Appendixes B-D, F-H of **Forouzan** do not form part of the syllabus for COS1521. The topics in these chapters and appendixes are dealt with in detail in some other modules in the School of Computing. Appendix A of **Forouzan** does not form part of the syllabus but it can be read together with Section 3.3, Chapter 3.

Part II provides additional exercises with solutions and supplementary explanations and/or study material for Chapters 2–4 and Appendix E of **Forouzan**.

Part I

Manual for the prescribed book

Module purpose, outcomes, assessment criteria and ranges

Purpose

COS1521 is one of a number of first-year Computer Science modules offered by the School of Computing at Unisa.

The purpose of this module is to introduce students to the computer as a system. The module covers hardware concepts such as internal representation of numbers and characters and basic computer architecture, and software concepts such as systems software and applications software. It also includes a brief introduction to databases, and to systems analysis and design.

Module outcomes, assessment criteria and ranges

Learning outcomes Students should be able to:	**Assessment Criteria** Evidence in the form of formative assessment (assignments) and summative assessment (examination) will show that:	**Range Statements**
Demonstrate how data are represented, manipulated and stored in a computer using number systems, Boolean algebra, Karnaugh maps, truth tables and basic logic circuits drawings, in the context of given problem statements, drawings, in the context of given problem statements.	• Conversions between different number systems (binary, octal, decimal and hexadecimal); • The application of different arithmetic methods in the binary number system; • The identification of computer data includes the different internal representations; • Explanations include the basic restrictions placed by computer architecture upon numerical computations; • The determination of outputs of basic combinational logic circuits for given inputs; • Graphical representations of the combinational circuits for given Boolean functions; • The simplifications of Boolean functions by implementing appropriate rules/methods; • The determination of a Boolean function for a given problem statement using truth tables (at most 4 variables); • Boolean expressions and binary logic that describe the behaviour of logic circuits; • The descriptions of the functioning of different types of combinational and sequential logic circuits.	Basic knowledge of internal data, logic gates, and memory elements will be demonstrated only in the context of the design of basic combinational and sequential logic circuits.
2. Demonstrate an understanding of the basic functions of computers, the software development process and units of hardware and software components.	• Today's computers are described in context of some short historical background, different architectures and ethical scenarios/issues; • Descriptions of software engineering and operating systems include the development of software in a historical context; • The description of a basic computer includes the three basic hardware	The context is basic computer hardware and systems software with its relevant algorithms.

	subsystems and their interconnectig functioning; • The description of an operating system includes the functioning of its components; • The descriptions of popular operating systems with references to different popular operating platforms; • The definition of an algorithm includes its relation to problem solving; • Definitions of the three algorithm constructs include descriptions of their use in algorithms; • Descriptions of basic algorithms include their applications; • Descriptions of the sorting and searching concepts of algorithms include an understanding of their mechanisms; • Descriptions of subalgorithms include their relations to algorithms; • Descriptions of the development process models in software engineering include the concepts of the software life-cycle phases and documentation.	
3. Demonstrate an understanding of the basics of data communications and networks.	• Descriptions of physical structures of networks include references to network criteria, physical structures and categories of networks; • The description of the Internet includes the TCP/IP protocol suite with reference to the characteristics of its layers and their relationships; • Descriptions of Internet applications in the context of client-server communications.	The context is the basics of Information Communication Technologies.
4. Describe datastuctures and how different databases function.	• Descriptions of data structures include references to the differentiation beween different structures; • Descriptions of file structures include references to updating and access methods, and categories of directories and of files; • Definitions of a database and some traditional database models include the relational database design; • The definition of a database management system (DBMS) includes its architecture; • Descriptions include the steps in database design.	The contexts are typical of the demands of first-year undergraduate study.

The specific assessment criteria for each chapter in **Forouzan** are listed on the first page of each chapter in the prescribed book.

The paragraphs below show where COS1521 fits into first-year modules offered by the School of Computing:

- COS1511 deals with the basic concepts of programming, using the programming language C++. It is aimed at students who have not done any programming before.
- COS1512 introduces the learner to objects and the object-oriented programming environment. COS1511 is a corequisite for COS1512.
- COS1521 provides a general background to computer systems.
- COS1501 introduces discrete mathematics relevant to Computer Science.
- INF1511 is an introductory course in Delphi programming.
- INF1505 provides a general background to business information systems.
- INF1520 deals with human-computer interaction.

•

UNIT 1

Introduction

Purpose

This unit supplements Chapter 1 of **Forouzan**. It provides an introduction to the computer as a device used for processing data.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

This chapter in **Forouzan** introduces the concept of considering the computer as a *black box* that processes data. This principle is often used in Computer Science in the sense that, when we describe a computing entity, whether it be a hardware device or a computer program, we are sometimes only interested in the output that will be produced given certain inputs. What happens inside the black box may not be of interest to us. The Turing model is used to illustrate this principle.

Forouzan describes the von Neumann model on which most modern computers are based. The concept of data that can be processed by a computer is discussed. Some of the software concepts that will be considered in ensuing chapters are introduced. A short history of the development of computers is given. The authors then highlight some social and ethical issues that resulted from the emergence of the modern digital computer and finally provide a short discussion of Computer Science (or Computing as we refer to it) as a discipline.

UNIT 2

Number Systems

Purpose

This unit supplements Chapter 2 of **Forouzan**. Different number systems and the way in which conversions between these number systems can be done are discussed.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book. In addition to these outcomes, you must be able to convert a hexadecimal number to octal and vice versa.

Note: The second outcome in the list involving non-positional number systems should be excluded because Section 2.3 does not form part of the syllabus.

Overview of the chapter

The binary, octal, decimal and hexadecimal number systems are discussed in this chapter. The conversions between the different number systems are explained.

Note: Section 2.3 (Non-positional number systems) does not form part of the syllabus.

UNIT 3

Data Storage

Purpose

This unit supplements Chapter 3 of **Forouzan**. It introduces different data types and the way in which these are represented in computers.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Note: Appendix A (*Unicode*) explains in some detail how text is stored by using ASCII and Unicode. Please note that Appendix A does not form part of the syllabus but you can read through it.

Overview of the chapter

This unit considers the representation of different types of data in the form of bit patterns. We look briefly at the different ways in which numbers, text, images, audio and video data are represented.

It is extremely important to remember the following:

All kinds of data are stored in the form of bit patterns, i.e. 0s and 1s.

The computer may use the hexadecimal number system, for example, to display the contents of memory on the screen but this is only to make it easier for us to read. All data in memory is in binary form, i.e. represented as 0s and 1s. Note that a bit pattern of length 4 is called a *nibble*.

Different ways in which integers can be represented are considered. These include the representation of unsigned numbers as well as the representation of signed numbers in sign-and-magnitude, and one’s and two’s complement format. The way in which real numbers are stored by using floating-point format, is discussed.

In the previous unit we saw how a decimal numbers can be represented as a string of bits by means of a combination of zeros and ones. We also said that a computer can only work effectively once all incoming data (and, in fact, the computer instructions) have been translated into a binary form. This binary form is a "natural" form for a computer, because it can very easily be implemented by internal electronic circuits. We will be taking a closer look at this implementation in the following two units.

In this unit, we consider the way in which alphabetic, numeric and other characters are represented inside the computer. We already know that these characters must be in binary form, but what, for instance, is the binary form of an "A" or a ","?

Computer manufacturers generally make use of a grouping of bits in order to achieve this representation. This grouping is done in accordance with some or other code. Just as octal and hexadecimal constitute a shorter way of writing binary numbers, these codes constitute a shorter way of writing down what is actually going on inside the computer.

The amount of data that can be stored in a computer depends on the size of the memory of the computer. The size of the memory is usually described in terms of the number of addressable memory positions, also known as computer words. The length or size of such a word is decided upon by the manufacturer and can consist of 4, 8, 16, 32, 48, 64 or even more bits.

UNIT 4

Operations on Data

Purpose

This unit supplements Chapter 4 and Appendix E of **Forouzan**. Arithmetic and logical operations on data are discussed. The basics of Boolean algebra and logic circuits are also covered.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book. The outcomes for Appendix E are listed below.

Learning outcomes for Appendix E

Once you have mastered the study material covered in Appendix E, you will be able to:

- determine the outcome of simple logic circuits when given the inputs,
- describe the effect of the NOT, AND, OR, NAND, NOR and XOR gates,
- use Boolean expressions to describe the behaviour of logic circuits,
- draw a logic circuit when given a Boolean expression or problem statement,
- determine whether or not two logic circuits are logically equivalent,
- use Karnaugh maps to simplify Boolean expressions,
- describe a combinational circuit,
- use a truth table to design a combinational circuit for a given circuit,
- distinguish between combinational and sequential circuits,
- describe how flip-flops are triggered, and
- explain how registers are constructed from flip-flops and how they function inside a computer.

Note: Please note that the ‘product of sums’ (pp. 570-573 (ed.4), pp. 550-553 (ed.3)) does not form part of the syllabus. Neither does Fig. J.1 and Example J.1 in Appendix J (operations on reals).

4.1 Overview of Chapter 4 and Appendix E

Chapter 4 in **Forouzan** should be studied together with Appendix E in **Forouzan**, this Unit 4 and the relevant material in Part II of this letter. **Note**: Part II provides additional study material and explanations that should be studied. This is a very important part of the study material for this module and you should pay particular attention to the concepts explained here.

In Chapter 4, operations on data such as logic operations, shift operations and arithmetic operations are discussed. Logical operations such as AND, OR, NOT and XOR are explained as well as the use of these to access individual bits within a byte or word. It is described how logical shift operations can be used to move bits within a byte. Note the difference between a logical shift and an arithmetic shift operation.

In Appendix E, it is explained how Boolean algebra can be used as a means of representing information in a computer. We briefly look at the two broad categories of logic circuits namely combinational and sequential circuits.

4.2 Summary of Appendix E: Boolean Algebra and Logic circuits

- Computers are built with components that can be in one of two states: on or off. *Boolean algebra* is used to represent information in a computer as it involves variables and constants that can only take on the values 0 or 1.
- *Boolean expressions* are combinations of *constants* (0 or 1), *variables* (letters such as *x*, *y* or *z*, etc) and *basic operators* (NOT, AND or OR respectively indicated by ′, · and +). In some cases we do not indicate AND by ·, we simply write, for example, xy in stead of x · y.
- *Logic gates* are electronic devices that create one output for some inputs. Boolean expressions are used to represent these inputs and outputs for which the logical values can then be determined. Logic gates such as the *buffer, NOT, AND, OR, NAND, NOR and XOR gates* each creates a specific *output* for some *input(s)* and these operations are described by using *truth tables*.
- *Electronic switches* are used in the *implementation of logic gates*. In these implementations, mostly NOT, NAND and NOR gates are used. Switches open or close when appropriate voltages are applied to inputs. In practice, switches are replaced by transistors that behave like switches when used in gates. *Input and output signals are interpreted in terms of 0s or 1s*. A 0 indicates no current and a 1 indicates that a current flows through the resistor.
- The *NOT gate* can be implemented by using an *electronic switch, a voltage source and a resistor*, the *NAND gate* by using *two electronic switches in series* and the *NOR gate* by using *two switches in parallel*. The behaviour of these circuits matches the values in the truth tables

of the corresponding logic gates.

- *Rules* are used in *Boolean algebra*. These rules are divided into three categories namely *axioms*, *theorems* and *identities*. All the rules can be extended by using more variables (eg. ab + cd = cd + ab – commutativity) and the rules can be used to simplify Boolean expressions. *De Morgan's rules* play a very important role in logic design.
- A *Boolean function* is a function with *n* Boolean input variables and one Boolean output variable that can be represented by a *truth table* or an *expression*.
- To implement a Boolean function by using logic gates, an expression for its truth table must be found. We implement the *sum of products* method where the function is expressed in terms of *minterms*. (Note that 'product of sums' and Example E.7 is excluded from the syllabus.)
- The number of gates required for a Boolean function can be reduced if simplification is carried out by means of the *algebraic method* (by applying Boolean algebra rules) or a *Karnaugh map*.
- Normally the standard components of a computer are logic circuits that can be divided into two categories namely *combinational circuits* and *sequential circuits*.
- A *combinational logic circuit* consists of a combination of logic gates with *n* inputs and *m* outputs and each *output depends entirely on all inputs*. Logic *circuits* can be logically *equivalent*. A *half adder* and a *multiplexer* are examples of combinational circuits.
- A *sequential circuit* can remember its current state to be used in future, in other words, a future state can be dependent on a current state. *Flip-flops*, *registers* and *digital counters* are examples of sequential circuits. It is possible to change a flip-flop from an *asynchronous* device to a *synchronous* device by adding a clock input.

UNIT 5

Computer Organisation

Purpose

This unit supplements Chapter 5 of **Forouzan**. The internal organisation of a modern digital computer is considered.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Note: Subsections 5.7.3 (Pipelining) and 5.7.4 (Parallel processing) and section 5.8 (A simple computer) are not included in the syllabus.

Overview of the chapter

This is a very important part of the study material for this module and you should pay particular attention to the concepts explained here. The internal organisation of a stand-alone computer is explained, starting with the identification of three subsystems. The internal organisation of each of these subsystems, as well as the way in which they can be interconnected, are explained. We also look at different ways in which I/O operations to and from memory can be handled.

The authors describe the way in which instructions within a computer program are executed. Different ways in which I/O operations to and from I/O devices can be synchronised with CPU operations are also discussed.

This chapter provides a discussion on the different approaches to computer architecture and organisation used on CISC and RISC machines.

Note: Read through the subsections of Section 5.6 (Different architectures) on pipelining and parallel processing, and Section 5.7 (A simple computer) where a simple computer system is used to illustrate the concepts covered in this chapter. Please note that these parts are beyond the scope of this module and need not be studied for the examination.

UNIT 6

Computer Networks

Purpose

This unit supplements Chapter 6 of **Forouzan**. Network principles as well as some devices and protocols used in networks.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

Different criteria that are relevant in computer networking are discussed first. **Forouzan** then describes some physical structures applicable to networks and discusses different network topologies. The different categories in which networks can be divided are also explained.

The way in which the popular TCP/IP network protocol suite is used to establish communication between different nodes on a network is explained. **Forouzan** provides a lay-out of the different layers within this protocol and gives an explanation of the role played by each layer.

Since the Internet plays such an important part in our daily lives, the authors discuss some applications that provide service to Internet users. These include email, the File Transfer Protocol (FTP) and the World Wide Web (WWW). Different components of the WWW are also described as well as the different kinds of documents that can be found within this environment.

UNIT 7

Operating Systems

Purpose

This unit supplements Chapter 7 of **Forouzan**. An introduction to the most important operating system concepts is provided.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

The evolution of operating systems is described. We also consider the different components comprising an operating system and discuss the main function of each component.

Finally, we look at a number of popular operating systems, namely Unix, Linux and Windows.

UNIT 8

Algorithms

Purpose

This unit supplements Chapter 8 of **Forouzan**. The concept of an algorithm is introduced.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book. **Note**: The last outcome in the list involving iterative and recursive algorithms should be excluded because Section 8.7 does not form part of the syllabus.

Overview of the chapter

The concept of an algorithm is introduced. The three main constructs used in algorithms, namely a sequence, a decision and a repetition are explained. We also look at two different ways in which an algorithm can be presented and at the concept of a subalgorithm.

By using several examples, such as sort and search algorithms, the textbook illustrates how algorithms can be used to provide a description of how a problem could be solved.

Work through all the examples of algorithms in Section 8.5 and make sure you understand how they work. You need not memorise any of these algorithms.

Note: Please note that Section 8.7 (Recursion) does not form part of the syllabus and will be dealt with in a second year module.

UNIT 9

Programming Languages

Purpose

This unit supplements Chapter 9 of **Forouzan**. Programming languages and different programming paradigms are discussed and compared.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book. **Note**: The last outcome in the list involving common concepts in languages should be excluded because Section 9.4 does not form part of the syllabus.

Overview of the chapter

An introduction of the evolution of computer languages is given. Translation methods by which high-level programming languages are converted to machine language are described. Different programming paradigms are compared and contrasted.

Note: Note that Section 9.4 (Common concepts) does not form part of the syllabus.

UNIT 10

Software Engineering

Purpose

This unit supplements Chapter 10 of **Forouzan**. The reader is introduced to software engineering.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

The software lifecycle is introduced. Two different process models for systems development are examined.

The software lifecycle consists of four phases. We consider the activities that occur and the tools that are used in each of these phases.

In the analysis and design phases, object- and procedure-oriented approaches are described. Software quality factors in the implementation phase are discussed. The attributes that determine the quality of a software product are examined and different types of testing are explained. Finally we look at the importance of documentation during each phase of the software lifecycle.

UNIT 11

Data Structures

Purpose

This unit supplements Chapter 11 of **Forouzan**. Different ways in which data can be stored by using data structures are considered.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

Different data structures such as arrays, records and linked lists are studied. We look at one- and two-dimensional arrays and describe how operations on these are handled. The way in data in which two-dimensional arrays are stored, is explained.

The concepts of a record and of fields within a record are discussed and the way in which individual records and fields are accessed is considered.

Finally, we look at linked lists, a data structure which you will frequently come across during your Computer Science studies. The use of pointers in such a list and operations on nodes within a linked list is explained and the way in which a linked list is traversed is discussed.

Note: **Chapter 12 of Forouzan is not included in the syllabus for this module.** Abstract data types will be dealt with in detail in some of the second-year modules. So, nothing to be studied here! Unit numbers correspond with chapter numbers, that is why UNIT 12 has been omitted.

UNIT 13

File Structures

Purpose

This unit supplements Chapter 13 of **Forouzan**. The concepts of files, file structures and file access methods are considered.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book.

Overview of the chapter

Different files structures are discussed. We first look at sequential files and how these are accessed and updated. Then we consider types of file that can be accessed randomly. These include indexed files and hashed files. We look at the way in which indexed files are accessed and explain what an inverted file is. The concepts of hashed files and different hashing methods are then discussed, as well as the way in which hashing collisions could be handled. The importance of the use of directories for systematically organising your files is described.

Finally, the differences between text files and binary files are high-lighted.

UNIT 14

Databases

Purpose

This unit supplements Chapter 14 of **Forouzan**. The reader is introduced to the most important database concepts.

Learning outcomes

The learning outcomes for this unit are listed on the first page of the corresponding chapter in the prescribed book. **Note**: The three outcomes (before the last outcome) involving database design should be excluded because Section 14.6 does not form part of the syllabus.

Overview of the chapter

The components comprising a Database Management System (DBMS) are introduced. We look at the three-level DBMS architecture proposed by the American National Standards Institute / Standards Planning and Requirements Committee (ANSI/SPARC). Three different database models, namely the hierarchical model, the network model and the relational model, are then introduced.

We look in detail at the relational model, a model that is commonly used. The concept of a relation is explained and different operations on relations are shown. The way in which Structured Query Language (SQL) is used to perform operations on a database is also illustrated.

Note: Please note that you need not know the syntax of SQL statements.

Note: Section 14.6 (Database design) of **Forouzan** does not form part of the syllabus.

Finally, we consider two other types of database that are commonly used in industry today, namely distributed databases and object-oriented databases.

The theory and practice of databases are further studied in some third-year modules.

Chapters 12 and15 - 18 of **Forouzan** are not included in the syllabus for this module. These topics are dealt with in detail in some other modules in the School of Computing.

Part II

Additional exercises and/or explanations for Forouzan, Chapters 2-4 and Appendix E

All page references are to the prescribed book, Forouzan.

Chapter 2

Number systems: Conversions between number systems

Exercise 2.1

Consider Table 2.2, (p. 22-23 (ed.4), p. 24-25 (ed.3)). Convert the binary and octal numbers to decimal.

Solution:

Binary to decimal:

$(0)_2 = 0x2^0 = (0)_{10}$

$(1)_2 = 1x2^0 = (1)_{10}$

$(10)_2 = 1x2^1 + 0x2^0 = 2+0 = (2)_{10}$

$(11)_2 = 1x2^1 + 1x2^0 = 2+1 = (3)_{10}$

$(100)_2 = 1x2^2 + 0x2^1 + 0x2^0 = 4+0+0 = (4)_{10}$

$(101)_2 = 1x2^2 + 0x2^1 + 1x2^0 = 4+0+1 = (5)_{10}$

$(110)_2 = 1x2^2 + 1x2^1 + 0x2^0 = 4+2+0 = (6)_{10}$

$(111)_2 = 1x2^2 + 1x2^1 + 1x2^0 = 4+2+1 = (7)_{10}$

$(1000)_2 = 1x2^3 + 0x2^2 + 0x2^1 + 0x2^0 = 8+0+0+0 = (8)_{10}$

$(1001)_2 = 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 8+0+0+1 = (9)_{10}$

$(1010)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 8+0+2+0 = (10)_{10}$

$(1011)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = 8+0+2+1 = (11)_{10}$

$(1100)_2 = 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 8+4+0+0 = (12)_{10}$

$(1101)_2 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8+4+0+1 = (13)_{10}$

$(1110)_2 = 1x2^3 + 1x2^2 + 1x2^1 + 0x2^0 = 8+4+2+0 = (14)_{10}$

$(1111)_2 = 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 8+4+2+1 = (15)_{10}$

One more example: $(10000)_2 = 1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0 = 16+0+0+0+0 = (16)_{10}$

Octal to decimal

$(0)_8 = 0x8^0 = (0)_{10}$

$(1)_8 = 1x8^0 = (1)_{10}$

$(2)_8 = 2x8^0 = 2x1 = (2)_{10}$

$(3)_8 = 3x8^0 = 3x1 = (3)_{10}$

$(4)_8 = 4x8^0 = (4)_{10}$

$(5)_8 = 5x8^0 = (5)_{10}$

$(6)_8 = 6x8^0 = (6)_{10}$

$(7)_8 = 7x8^0 = (7)_{10}$

$(10)_8 = 1x8^1 + 0x8^0 = 8+0 = (8)_{10}$

$(11)_8 = 1x8^1 + 1x8^0 = 8+1 = (9)_{10}$

$(12)_8 = 1x8^1 + 2x8^0 = 8+2 = (10)_{10}$

$(13)_8 = 1x8^1 + 3x8^0 = 8+3 = (11)_{10}$

$(14)_8 = 1x8^1 + 4x8^0 = 8+4 = (12)_{10}$

$(15)_8 = 1x8^1 + 5x8^0 = 8+5 = (13)_{10}$

$(16)_8 = 1x8^1 + 6x8^0 = 8+6 = (14)_{10}$

$(17)_8 = 1x8^1 + 7x8^0 = 8+7 = (15)_{10}$

Exercise 2.2

Convert the following numbers to decimal:

a) $(11111)_2$; $(110010.1)_2$; $(10001.1010)_2$

b) $(203)_8$; $(102.1)_8$; $(20.101)_8$

c) $(A01)_{16}$; $(C.A)_{16}$; $(100A)_{16}$

Solution:

a) Binary to decimal:

$(11111)_2 = 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0$

$= 16 + 8 + 4 + 2 + 1$

$= (31)_{10}$

$(110010.1)_2 = 1x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0 + 1x2^{-1}$

$= 32 + 16 + 0 + 0 + 2 + 0 + ½$

$= (50.5)_{10}$

$(10001.1010)_2 = 1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4}$

$= 16 + 0 + 0 + 0 + 1 + ½ + 0 + ⅛ + 0$

$= (17.625)_{10}$

b) Octal to decimal:

$(203)_8 = 2x8^2 + 0x8^1 + 3x8^0 = 128 + 0 + 3 = (131)_{10}$

$(102.1)_8 = 1x8^2 + 0x8^1 + 2x8^0 + 1x8^{-1} = 64 + 0 + 2 + ⅛$

$= (66.125)_{10}$

$(20.101)_8 = 2x8^1 + 0x8^0 + 1x8^{-1} + 0x8^{-2} + 1x8^{-3}$

$= 16 + 0 + \frac{1}{8} + 0 + {}^{1}/_{512}$

$= (16.126953125)_{10}$

c) Hexadecimal to decimal:

$(A01)_{16} = Ax16^2 + 0x16^1 + 1x16^0$

$= 10x256 + 0 + 1x1$

$= 2560 + 1$

$= (2561)_{10}$

$(C.A)_{16} = Cx16^0 + Ax16^{-1}$

$= 12x1 + 10x16^{-1}$

$= 12 + 10x\ {}^{1}/_{16}$

$= 12 + 0.625$

$= (12.625)_{10}$

$(100A)_{16} = 1x16^3 + 0x16^2 + 0x16^1 + Ax16^0$

$= 4096 + 0 + 0 + 10x16^0$

$= 4096 + 10$

$= (4106)_{10}$

Exercise 2.3

Convert the following decimal numbers to binary (fractional part up to 5 binary digits):

a) 91; b) 0.55; c) 0.6875; d) 32.125

Solution:

a) 91 to binary:

0←1←2←5←11←22←45←**91** Decimal (Divide repetitively by 2; remainders appear below.)

↓	↓	↓	↓	↓	↓	↓	
1	0	1	1	0	1	1	Binary

The result is $(91)_{10} = (1011011)_2$.

Alternative method:

91 / 2 = 45 remainder 1

45 / 2 = 22 remainder 1

22 / 2 = 11 remainder 0

11 / 2 = 5 remainder 1

5 / 2 = 2 remainder 1

2 / 2 = 1 remainder 0

1 / 2 = 0 remainder 1. ↑ Write down the **reverse** order of the binary digits:

Thus $(91)_{10} = (1011011)_2$.

These remainders in **reverse** order give the binary digits of the number. In the first method, the binary digits appear in the correct order in which the number is represented.

Another (preferred) method:

Break **91** into packets that are of the same sizes as the binary place values:

91 = 64 + 16 + 8 + 2 + 1. These numbers are represented as binary numbers:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0	1	0	1	1	0	1	1.

Thus $(91)_{10} = (1011011)_2$.

b) 0.55 to binary:

Decimal 0.55→0.10→0.2→0.4→0.8 (Multiply repetitively by 2; integer parts appear below.)

↓ ↓ ↓ ↓ ↓

Binary · 1 0 0 0 1 (5 binary digits)

The result is $(0.55)_{10} = (0.10001)_2$.

Alternative method:

0.55 x 2 = **1**.10 = 0.1 + **1**

0.1 x 2 = 0.2 + **0**

0.2 x 2 = 0.4 + **0**

0.4 x 2 = 0.8 + **0**

0.8 x 2 = **1**.6 = 0.6 + **1**. Thus $(0.55)_{10} = (0.\mathbf{10001})_2$.

c) 0.6875 to binary:

Decimal 0.6875→0.375→0.75→0.5→0.0 (Multiply repetitively by 2; integer parts appear below.)

↓ ↓ ↓ ↓

Binary · 1 0 1 1

The result is $(0.6875)_{10} = (0.1011)_2$.

Alternative method:

0.6875 x 2 = 1.3750 = 0.375 + 1

0.375 x 2 = 0.75 = 0.75 + 0

0.75 x 2 = 1.5 = 0.5 + 1

0.5 x 2 = 1.0 = 0.0 + 1. Thus $(0.6875)_{10} = (0.1011)_2$.

Note that the binary digits of the integer part are written in reverse order to that in which they were calculated. This is not true for the fractional part.

d) 32.125 to binary:

Integer part:

0←1←2←4←8←16←**32** Decimal (Divide repetitively by 2; remainders appear below.)

↓ ↓ ↓ ↓ ↓ ↓

1 0 0 0 0 0 Binary

The result is $(32)_{10} = (100000)_2$.

Alternative method:

32 / 2 = 16 remainder 0

16 / 2 = 8 remainder 0

8 / 2 = 4 remainder 0

4 / 2 = 2 remainder 0

2 / 2 = 1 remainder 0

1 / 2 = 0 remainder 1. Thus $(32)_{10} = (100000)_2$

Fractional part:

Decimal 0.125→0.25→0.5→0.0 (Multiply repetitively by 2; integral parts appear below.)

↓ ↓ ↓

Binary · 0 0 1

The result is $(0.125)_{10} = (0.001)_2$.

Alternative method:

0.125 x 2 = 0.25 = 0.25 + 0

0.25 x 2 = 0.5 = 0.5 + 0

0.5 x 2 = 1.0 = 0.0 + 1. Thus $(0.125)_{10} = (0.001)_2$.

The final result is $(32.125)_{10} = (100000.001)_2$.

Note again that the binary digits of the integer part are written in the *reverse* order to that in which it was calculated. This is not true for the fractional part.

Exercise 2.4

Convert the following decimal numbers to octal (truncated to 6 places):

a) 273;   b) 958;   c)10.36

Solution:

a) 273 to octal:

273 / 8 = 34 remainder 1

34 / 8 = 4 remainder 2

4 / 8 = 0 remainder 4. Thus $(273)_{10} = (421)_8$.

b) 958 to octal:

958 / 8 = 119 remainder 6

119 / 8 = 14 remainder 7

14 / 8 = 1 remainder 6

1 / 8 = 0 remainder 1. Thus $(958)_{10} = (1676)_8$.

c) 10.36 to octal:

Integer part:10 / 8 = 1 remainder 2

1 / 8 = 0 remainder 1. Thus $(10)_{10} = (12)_8$.

Fractional part:

0.36 x 8 = 2.88 = 0.88 + 2

0.88 x 8 = 7.04 = 0.04 + 7

0.04 x 8 = 0.32 = 0.32 + 0

0.32 x 8 = 2.56 = 0.56 + 2

$0.56 \times 8 = 4.48 = 0.48 + 4$

$0.48 \times 8 = 3.84 = 0.84 + 3$. Thus $(0.36)_{10} = (0.270243)_8$ (to 6 octal digits).

The final result is $(10.36)_{10} = (12.270243)_8$.

Exercise 2.5

Convert the following decimal numbers to hexadecimal:

a) 255; b) 128; c) 412

Solution:

a) 255 to hexadecimal:

255 / 16 = 15 remainder **15** = 15 remainder $\mathbf{F_{16}}$

15 / 16 = 0 remainder **15** = 0 remainder $\mathbf{F_{16}}$. Thus $(255)_{10} = (FF)_{16}$.

b) 128 to hexadecimal:

128 / 16 = 8 remainder 0

8 / 16 = 0 remainder 8. Thus $(128)_{10} = (80)_{16}$.

c) 412 to hexadecimal:

412 / 16 = 25 remainder **12** = 25 remainder $\mathbf{C_{16}}$

25 / 16 = 1 remainder 9

1 / 16 = 0 remainder 1. Thus $(412)_{10} = (19C)_{16}$.

Exercise 2.6

Convert the following binary numbers to octal:

a) $(101110.11)_2$; b) $(10110.101001)_2$; c) $(10.1011)_2$

Solution:

c) $(101110.11)_2$ to octal:
101 110.11 = $(101\ 110.110)_2$

= $(5 \quad 6\ .\ \ 6\ )_8$

= $(56.6)_8$

(Form groups of 3 digits; add a 0 at the end of the given number to form a group of 3 digits; convert each group to an octal number.)

b) $(10110.101001)_2$ to octal:

10110.101001 = $(010\ 110.101\ 001)_2$

= $(2 \quad 6\ .\ \ 5 \quad 1\ )_8$

= $(26.51)_8$

c) $(10.1011)_2$ to octal:
10.1011= $(010.101\ 100)_2$

= $(2\ .\ \ 5 \quad 4\ )_8$

= $(2.54)_8$

Note: the grouping of the fractional part must be done in such a way that the last group forms a complete octal number. Had this not been done, we would have written 2.51 instead of 2.54.

Exercise 2.7

Convert the following octal numbers to binary:

a) $(703)_8$; b) $(403.06)_8$; c) $(3.305)_8$

Solution:

a) $(703)_8$ to binary:
$(703)_8$ = $(\mathbf{7} \quad 0 \quad 3)_8$ (Convert each octal number to binary, e.g. $(7)_8$ = $(111)_2$, etc.)

= $(\mathbf{111}\ 000\ 011)_2$

$= (111000011)_2$

b) $(403.06)_8$ to binary:

$(403.06)_8 = (100\ 000\ 011.000\ 110)_2$

$= (100000011.00011)_2$

c) $(3.305)_8$ to binary:

$(3.305)_8 = (011.011\ 000\ 101)_2$

$= (11.011000101)_2$

Exercise 2.8

Convert the following binary numbers to hexadecimal:

a) $(1011111.0011)_2$; b) $(111111.11)_2$; c) $(10000.00011)_2$

Solution:

a) $(101\ 1111.0011)_2$ to hexadecimal:

$(101\ 1111.0011)_2 = (0101\ 1111.0011)_2$

$= (5 \quad F \ . \quad 3)_{16}$

$= (5F.3)_{16}$

(Form groups of 4 digits; add a 0 to the beginning of the given number to form a group of 4 digits; convert each group to a hexadecimal number.)

b) $(111111.11)_2$ to hexadecimal:

$(11\ 1111.11)_2 = (0011\ 1111.1100)_2$

$= (3 \quad F \ . \quad C)_{16}$

= $(3F.C)_{16}$

(Form groups of 4 digits; add 0s to the beginning and end of the given number to form groups of 4 digits; convert each group to a hexadecimal number.)

c) $(10000.00011)_2$ to hexadecimal:

$(1\ 0000.0001\ 1)_2$ = $(0001\ 0000.0001\ 1000)_2$

= $(1 \quad 0\ .\ 1 \quad 8)_{16}$

= $(10.18)_{16}$

Exercise 2.9

Convert the following hexadecimal numbers to binary:

a) $(3F.C)_{16}$; b) $(50.72)_{16}$; c) $(62.A)_{16}$

Solution:

a) $(\mathbf{3}F.C)_{16}$ to binary:

$(\mathbf{3}F.C)_{16}$ = $(\mathbf{0011}\ 1111.1100)_2$ (Convert each hexadecimal number to binary, e.g. $(\mathbf{3})_{16}$ = $(\mathbf{0011})_2$, etc.)

= $(111111.11)_2$

b) $(50.72)_{16}$ to binary:
$(50.72)_{16}$ = $(0101\ 0000.0111\ 0010)_2$

= $(1010000.011100)_2$

c) $(62.A)_{16}$ to binary:
$(62.A)_{16}$ = $(0110\ 0010.1010)_2$

= $(1100010.101)_2$

Exercise 2.10

Convert the following hexadecimal numbers to octal:

a) $(3F.C)_{16}$; b) $(50.72)_{16}$; c) $(62.A)_{16}$

Solution:

a) $(3F.C)_{16}$ to octal:

$(3F.C)_{16} = (0011\ 1111.1100)_2$ (Convert each hexadecimal number to binary.)

$= (\mathbf{111}\ 111.110)_2$ (Regroup to form groups of 3.)

$= (\mathbf{7}\quad 7\ .\quad 6)_8$ (Convert each binary group to octal, e.g. $(\mathbf{111})_2 = (\mathbf{7})_8$.)

$= (77.6)_8$

b) $(50.72)_{16}$ to octal:

$(50.72)_{16} = (0101\ 0000.0111\ 0010)_2$

$= (001\ 010\ 000.011\ 100\ 100)_2$

$= (1\quad 2\quad 0\ .\quad 3\quad 4\quad 4)_8$

$= (120.344)_8$

c) $(62.A)_{16}$ to octal:

$(62.A)_{16} = (0110\ 0010.1010)_2$

$= (001\ 100\ 010.101)_2$

$= (1\quad 4\quad 2\ .\ 5\ )_8$

$= (142.5)_8$

Exercise 2.11

Convert the following octal numbers to hexadecimal:

a) $(7123)_8$; b) $(213.56)_8$; c) $(23.02)_8$

Solution:

$(7123)_8$ to hexadecimal:

$(7123)_8 = (111\ 001\ 010\ 011)_2$ (Convert each octal number to binary.)

$= (\mathbf{1110}\ 0101\ 0011)_2$ (Regroup to form groups of 4.)

$= (\mathbf{E}\quad 5\quad 3\ )_{16}$ (Convert each binary group to hexadecimal, e.g. $(\mathbf{1110})_2 = (\mathbf{E})_{16}$.)

$= (E53)_{16}$

a) $(213.56)_8$ to hexadecimal:

$(213.56)_8 = (010\ 001\ 011\ .\ 101\ 110)_2$

$= (1000\ 1011\ .\ 1011\ 1000)_2$ (Add two 0s at the end to form a group of 4 digits.)

$= (8\quad B\ .\quad B\quad 8)_{16}$

$= (8B.B8)_{16}$

b) $(23.02)_8$ to hexadecimal:

$(23.02)_8 = (010\ 011\ .\ 000\ 010)_2$

$= (0001\ 0011\ .\ 0000\ 1000)_2$ (Add two 0s at the beginning and end.)

$= (1\quad 3\ .\quad 0\quad 8\ )_{16}$

$= (13.08)_{16}$

Chapter 3

Data Storage

Exercise 3.1

Write the following numbers in normalised form:

a) 1235.8 (dec.); b) 0.056×10^2 (dec.); c) $(1111.01)_2$ d) $(0.0111001)_2$

Solution:

a) 1235.8 = 1.2358x102
b) $0.056 \times 10^2 = 5.6$
$(1111.01)_2 = (1.11101)_2 \times (2^3)_{10}$ (Sign: +; Exponent: 3; Mantissa: 11101)

c) $(0.0111001)_2 = (1.11001)_2 \times (2^{-2})_{10}$ (Sign: -; Exponent: -2; Mantissa: 11001)

Chapter 4:

Operations on data

Exercise 4.1

Calculate:

a) $(101)_2 + (101)_2$ b) $(1100)_2 + (1010)_2$ c) $(1111)_2 + (1001)_2$

(These are unsigned numbers.)

Solution:

First look at a simple example in decimal: 19 + 2 = 21 (The base is 10, i.e. ten.)

Place values:	10^1	10^0
Carry:	*1*	
Augend:	*1*	*9*
Addend:	*0*	*2*
Sum:	*2*	*1*

Explanation:

Column 10^0*:* *9 + 2 = 1 + 10 (i.e. 1 + carry of 1 to the next column. The base is 10, so a carry takes place when we get a 10.)*

Column 10^1*:* *(carry of 1) + 1 + 0 = 2*

Addition in binary is similar to addition in decimal - the base of the number system in which the addition is being done must be taken into account.

Addition in binary (the base is 2_{10}, i.e. two):

a)		b)		c)	
	101		1100		1111
	+ 101		+ 1010		+ 1001
	1010		10110		11000

Example:

Place values:	2^3	2^2	2^1	2^0
Carry:	*1*	*1*	*1*	
Augent:	*0*	*1*	*1*	*1*
Addent:	*0*	*1*	*0*	*1*
Sum:	*1*	*1*	*0*	*0*

Explanation:

Column 2^0: $1 + 1 = 2_{10} = 0 + (10)_2$ *(i.e. 0 + carry of 1 to the next column. When we get a 2, a carry takes place.)*

Column 2^1: *(carry of 1)* $+ 1 + 0 = 2_{10} = 0 + (10)_2$ *(i.e. 0 + carry of 1 to the next column)*

Column 2^2: *(carry of 1)* $+ 1 + 1 = 3_{10} = 1 + 2_{10} = 1 + (10)_2$ *(i.e. 1 + carry of 1 to the next column)*

Column 2^3: *(carry of 1)* $+ 0 + 0 = 1$ *peripheral*

Supplement to Appendix E:

Boolean algebra & Logic circuits

1. Boolean algebra (pp. 563-574(ed.4), pp. 543-554(ed.3))

Exercises and/or explanations material are provided in the following sections:

1.1 Binary logical expressions and operators

1.2 Truth tables for Boolean functions

1.3 Boolean rules

1.4 Application of Boolean rules

1.5 Algebraic simplification of Boolean functions

1.6 Boolean function transformation to sum of products (minterms) form

1.7 Karnaugh maps (diagrams)

1.8 Forming groups in Karnaugh maps

1.9 Simplification of Boolean functions by using Karnaugh maps

1.1 Binary logical expressions and operators (pp. 563-566 (ed.4), pp. 543-546(ed.3))

Exercise E.1

Consider the following binary variables:

A = 1, B = 0, C = 1, D = 0

Calculate the logical values of the following expressions:

a) A·B; b) A+B; c) A+C; d) A′; e) D′; f) A·C; g) B+D. (Refer to the tables in Figure E.1, p. 564 (ed.4), p. 544 (ed.3))

Solution:

a) A·B = 1·0 = 0

b) A+B = 1+0 = 1

c) A+C = 1+1 = 1

d) A′ = 1′ = 0

e) D′ = 0′ = 1

f) $A \cdot C$ = $1 \cdot 1$ = 1

g) $B+D$ = $0+0$ = 0

Exercise E.2

Given that A = 0, B = 1, C = 1 and D = 0, evaluate the following expressions:

a) $(A + B)A$
b) $A' + B \cdot D$
c) $A' + A'$
d) $(A + B)'$

e) $AB' + (B + D)$

Solution:

A = 0, B = 1, C = 1, D = 0

a) $(A+B)A$
$=(0+1)0$
$=1 \cdot 0$
$=0$

b) $A'+BD$
$=0'+1 \cdot 0$
$=1+0$
$=1$

(c) $A'+A'$
$=0'+ 0'$
$=1+1$
$=1$

d) $(A+B)'$
$=(0+1)'$
$=1'$
$=0$

e) $AB'+(B+D)$
$=0 \cdot 1'+ (1+0)$
$=0 \cdot 0 + 1$
$=0 + 1=1$

f) Evaluate the expression (AB+AC') · (A' BC+AC)', for A=1, B=1, C=0.

Solution:

$(AB + AC') \cdot (A' BC + AC)'$

$= (1 \cdot 1 + 1 \cdot 0') \cdot (1' \cdot 1 \cdot 0 + 1 \cdot 0)'$

$= (1 + 1) \cdot (0 + 0)'$

$= 1 \cdot 0'$

= 1 · 1

= 1

1.2 Truth tables for Boolean functions (p. 571 (ed.4), p. 551 (ed.3))

Exercise E.3

a) Use a truth table to show that x + x'y = x + y.

Solution:

x	**y**	**x + x'y**	**x + y**
0	0	0 + 1 · 0 = **0**	0 + 0 = **0**
0	1	0 + 1 · 1 = **1**	0 + 1 = **1**
1	0	1 + 0 · 0 = **1**	1 + 0 = **1**
1	1	1 + 0 · 1 = **1**	1 + 1 = **1**

The results in the final two columns are equivalent, hence x + x'y = x + y.

b) Use truth tables to show that the following Boolean expressions are equivalent:

X = A′B′C + AB′C′ + AB′C and Y = AB′ + B′C

Solution:

Boolean expressions are equivalent if their final outputs are the same. This means that the final columns in the truth tables of these expressions must have identical inscriptions.

Let X = A′B′C + AB′C′ + AB′C

A	B	C	A′B′C	AB′C′	AB′C	X
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	0	0

Let Y = AB′ + B′C

A	B	C	AB′	B′C	Y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	0

The entries in the two final columns are identical, so X is equivalent to Y.

This means that $A'B'C + AB'C' + AB'C = AB' + B'C$.

1.3 Boolean rules (p. 568 (ed.4), p. 548(ed.3))

Forouzan divide Boolean rules into three categories: *axioms*, *theorems* and *identities*. Not all Boolean algebra rules are named in **Forouzan**. We provide the following table with Boolean rules:

BOOLEAN RULES	
(i) **Associative rule** (a) $x + (y + z) = (x + y) + z$ (b) $x(yz) = (xy)z$ (We usually write xy instead of $x \cdot y$)	(ii) **Commutative rule** (a) $x + y = y + x$ (b) $xy = yx$
(iii) **Distributive rule** (a) $x + (yz) = (x + y)(x + z)$ (b) $x(y + z) = xy + xz$	(iv) **Identity rule** There exist two elements 0 and 1 such that: (a) $x + 0 = x$ (b) $x \cdot 1 = x$
(v) **Complement rule** (a) $x + x' = 1$ (b) $xx' = 0$	(vi) **Idempotence** (a) $x + x = x$ (b) $x \cdot x = x$
(vii) **Absorption** (a) $x + (xy) = x$ (Not provided in **Forouzan**.) (b) $x (x + y) = x$ (Not provided in **Forouzan**.) (c) $x (x'+ y) = xy$ OR $x'(x+ y) = x'y$ (d) $x + (x' \cdot y) = x + y$ (Similar to (e).) (e) $x' + (x \cdot y) = x' + y$ (Not provided in **Forouzan**.)	(viii) **Null (Zero) or Boundedness** (a) $x + 1 = 1$ (b) $x \cdot 0 = 0$
(ix) **De Morgan** (a) $(x + y)' = x'y'$ (b) $(xy)' = x' + y'$	(x) **Double negative or Involution** $(x')' = x$

1.4 Application of Boolean rules (p. 571 (ed.4), p. 551(ed.3))

Boolean rules can be expanded. Note: It is not necessary to provide the names of rules that you apply in your solutions. We discuss a few rules provided in the above table.

(ii) **Commutative**

(a) $x + y = y + x$

(b) $xy = yx$

Commutativity identity Rule (a) applied:

$xy + yw = yw + xy$

(ix) **De Morgan**

(a) $(x + y)' = x'y'$

(b) $(xy)' = x' + y'$

De Morgan's theorem applied**:**

Example: Apply De Morgan's theorem to the following Boolean expression: $(VWT + V' T')'$

De Morgan's theorem, Rule (a): $(a + b)' = a' b'$

Let $VWT = a$, and $V'T' = b$,

then $(VWT + V' T')'$

$= (a + b)'$

$= a'b'$ *De Morgan*

$= (VWT)' (V' T')'$.

It is possible to apply De Morgan's theorem again:

De Morgan's theorem, Rule (b): $(mn)' = m' + n'$. Note: $(mn)' \neq m'n'$.

We can expand this theorem: $(mns)' = m' + n' + s'$

Back to the example: Simplify $(VWT)'\,(V'\,T'\,)'$

Consider $(VWT)'$: $(VWT)' = V' + W' + T'$ $\underline{1}$

Consider $(V'\,T')'$: Let $V' = m$ and $T' = n$, then

$(V'\,T')'$

$= (mn\,'$

$= m' + n'$

$= (V')' + (T')'$

$= V + T$ $\underline{2}$ *Involution*

From $\underline{1}$ and $\underline{2}$ it follows that $(VWT + V'\,T')' = (VWT)'\,(V'\,T')' = (V' + W' + T')(V + T)$.

It is possible to simplify the final expression - try this!

(vii) **Absorption**

(a) $x + (xy) = x$ (Not provided in **Forouzan**.)

(b) $x\,(x + y) = x$ (Not provided in **Forouzan**.)

(c) $x\,(x' + y) = xy$

(d) $x + (x' \cdot y) = x + y$

(e) $x' + (x \cdot y) = x' + y$ (Not provided in **Forouzan**. Similar to (d).)

We look at Rule (a): $\underline{x} + \underline{x}y = \underline{x}$

($\underline{x}$ stands alone and is present in the second term $\underline{x}y$. The second term falls away.)

Absorption rule applied:

$(\underline{vw'} + \underline{vw'}yu) + \underline{vw'}u = \underline{vw'} + \underline{vw'}u = \underline{vw'}$ (Apply the rule twice.)

We look at Rule (e): $\underline{x'} + (\underline{x} \cdot y) = \underline{x'} + y$ (The x disappears from the second term.)

The absorption rule applied:

$\underline{x}' + \underline{x}uv'w + \underline{x}y'$ (x' stands alone and x is present in the other terms.)

$= \underline{x}' + \underline{x}(uv'w + y')$ *distributive*

$= \underline{x}' + uv'w + y'$ (The x disappears from the second term.)

1.5. Algebraic simplification of Boolean functions (p. 571 (ed.4), p. 551 (ed.3))

Exercise E.4

Simplify the following Boolean functions algebraically:

a) $xy'z' + xy'z + x'yz + x'yz'$
b) $A'B'C + AB'C' + AB'C + ABC' + ABC$
c) $((x'+y'+z)' + (x+y') + z')'$
d) $(x' + y)(y' + z)(x + z')'$
e) $((A' + B' + C)' + (A + B')' + C')'$
f) $xyz + xy'z + x'yz' + x'y'z'$
g) $(s + pq'r)' + r' + rp' + rp's$
h) $(pqr' + rs)' + (p'r + prs)'$

Solution:

a) $xy'z' + xy'z + x'yz + x'yz'$

$= xy'(z' + z) + x'y(z + z')$ *Distributive*

$= xy' + x'y$ *Complement*

b) $A'B'C + AB'C' + AB'C + ABC' + ABC$

$= A'B'C + AB'(C'+C) + AB(C'+C)$ *Distributive*

$= A'B'C + AB' + AB$

Complement

$= A'B'C + A(B' + B)$ *Distributive*

$= A'B'C + A$ *Complement*

$= A + A'B'C$ *Commutative*

$= A + A'(B'C)$ *Associative*

$= A + B'C$ *Absorption*

c) $((x'+y'+z)' + (x+y') + z')'$

$= ((x'+y'+z)')' \cdot (x+y')' \cdot z''$ *De Morgan*

$= (x'+y'+z)\cdot(x+y')'\cdot z$
Double negative

$= (x'+y'+z)\cdot x'y''\cdot z$ *De Morgan and double negative*

$= x'x'yz + y'x'yz + zx'yz$ *Distributive*

$= x'yz + y'yx'z + x'yz$ *Idempotence, commutative*

$= x'yz + 0\cdot x'z + x'yz$ *Complement*

$= x'yz + 0 + x'yz$ *Null*

$= x'yz + x'yz$ *Identity*

$= x'yz$ *Idempotence*

c) $(x' + y)(y' + z)(x + z')'$
$= x' y' + x' z + y y' + yz) (x' z)$ *Distributive, De Morgan, involution*

$= (x' y' + x' z + 0 + yz) (x' z)$ *Complement*

$= x' y' z + x' z + x' zy$ *Distributive, idempotent*

$= x' z (y' + 1 + y)$ *Distributive*

$= x' z$ *Null, identity*

d) $((A' + B' + C)' + (A + B')' + C')'$

$= (A' + B' + C) \cdot (A + B') \cdot C$ *De Morgan, involution*

$= (A' + B' + C) \cdot (AC + B'C)$ *Distributive*

$= A'AC + A'B'C + B'AC + B'B'C + CAC + CB'C$ *Distributive*

$= 0 + A'B'C + B'AC + B'C + AC + B'C$ *Complement,,null,idempotence, commutative*

$= A'B'C + AB'C + B'C + AC$ *Distributive, idempotence*

$= B'C \cdot (A' + A + 1) + AC$ *Distributive, commutative*

$= B'C + AC$ *Null, identity*

e) $xyz + xy'z + x'yz' + x'y'z'$

$= xz(y+y') + x'z'(y+y')$ *Distributive*

$= xz(1) + x'z'$ *Complement*

$= xz + x'z'$ *Identity*

g) $(s + pq'r)' + r' + rp' + rp's$

$=$ $\mathbf{s' \cdot (pq'r)'} + r' + rp' + rp's$ ***de Morgan**.*

$= s' \cdot (pq'r)' + \underline{r'} + \underline{r}(p' + p's)$ *Commutative (Consider $\underline{r'} + \underline{r(}p' + p's$: $\underline{r'}$ stands alone and $\underline{r}$ is present in the other term, so we can use the absorption rule (e).)*

$= s'$ $\mathbf{(p' + q'' + r')}$ $+ \underline{r'} + p' + p's$ ***de Morgan**, absorption*

$= s'p' + s'q + s'r' + r' + p' + p's$ *Distributive, involution*

$= (s'\underline{p'} + \underline{p'} + \underline{p}'s) + (s'\underline{r'} + \underline{r'}) + s'q$ *Associative (Consider $s'\underline{p'} + \underline{p'} + \underline{p}'s$: $\underline{p'}$ stands alone and p' is present in the terms, $s'\underline{p'}$ and $\underline{p}'s$, and consider $s'\underline{r'} + \underline{r'}$: $\underline{r'}$ stands alone and r' is present in another term,* *s'r', so we can apply the absorption rule (a) in both cases.)*

= p' + r' + s'q A*bsorption*

h) (pqr' + rs)' + (p'r + prs)'

= (pqr')' · (rs)' + (r · (<u>p'</u> + <u>p</u>s))' *de Morgan, distributive*

= (p' + q' + r'') · (r' + s') + (r · (**<u>p'</u> + s**))' *de Morgan, **absorption rule (e)***

= (p'r' + p's' + q'r' + q's' + rr' + rs') + (**r' + (p' + s)'**) *Involution, distributive, **de Morgan***

= p'r' + p's' + q'r' + q's' + 0 + rs' + r' + **p''s'** *Complement, **de Morgan***

= p'r' + q'r' + r' + p's' + ps'+ q's' + rs' *Commutative, involution, identity*

= (p'<u>r'</u> + q'<u>r'</u> + <u>r'</u>) + s'(p' + p + q' + r) *Distributive, associative*

= <u>r'</u> + s'(1 + q' + r) *Absorption rule (a), complement*

= (r' + s') · 1 *Null*

= r' + s' *Identity*

1.6 Boolean function transformation to sum of products (minterms) form (p. 572 (ed.4), p. 552 (ed.3))

Note: **The product of sums method (p. 570 (ed.4), p. 550 (ed.3)) is excluded from the syllabus.)**

We can use the **algebraic** method using Boolean rules or a **truth table** to **transform a Boolean function** into an expression consisting of **minterms**. A minterm is a product of all variables in a Boolean function in which each variable appears only once (the variables may occur in complemented form).

We first **transform a Boolean function algebraically to a sum of products (minterms)**. (This is additional study material.)

We write Boolean expressions in sum of minterms form with the aid of Boolean rules discussed earlier. Consider the expression F(A,B,C) = A′C + AB′. This expression is **not** in sum of minterms form because **three variables are present in the function and each variable must occur in every minterm**. We see that the variable B (or B′) does not occur in the first term and C (or C′)

does not occur in the second term. We can use the complement rule (x + x′ = 1) to incorporate these variables.

$F = A'C + AB'$

$= A'(B+B')C + AB'(C+C')$ $(B+B'=1 \text{ and } C+C'=1)$

$= (A'B+A'B')C + AB'C + AB'C'$

$= A'BC + A'B'C + AB'C + AB'C'$ (sum of minterms form)

$= m_3 + m_1 + m_5 + m_4$ (m-notation) (m-notation is described on the next page.)

Note that the **order** in which the **variables** appear in each minterm of F(A,B,C) must be the **same for all minterms**. That is, A (or A′) occurs first, then B (or B′) and then C (or C′).

Secondly we use a **truth table to transform a Boolean function to a sum of products (minterms)**.

We use the **sum of products** (**minterms**) method to change a truth table into a Boolean expression in which each term is called a **minterm**.

Consider the following table:

x	y	F	minterms	m-notation
0	0	0	x′y′	m_0
0	1	1	x′y	m_1
1	0	1	xy′	m_2
1	1	0	xy	m_3

The entries in the table called **minterms** were obtained in the following way: Each minterm contains **both variables x and y** (Order is important: first x then y.). How did we decide whether to write xy, x′y′, x′y or xy′? The variables occur in **complemented** form if the **value of the variable**

is 0 in the row concerned; the variable is **not complemented** if its **value is 1** in the row concerned. The minterm in the case where x=1 and y=0 is thus xy', and $x'y'$ where x=0 and y=0, et cetera.

We give the minterms names: m_0, m_1, m_2, and m_3 i.e. **m-notation**:

$x'y'$ = $\mathbf{m_0}$ (the x = 0, y = 0 row; $(00)_2 = 0$)

$x'y$ = $\mathbf{m_1}$ (the x = 0, y = 1 row; $(01)_2 = 1$)

xy' = $\mathbf{m_2}$ (the x = 1, y = 0 row; $(10)_2 = 2$)

xy = $\mathbf{m_3}$ (the x = 1, y = 1 row; $(11)_2 = 3$)

m_0, m_1, m_2 and m_3 are assigned as follows:

For the $x'y'$ minterm, x=0 and y=0. This is the m_0 minterm. For the xy minterm, x=1 and y=1. This is the m_3 minterm because $(11)_2 = (\mathbf{3})_{10}$. The subscript of a minterm is derived from the binary value of x and y.

The expression F is obtained by writing down the sum of the minterms where F is equal to 1:

$F = x'y + xy'$ (row 2 and row 3).

$= m_1 + m_2$ (m-notation)

F is now in **sum of minterms** or **sum of products** form.

Note: Four (2^2) minterms are formed for two variables. In general there are 2^n minterms for n variables.

Minterms for 4 variables (A, B, C and D):

A	B	C	D	Minterms	m-notation
0	0	0	0	A′B′C′D′	m_0
0	0	0	1	A′B′C′D	m_1
0	0	1	0	A′B′CD′	m_2
0	0	1	1	A′B′CD	m_3
0	1	0	0	A′BC′D′	m_4
0	1	0	1	A′BC′D	m_5
0	1	1	0	A′BCD′	m_6
0	1	1	1	A′BCD	m_7
1	0	0	0	AB′C′D′	m_8
1	0	0	1	AB′C′D	m_9
1	0	1	0	AB′CD′	m_{10}
1	0	1	1	AB′CD	m_{11}
1	1	0	0	ABC′D′	m_{12}
1	1	0	1	ABC′D	m_{13}
1	1	1	0	ABCD′	m_{14}
1	1	1	1	ABCD	m_{15}

For example, for the ABCD minterm, A=1, B=1, C=1 and D=1. It is therefore the m_{15}-minterm because $(1111)_2 = (15)_{10}$.

Exercise E.5

Write the following expressions in sum of products (minterms) form by using Boolean algebra:

$F_1(A,C) = A+C$; $F_2(A,B,C,D) = ABC + CD$; $F_3(X,Y,Z) = XY + XZ$

Solution:

$F_1(A,C) = A+C$

$= A(C+C') + (A+A')C$

$= AC + AC' + AC + A'C$

$= AC + AC' + A'C$

$= m_3 + m_2 + m_1$

$F_2(A,B,C,D) = ABC + CD$

$= ABC(D+D') + (A+A')(B+B')CD$

$= ABCD + ABCD' + (AB+AB'+A'B+A'B')CD$

$= ABCD + ABCD' + ABCD + AB'CD + A'BCD + A'B'CD$

$= ABCD + ABCD' + AB'CD + A'BCD + A'B'CD$

$= m_{15} + m_{14} + m_{11} + m_7 + m_3$

$F_3(X,Y,Z) = XY + X'Z$

$= XY(Z+Z') + X'(Y+Y')Z$

$= XYZ + XYZ' + X'YZ + X'Y'Z$

$= m_7 + m_6 + m_3 + m_1$

1.7 Karnaugh maps (diagrams) (pp. 572-574 (ed.4), pp. 552-554 (ed.3))

Now that we know how to write Boolean functions in sum of minterms (products) form, we can proceed with the simplification of these functions by using Karnaugh maps.

A Boolean function can be written in sum of minterms form and we can make use of this property to draw a **Karnaugh map (diagram)** with which to simplify the function. A Karnaugh map is a graphic representation of a Boolean function where squares are used to represent minterms. If a particular **square** in the map contains a '**1**', it means that the **minterm represented by that square is included in the function.** In other words, the Boolean function consists of the 'sum' of the minterms indicated by the squares.

The Karnaugh maps for two, three and four variables are as follows:

	B'	B
A'	m_0	m_1
A	m_2	m_3

A two-variable Karnaugh map

	$B'C'$	$B'C$	BC	BC'
A'	m_0	m_1	m_3	m_2
A	m_4	m_5	m_7	m_6

A **three-variable Karnaugh map**

	C′D′	C′D	CD	CD′
A′B′	m_0	m_1	m_3	m_2
A′B	m_4	m_5	m_7	m_6
AB	m_{12}	m_{13}	m_{15}	m_{14}
AB′	m_8	m_9	m_{11}	m_{10}

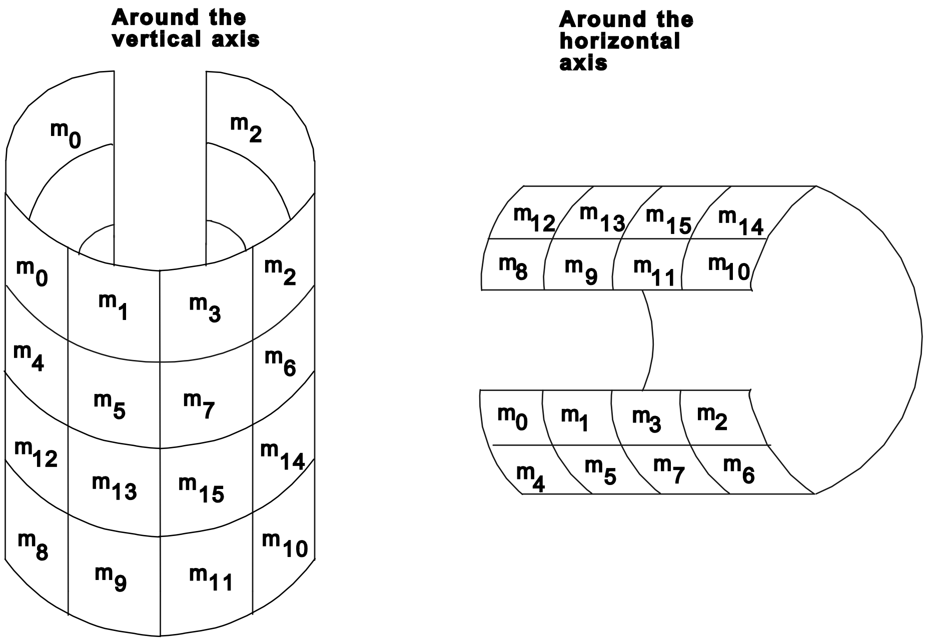
A four-variable Karnaugh map

We already know that there are 2^n minterms for n variables. The two, three and four-variable Karnaugh maps therefore consist of 4, 8 and 16 squares respectively. Each square represents a minterm. In order to find the minterm that corresponds to a specific square, one has to write down the product of the variables to the left of the row and at the top of the column indicating the square.

Any two adjacent squares in a Karnaugh map differ only in respect of one variable, which is complemented in one square and not complemented in the other. It is clear that, for example, B′C and BC′ cannot be two adjacent column headings since they differ in two respects.

The ‘wrap-around’ principle:

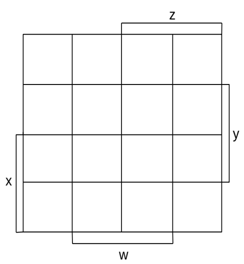
Note that m_0 and m_2, m_4 and m_6, m_{12} and m_{14}, m_8 and m_{10}, as well as m_0 and m_8, m_1 and m_9, et cetera, are adjacent. One has to visualize the diagram as a ‘round’ one, as depicted in the following figure:



Karnaugh maps on horizontal and vertical axes

Different notations can be used for Karnaugh maps. Look at the notations used in **Forouzan** , (pp. 572, 573 (ed.4), pp. 552, 553 (ed.3)).

The following four-variable notation for a Karnaugh map is not provided in **Forouzan**:



The rows/columns for x, y, z and w are shown in the map. It is also the case that in the top two rows, x′ is represented, in the top and bottom rows, y′ is represented, in the two left hand side columns, z′ is represented, and in the leftmost and rightmost two columns, w′ is represented.

Exercise E.6

Draw Karnaugh maps for:

(a) $F_1(X,Y) = X'Y + XY' + XY$; (b) $F_2(A,B,C) = AB' + ABC$;

(c) $F_3(X,Y,Z) = X'YZ + XYZ' + X'YZ' + X'Y'Z'$; (d) $F_4(A,B,C,D) = m_7 + m_{13} + m_{11} + m_3 + m_2 + m_0$

Solution:

(a) $F_1(X,Y) = X'Y + XY' + XY$

	Y′	Y
X′		1
X	1	1

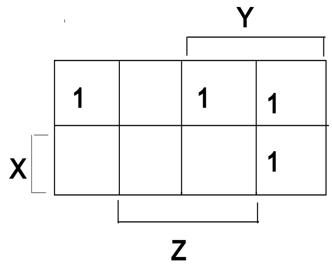
(b) $F_2(A,B,C) = AB' + ABC$

$= AB'(C + C') + ABC$

$= AB'C + AB'C' + ABC$

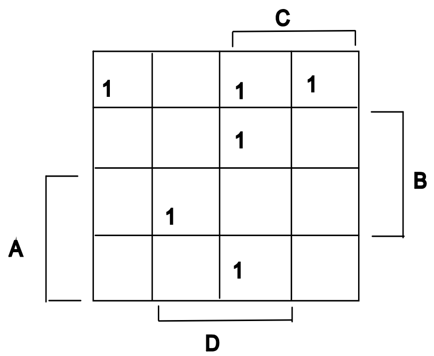
	B′C′	B′C	BC	BC′
A′				
A	1	1	1	

(c) $F_3(X,Y,Z) = X'YZ + XYZ' + X'YZ' + X'Y'Z'$



d) $F_3(A,B,C,D) = m_7 + m_{13} + m_{11} + m_3 + m_2 + m_0$

$= A'BCD + ABC'D + AB'CD + A'B'CD + A'B'CD' + A'B'C'D'$



1.8 Forming groups in Karnaugh maps

Minterms can be grouped in some way in a Karnaugh map and then the simplified terms for a Boolean function can be derived from the groups in the map. The following rules must be followed when groups are formed:

- Group the marked squares (1s) in a map into groups of 8, 4, 2 or 1. (3, 5, 6 or 7 ones are not permissible.) Groups are formed by grouping **adjacent** squares, in other words groups are formed horizontally or vertically. We **cannot** put only m_1 and m_7 together in a group, for example.

- Make the groups as large as possible to obtain the simplest expression.

- With each new group that is formed, one or more minterm(s) that was/were not grouped before, must be included. (In this way we try to form the least number of groups.)

- **The 'wrap-around' principle:** The minterms m_0, m_4, m_2 and m_6 in the 3-variable Karnaugh map are seen as adjacent. The same principle applies for the 4-variable Karnaugh map. For example, we may group m_1 and m_9 together.

- A minterm may be included in more than 1 group [x + x + .. + x = x (idempotence)].

1.9 Simplification of Boolean functions by using Karnaugh maps

Two methods can be used:

- Algebraic simplification of groups of adjacent terms.
- Derive simplified terms directly from the Karnaugh map.

We look at these two methods.

Algebraic simplification of adjacent terms in a Karnaugh map:

Look at a **three-variable** map:

	B′C′	B′C	BC	BC′
A′	m_0	m_1	m_3	m_2
A	m_4	m_5	m_7	m_6

m_3 = A′BC and m_2 = A′BC′ differ only in respect of the variable C. C is not complemented in m_3, whilst in m_2 it is. It follows from the rules of Boolean algebra that the **sum of two minterms** in adjacent squares can be **simplified algebraically** to a single product. Look at m_2 and m_3 again:

$m_3 + m_2$ =A′BC + A′BC′

=A′B (C+C′) *Distributive*

=A′B Complement

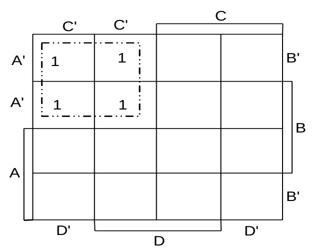
In a **four-variable** Karnaugh map the following holds:

- If two adjacent squares are marked with a 1, these squares reduce to a single term with **three** variables;
- If four adjacent squares are marked, these squares reduce to a single term with **two** variables;
- If eight adjacent squares are marked, they reduce to a single term with **one** variable;
- And naturally, if all sixteen squares are marked, the expression is equal to 1.

Derive the simplified term of a group of adjacent minterms directly from a Karnaugh diagram:

It is possible to group adjacent minterms of a Karnaugh map and then **algebraically** determine the **simplified form** as shown in the previous example. We look at an **alternative method** whereby we **group adjacent minterms** in a Karnaugh map together in **groups** of **1, 2, 4, 8 or 16** and then derive the terms of the **simplified expression directly from the map** without using Boolean algebraic manipulations.

Suppose you are asked to determine (without using algebraic manipulations) the simplified form of the function represented in the following Karnaugh diagram:



Two possible methods can be used to find the representative term of a group of minterms:

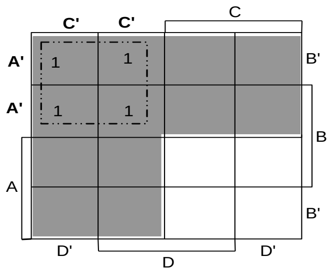
- Determine the **intersection of the variable domains**. (pp. 572-573 (ed.4), pp. 552-553(ed.3))

- Determine **which variables** in the map should be **present in the representative term**. If a **whole group** occurs in the **domain of a variable**, the **variable is included in the representative term**. If a group occurs partly in the domain of a variable (say A) and partly in the domain of its complement (A′), the variable or its complement cannot be part of the representative term.

Solution:

First method: Determine the **intersection of the variable domains**.

We highlight the two domains in which the group lies:



The top two rows represent domain A′ and the leftmost two columns represent domain C′. The **group of 1s** in the map is the **intersection of A′ and C′ domains**, which is represented as **A′C′**.

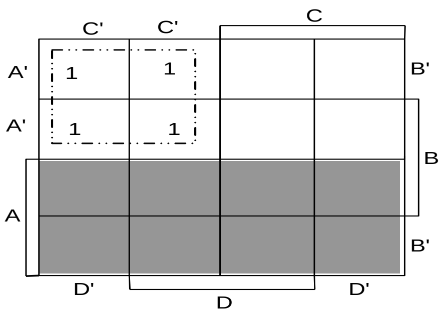
Thus the simplified term representing the group is A′C′.

Second method: Determine **which variables** in the map should be **present in the representative term**.

All the variables in the diagram must be considered in order to decide whether the variable or its complement or neither must be part of the representative term.

Consider domain A:

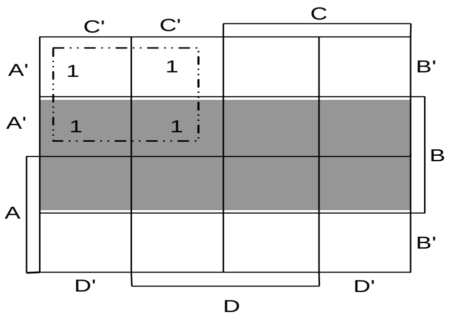
Domain A is represented in the bottom two rows and domain A′ in the top two rows.



The group is in domain A′, so A′ is part of the representative term.

Consider domain B:

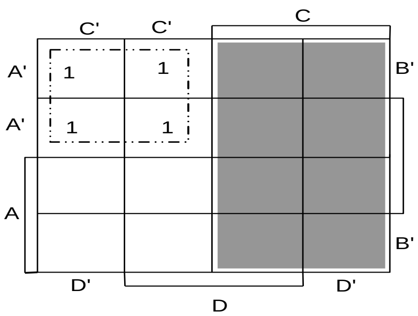
Domain B is represented in the middle two rows and domain B′ in the top and bottom rows.



The group is partly in domain B and partly in domain B′, so neither B nor B′ is part of the representative term.

Consider C:

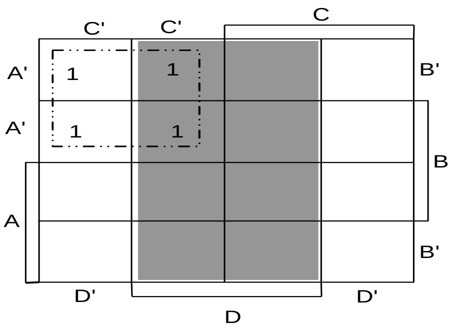
Domain C is represented in the rightmost two columns and domain C′ is in the leftmost two columns.



The group is in domain C′, so C′ is part of the representative term.

Consider D:

Domain D is represented in the middle two columns and domain D′ in the leftmost and rightmost columns.



The group is partly in domain D and partly in domain D′, so neither D nor D′ is part of the representative term.

Finally, only A′ and C′ is part of the term, thus the term representing the group is A′C′.

In summary:

In order to simplify Boolean functions with the aid of Karnaugh maps:

- Write the function in sum of minterms form.
- Draw the Karnaugh map.
- Place a 1 in all the squares that represent minterms which occur in the function.
- Arrange adjacent squares into groups of 1, 2, 4, 8, et cetera, but form the biggest and least possible number of groups.
- Apply some method of simplification described above.

Exercise E.7

Simplify the following Boolean functions with the aid of Karnaugh maps:

(Derive the terms of the functions directly from the relevant maps without making use of algebraic manipulations or truth tables.)

(a) $F_1(x,y,z) = x'y'z' + x'yz' + xyz' + xyz$;

(b) $F_2(x,y,z,w) = xy + x'yz'w + x'yzw + xy'z'w'$;

(c) $F_3(x,y,z) = m_0 + m_2 + m_3 + m_4 + m_6 + m_7$;

(d) $F_4(A,B,C,D) = m_0 + m_1 + m_2 + m_4 + m_5 + m_6 + m_8 + m_{10}$;

(e) $F_5(A,B,C,D) = m_0 + m_1 + m_2 + m_4 + m_5 + m_6 + m_8 + m_9 + m_{12} + m_{13} + m_{14}$;

(f) $F_6(A,B,C,D) = m_0 + m_1 + m_6 + m_8 + m_{13} + m_{14} + m_{15}$;

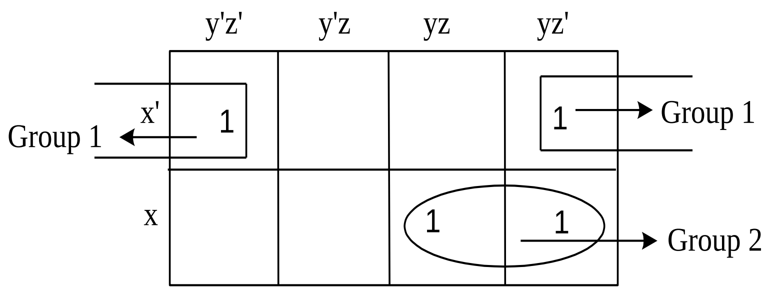
(g) $F_7(A,B,C,D) = B' + ABC'D + A'BD$.

Solution:

(a) $F_1 = x'y'z' + x'yz' + xyz' + xyz$

Note: When you place the variables in the Karnaugh map in a different order, the 1s in the diagram will appear in some other positions, so it therefore advisable to stick to the given order.

Determine the simplified terms of the expression F_1 ***directly*** *from the Karnaugh map (without using algebraic manipulations).*



Note*: We number the groups in no particular order so that we can refer to them in the explanations.*

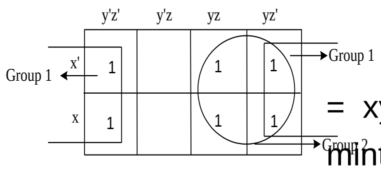
Find the term that represents Group 1:

Group 1 lies in the leftmost and rightmost columns of the diagram. The group lies partly in domain y and partly in domain y' so y or y' cannot be part of the term. The group lies in domains x' and z', so x' and z' must be included in the term. Group 1 is thus represented by the term x'z'.

Find the term that represents Group 2:

Group 2 lies in the bottom row and in the two rightmost columns. The group lies in the intersection of domain x and domain y. However, it lies partly in domain z and partly in domain z', so z or z' cannot be part of the term. Group 2 is thus represented by the term xy.

The simplified form of F_1 derived **directly** from the Karnaugh map: $F_1 = x'z' + xy$



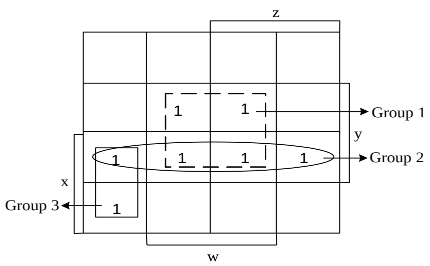
(b) $F_2 = xy + x'yz'w + x'yzw + xy'z'w'$ (F_2 is not in sum of minterms form.)

$= xy(z + z') + x'yz'w + x'yzw + xy'z'w'$

$= xyz + xyz' + x'yz'w + x'yzw + xy'z'w'$

$= xyz(w + w') + xyz'(w + w') + x'yz'w + x'yzw + xy'z'w'$

$= xyzw + xyzw' + xyz'w + xyz'w' + x'yz'w + x'yzw + xy'z'w'$ (Sum of minterms form.)



Group 1 gives the term yw. Group 2 gives the term xy. Group 3 gives the term $xz'w'$.

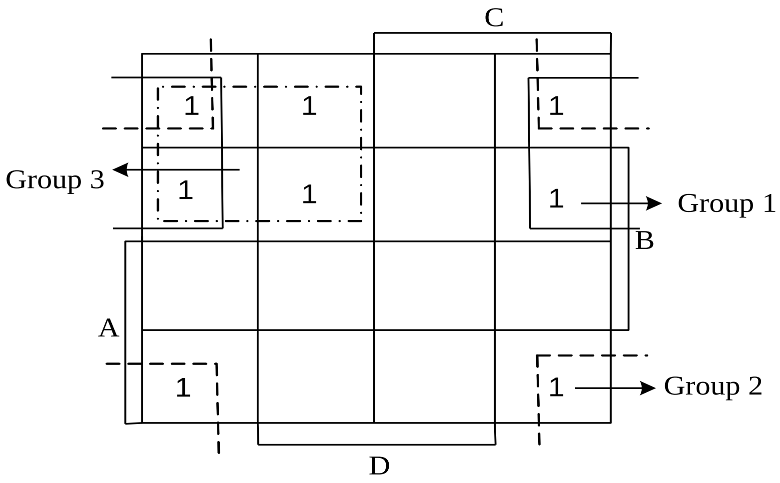
Directly from the map: $F_2 = yw + xy + xz'w'$

(c) $F_3 = m_0 + m_2 + m_3 + m_4 + m_6 + m_7$

Group 1 gives the term z'. Group 2 gives the term y.

Directly from the map: $F_3 = z' + y$

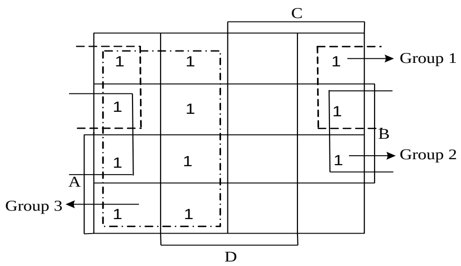
(d) $F_4 = m_0 + m_1 + m_2 + m_4 + m_5 + m_6 + m_8 + m_{10}$



Group 1 gives the term A′D′. Group 2 gives the term B′D′. Group 3 gives the term A′C′.

Directly from the map: $F_5 = A'D' + B'D' + A'C'$

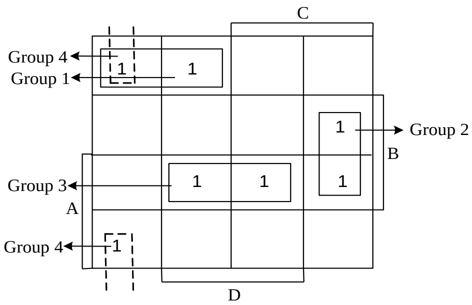
(e) $F_5 = m_0 + m_1 + m_2 + m_4 + m_5 + m_6 + m_8 + m_9 + m_{12} + m_{13} + m_{14}$



Group 1 gives the term A′D′. Group 2 gives the term BD′. Group 3 gives the term C′.

Directly from the map: $F_6 = A'D' + BD' + C'$

(f) $F_6 = m_0 + m_1 + m_6 + m_8 + m_{13} + m_{14} + m_{15}$



Group 1 gives the term A′B′C′ . Group 2 gives the term BCD′. Group 3 gives the term ABD.

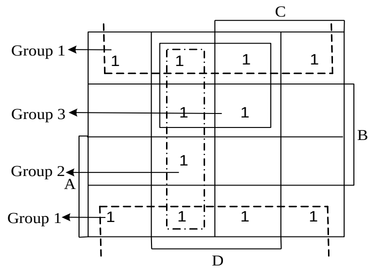
Group 4 gives the term B′C′D′.

Directly from the map: $F_7 = A'B'C' + BCD' + ABD + B'C'D'$

(g) $F_7 = B' + ABC'D + A'BD = (A + A')B' (C + C')(D + D') + ABC'D + A'B(C + C')D$

$= AB'CD + AB'CD' + AB'C'D + AB'C'D' + A'B'CD + A'B'CD' + A'B'C'D + A'B'C'D' + ABC'D + A'BCD + A'BC'D$

$= m_{11} + m_{10} + m_9 + m_8 + m_3 + m_2 + m_1 + m_0 + m_{13} + m_7 + m_5$



Group 1 gives the term B′. Group 2 gives the term C′D. Group 3 gives the term A′D.

Directly from the map: $F_8 = B' + C'D + A'D$

2. Logic circuits (p. 574-575 (ed.4), p. 554–555 (ed.3))

2.1 Combinational logic circuits

2.2 Logically equivalent circuits

2.3 Designing a logic circuit

2.1 Combinational logic circuits (p. 574 (ed.4), p. 554 (ed.3))

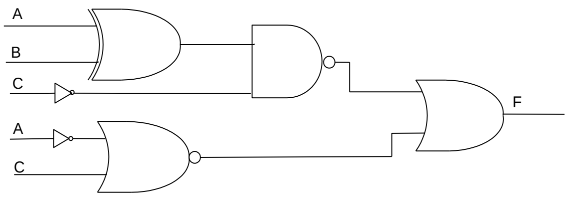
A **combinational circuit** is constructed by using a **combination of logic gates**. The outputs of a combinational circuit at any given time are determined by the current inputs. The operation executed by the combinational circuit is fully specified by a set of logical expressions. (The term ‘logical expression’ is often used when we refer to the Boolean function that describes a logic circuit.)

Exercise E.8

Draw the logic circuit for the following Boolean function (do not simplify the expression):

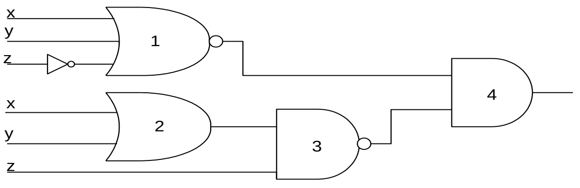
F(A, B, C) = [(A ⊕ B) · C']' + (C + A')' (⊕ denotes the XOR gate, see p. 75 (ed.4), p.76) (ed.3))

Solution:



Exercise E.9

Provide the outputs for Gates 1 - 4 of the following logic circuit:



Solution:

Outputs: Gate 1: (x + y + z')';

Gate 2: (x + y);

Gate 3: ((x + y) · z)';

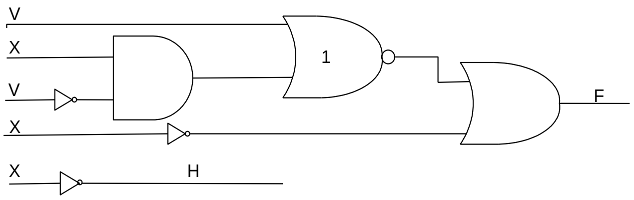
Gate 4: (x + y + z')' · ((x + y) · z)'.

2.2 Logically equivalent circuits

Different **circuits** can be **equivalent**, in other words, the **outputs of the circuits are equal**. This is a very important concept when logic circuits are designed because a more simple circuit can be used if its output is the same as the output of a more complicated circuit.

Exercise E.10

Are the following two logic circuits logically equivalent? Motivate your answer.



Solution:

The output for F is (v + (xv'))' + x' and the output for H is x'.
We can simplify F:

F = (v + (xv'))' + x'

= v'·(xv')' + x' *de Morgan*

= v'(x' + v'') + x' *de Morgan*

= v'x' + v'v + x' *Associative, involution*

= v'<u>x'</u> + 0 + <u>x'</u> *Complement*

= x' *Absorption*

= H

The two logic circuits are indeed logically equivalent because they have the same outputs.

2.3 Designing logic circuits

We look at some aspects to consider when designing a logic circuit for some given a problem statement.

The procedure for designing a logic circuit is as follows:

- State the problem.
- Determine the number of input variables that are available and the number of output variables that are required.
- Identify the input and output variables by means of names or symbols.
- Compile the truth table stating the relationships between the inputs and the outputs.
- Derive the simplified Boolean expression for each output.
- Draw the logic circuit diagram.

In practice, our objectives for simplifying a logic circuit are:

- The minimum number of gates.
- The minimum number of interconnections.
- The minimum number of inputs to a gate.

These requirements have the following advantages:

- Cost saving owing to fewer physical components.
- Greater reliability, because with fewer physical components there is less chance of errors.
- A smaller number of interconnections and gates yield a faster circuit.

We look at a few exercises where the above mentioned principles are applied.

Exercise E.11

Using a truth table, determine the Boolean expression F(A, B, C) if F = 1 whenever the input contains only one 1.

Solution:

A B C	Minterms	F
0 0 0	m_0 = m_{000} = A'B'C'	0
0 0 1	m_1 = m_{001} = A'B'C	1
0 1 0	m_2 = m_{010} = A'BC'	1
0 1 1	m_3 = m_{011} = A'BC	0
1 0 0	m_4 = m_{100} = AB'C'	1
1 0 1	m_5 = m_{101} = AB'C	0
1 1 0	m_6 = m_{110} = ABC'	0
1 1 1	m_7 = m_{111} = ABC	0

Determine the output for each row in the table.

E.g. In row 2: A = 0, B = 0 and C = 1, which means that the input contains exactly one 1, so F = 1.

From the truth table: F(A, B, C) = $m_1 + m_2 + m_4$ *(m-notation)*

= $m_{001} + m_{010} + m_{100}$

= A'B'C + A'BC' + AB'C'

*(Note: e.g. $m_{00\underline{1}}$ = **A'B'**C because the **variable is complemented** when we have a **0** and not complemented when we have a 1.)*

Exercise E.12

Consider the Boolean expression F(x, y, z) = x'z + x'y + xy'z + yz.

(i) **Draw the truth table for F.**

(ii) **Derive the sum-of-minterms expression for F from the truth table.**

(iii) **Draw the Karnaugh diagram for F.**

(iv) **Derive the simplified expression directly from the Karnaugh diagram.**

(v) **Determine the simplified form of F algebraically.**

(vi) **Draw the logic circuit for the simplified F.**

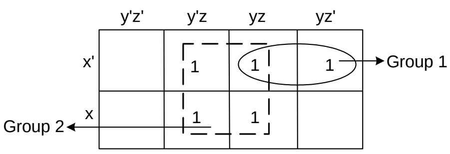
Solution:

(i)

x	y	z	x'z	x'y	xy'z	yz	F
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1
0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	1	0	1
1	1	0	0	0	0	0	0
1	1	1	0	0	0	1	1

(ii) $F = m_1 + m_2 + m_3 + m_5 + m_7 = x'y'z + x'yz' + x'yz + xy'z + xyz$

(iii)



(iv) $F = x'y + z$, *because: Group 1 gives the term x'y.*

Group 2 gives the term z.

(v) $F = x'z + x'y + xy'z + yz$

$= x'y + z(\mathbf{x'} + \mathbf{xy'} + y)$ *Distributive*

$= x'y + z(\mathbf{x'} + \mathbf{y'} + y)$ *Absorption Rule (e)*

$= x'y + z(x' + 1)$ *Complement*

$= x'y + z \cdot 1$ *Null*

$= x'y + z$ *Identity*

Exercise E.13

Four smoke sensors are placed in a room. The fire alarm must be activated as soon as three or more of the sensors detect smoke in the room. Follow the design procedure described before to design the logic circuit for the alarm.

Solution:

Number of input variables: 4; Number of output variables: 1.

Input variables: A, B, C, D representing the sensors. (If a sensor detects smoke, the input is 1.)

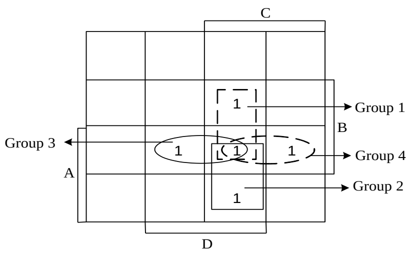
Output variable: S (S is equal to 1 if at least three sensors detect smoke.)

Truth table:

A	B	C	D	S	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	**1**	**1**	**1**	1	m_7 (**Three** sensors detect smoke.)
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	m_{11} (Three sensors detect smoke.)
1	1	0	0	0	
1	1	0	1	1	m_{13} (Three sensors detect smoke.)
1	1	1	0	1	m_{14} (Three sensors detect smoke.)
1	**1**	**1**	**1**	1	m_{15} (**Four** sensors detect smoke.)

$S = m_7 + m_{11} + m_{13} + m_{14} + m_{15}$

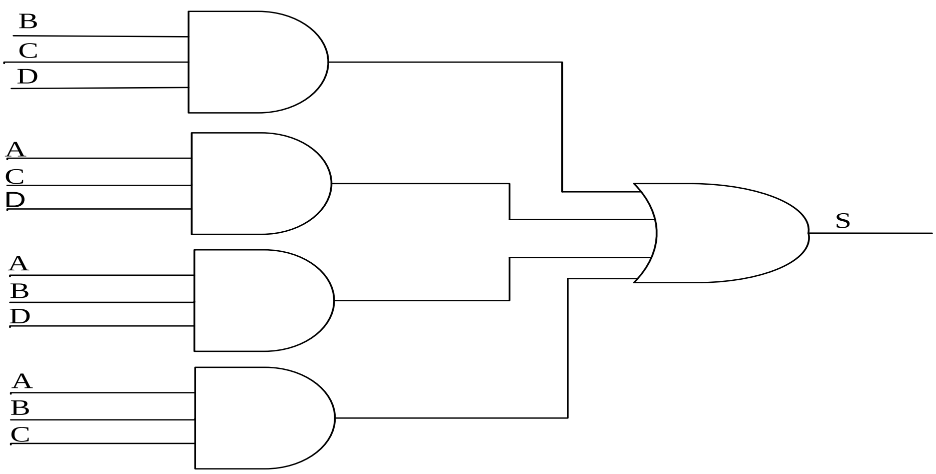
Karnaugh map:



Group 1: BCD; Group 2: ACD; Group 3: ABD; Group 4: ABC.

The simplest expression: S **=** BCD + ACD + ABD + ABC.

Logic circuit:



Exercise E.14

There are 4 different processes involved in the production of bricks in a brick factory and one person is able to handle 2 different processes. The 4 processes are:

Process 1:   mix clay

Process 2:   mould clay

Process 3:   bake bricks

Process 4:   package bricks

The 4 employers of the brick factory handle a combination of the processes in the following way:

Person A handles processes 1 and 3

Person B handles processes 2 and 4

Person C handles processes 2 and 3

Person D handles processes 1 and 4

(i) Draw a truth table to derive the sum of minterms expression for a Boolean expression (in m-notation) that will output a 1 when the brick factory is in full production, i.e. when all four different processes are executed, and a 0 otherwise.

(ii) Use a Karnaugh map to simplify the expression in (i) to its simplest form. (Do not use algebraic simplifications.)

(iii) Draw a logic circuit for the expression in (ii).

(iv) What is the minimum number of persons that must be present to keep the factory in full production?

(i) Truth table:

A	B	C	D	F	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	m_3
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	m_7
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	m_{11}
1	1	0	0	1	m_{12}
1	1	0	1	1	m_{13}
1	1	1	0	1	m_{14}
1	1	1	1	1	m_{15}

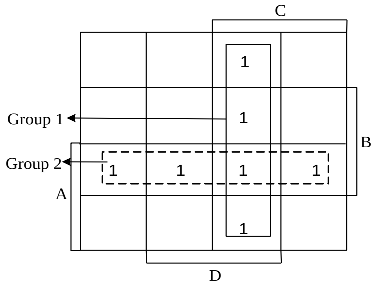
Consider row 4 in the table:

The bispatern is 0011, i.e.only persons C and D are active in the factory. C handles processes 2 and 3, and D handles processes 1 and 4. This means that the factory is in full production, i.e. all four different processes are being executed. Thus the ouput is 1.

Consider all the rows in the table in a similar way to determine the outputs.

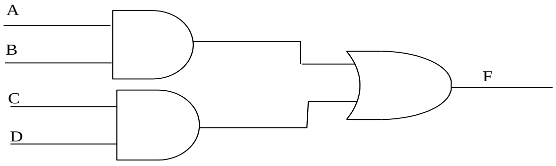
F = $m_3 + m_7 + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}$

(ii) Karnaugh map:



The simplest expression: F = CD + AB

(iii) Logic circuit:



(iv) Two people. This is determined by looking at all the rows in the truth table where the output is equal to 1. At least 2 people are required to handle all four processes.

Exercise E.15

A logic circuit has 4 input variables P, Q, R and S. X represents the decimal equivalent of the binary representation of the 4 input lines. (E.g. when P=0, Q=0, R=1 and S=1, then X=3.)

(i) By using a truth table, determine the Boolean expression F(P, Q, R, S) (in m-notation), if

F = [(X MOD 3 = 2) or (X MOD 5 = 2)].

(ii) Use a Karnaugh map to obtain the simplest form of F(P, Q, R, S). Find the simplified terms

of F directly from the Karnaugh map. (Do not use algebraic simplifications.)

> Hint: What does x MOD y = z mean? When x is divided by y, the remainder is z.
>
> For example: 13 ÷ 5 = 2 remainder 3, so 13 MOD 5 = 3.

Solution:

(i) The output will be 1 under the following condition:

F = [(X MOD 3 = 2) or (X MOD 5 = 2)].

If we look at row 3 in the following truth table, we see that

X = 2, so 2 ÷ 3 = 0 remainder 2, i.e. X MOD 3 = 2, so the output is 1.

Also: 2 ÷ 5 = 0 remainder 2, i.e. X MOD 5 = 2, so the output is 1.

The result is F = 1 + 1 = 1.

Row 6: 5 MOD 3 = 2, so the output is 1. Also 5 MOD 5 = 0, so the output is 0.

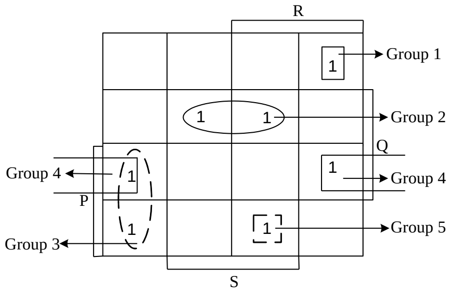
This result is F = 1 + 0 = 1.

In this way we investigate all the rows to determine the outputs.

P	**Q**	**R**	**S**	**X**	**X MOD 3=2**	**X MOD 5=2**	**F**
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	2	1	1	1
0	0	1	1	3	0	0	0
0	1	0	0	4	0	0	0
0	1	0	1	5	1	0	1
0	1	1	0	6	0	0	0
0	1	1	1	7	0	1	1
1	0	0	0	8	1	0	1
1	0	0	1	9	0	0	0
1	0	1	0	10	0	0	0
1	0	1	1	11	1	0	1
1	1	0	0	12	0	1	1
1	1	0	1	13	0	0	0
1	1	1	0	14	1	0	1
1	1	1	1	15	0	0	0

$F = m_2 + m_5 + m_7 + m_8 + m_{11} + m_{12} + m_{14}$

ii) Karnaugh map:



Group 1: P'Q'RS'; Group 2: P'QS; Group 3: PR'S'; Group 4: PQS'; Group 5: PQ'RS.

The simplest expression: F = P'Q'RS' + P'QS + PR'S' + PQS'+ PQ'RS

Exercise E.16

Draw a logic circuit that accepts a decimal number between 0 and 15 as input and provides an output of 1 if the number is 0 or 1 or a prime number.

Solution: Step 1: We need 4 variables (A, B, C, D) to represent the binary equivalent of the decimal number ($1111_2 = 15_{10}$).

Step 2: Truth table: (This step is not necessary if you are able to draw the Karnaugh diagram directly from the problem statement.)

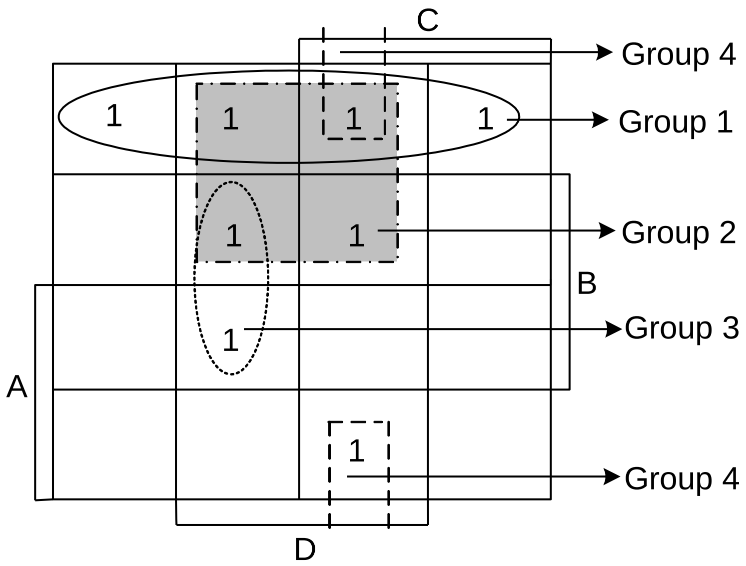
The truth table for F:

A	B	C	D	F	Interpretation	minterms
0	0	0	0	**1**	$(0000)_2 = (0)_{10}$ The number is **0**, thus the output is **1**.	m_0
0	0	0	1	**1**	$(0001)_2 = (1)_{10}$ The number is **1**, thus the output is **1**.	m_1
0	0	1	0	**1**	$(0010)_2 = (2)_{10}$ **2 is a prime number**, thus the output is **1**.	m_2
0	0	1	1	**1**	$(0011)_2 = (3)_{10}$ **3 is a prime number**, thus the output is **1**.	m_3
0	1	0	0	**0**	$(0100)_2 = (4)_{10}$ 4 is **not** a prime number, thus the output is **0**.	m_4
0	1	0	1	**1**	etc.	m_5
0	1	1	0	**0**		m_6
0	1	1	1	**1**		m_7
1	0	0	0	**0**		m_8
1	0	0	1	**0**		m_9
1	0	1	0	**0**		m_{10}
1	0	1	1	**1**		m_{11}
1	1	0	0	**0**		m_{12}
1	1	0	1	**1**		m_{13}
1	1	1	0	**0**		m_{14}
1	1	1	1	**0**		m_{15}

From the truth table:

$F = m_0 + m_1 + m_2 + m_3 + m_5 + m_7 + m_{11} + m_{13}$

Step 3: Karnaugh diagram:



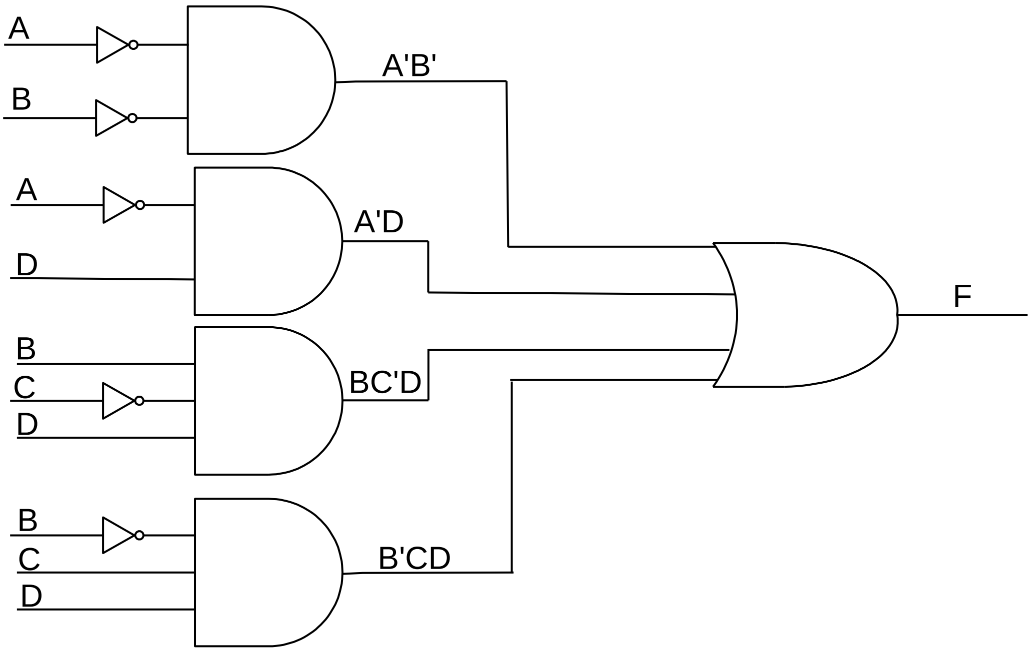
Step 4: F = A'B' + A'D + BC'D + B'CD *because: Group 1 gives the term A'B'.*

Group 2 gives the term A'D.

Group 3 gives the term BC'D.

Group 4 gives the term B'CD.

Step 5: Logic circuit:



Exercise E.17

A person will be interested in buying a house with any one of 3 sets of features x, y or z. Possible sets of features of a house in which he/she will be interested:

x) **face brick exterior and flat roof**

y) **pitched roof, carpeted floors and the house bigger than 250 square metres**

z) **plastered exterior walls, pitched roof, tiled floors and the house smaller than 250 square**

metres.

Suppose the input variables A, B, C and D take on the value 1 or 0 in the following cases:

A = 1 if the house has a face brick exterior, or A = 0 if the house has plastered exterior walls

B = 1 if the house has a flat roof, or B = 0 if the house has a pitched roof

C = 1 if the house has carpeted floors, or C = 0 if the house has tiled floors

D = 1 if the house is smaller than 250 square metres, or D = 0 if the house is bigger than 250 square metres

Construct a truth table to determine the Boolean function F(A, B, C, D) which will give a 1 whenever the person is interested in buying a house. Give F as a sum-of-minterms in m-notation.

Use a Karnaugh diagram to find the simplest form of F(A, B, C, D)

Solution:

Determine F in the truth table in the following way:

Consider the cases when a person is interested in buying the house:

First we look at the set of <u>features x</u>:

Face brick exterior (A = 1) and flat roof (B = 1). (The values of C and D do not matter.)

In the last 4 rows of the table we have A = 1 and B = 1, thus F is 1 in these 4 rows.

Look at the set of <u>features y</u>:

Pitched roof (B = 0), carpeted floors (C = 1) and the house bigger than 250 square metres

(D = 0). (The value of A does not matter.)

In the third and eleventh rows we have B = 0, C = 1, D = 0, thus F is 1 in these 2 rows.

Look at the set of <u>features z</u>:

Plastered exterior walls (A = 0), pitched roof (B = 0), tiled floors (C = 0) and the house smaller than 250 square metres (D = 1).

In the second row we have A = 0, B = 0, C = 0, D = 1, thus F is 1 in this row.

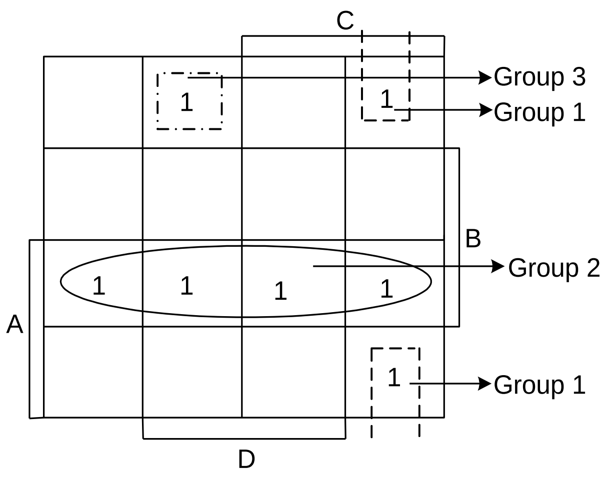
In short: F = 1 whenever (A = 1, B = 1) or (B = 0, C= 1, D= 0) or (A = 0, B = 0, C = 0, D = 1)

In all the rows of the table which are not mentioned above, F has an output of 0 because the person is not interested in buying the house (e.g. where A = 0, B = 0, C = 0 and D = 0).

A	**B**	**C**	**D**	**F**	minterms
0	0	0	0	0	
0	0	0	1	1	m_1
0	0	1	0	1	m_2
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	m_{10}
1	0	1	1	0	
1	1	0	0	1	m_{12}
1	1	0	1	1	m_{13}
1	1	1	0	1	m_{14}
1	1	1	1	1	m_{15}

From the table we derive F in m-notation: $F = m_1 + m_2 + m_{10} + m_{12} + m_{13} + m_{14} + m_{15}$.

Karnaugh diagram:



Group 1 gives B'CD'; Group 2 gives AB; Group 3 gives A'B'C'D.

The simplest form of F: F = B'CD' + AB + A'B'C'D.

PART III

Solution of the self-assessment exercise

QUESTION 1

(a) *List the four subsystems comprising a machine based on the von Neumann model*.

A machine based on the von Neumann model consists of the following 4 subsystems:

- memory in which both data and the computer program are stored,
- an Arithmetic Logic Unit (ALU) that performs calculations and logical operations,
- an input/output subsystem that accepts input from outside the computer and outputs the results, and
- a control unit that controls the operation of the other three subsystems.

(**Forouzan**, pp. 4, 5)

(b) *What does the concept 'a stored program' mean?*

The program instructions are stored in memory and are not wired into the hardware.

(**Forouzan**, p.5)

(c) *What are the two important aspects of programming that must be understood when we consider the von Neumann model?*

- Both the computer program and the data are stored in memory.
- A computer program consists of a sequence of instructions.

(**Forouzan**, pp. 5, 6)

(d) *Why does it make sense that data and program instructions have the same format?*

If data and programs have the same format, i.e. they are both represented by a binary pattern, both can be stored in the same computer memory.

(**Forouzan**, p. 5)

(e) *What is a computer program?*

A computer program consists of a sequence of instructions written in a computer programming language, telling the computer what to do with data.

(**Forouzan**, p. 2)

(f) *Describe in your own words what an algorithm is*.

An algorithm is a step-by-step description of a method that can be used to solve a particular problem.

(**Forouzan**, p. 8)

(g) *What is meant by the term 'software engineering' as defined in the context of the textbook?*

Software engineering is regarded as the process of designing and writing structured computer programs using a set of strict rules and principles.

(**Forouzan**, p. 9)

(h) *List some of the main functions of an operating system.*

- tell the computer program where to obtain input
- tell the computer program where to present the output
- memory management

There are many more but we will consider those in Chapter 7 of **Forouzan**.

(**Forouzan**, p. 9)

(i) *Compare and contrast the memory contents of early computers with the memory contents of a computer based on the von Neumann model.*

In early computer systems, the memory of a computer was used exclusively for data storage and the computer program was hardwired into the computer itself, i.e. the program formed part of the computer hardware. In a von Neumann model, both data and the computer program are stored in memory.

(**Forouzan**, p. 5)

(j) *According to the von Neumann model, can the hard disks of today be used as input or output? Explain*.

According to the von Neumann model, the hard disks of today can be used for both input and output since output data can be written to the hard disk and input data can be read from the hard disk.

(**Forouzan**, p. 5)

QUESTION 2

(a) $(10101.1)_2 = 1x2^4 + 1x2^2 + 1x2^0 + 1x2^{-1}$

$= 1x16 + 1x4 + 1x1 + 1x0.5$

$= 16 + 4 + 1 + 0.5$

$= (21.5)_{10}$

(b) $(1010011.01)_2 = 1x2^6 + 1x2^4 + 1x2^1 + 1x2^0 + 1x2^{-2}$

$= 1x64 + 1x16 + 1x2 + 1x1 + 1x0.25$

$= 64 + 16 + 2 + 1 + 0.25$

$= (83.25)_{10}$

(c) $(517)_8 = 5x8^2 + 1x8^1 + 7x8^0$

$= 5x64 + 1x8 + 7x1$

$= 320 + 8 + 7$

$= (335)_{10}$

(d) $(710.01)_8$ $= 7x8^2 + 1x8^1 + 1x8^{-2}$

$= 7x64 + 1x8 + 1x0.015625$

$= 448 + 8 + 0.015625$

$= (456.015625)_{10}$

(e) $(A9F)_{16}$ $= Ax16^2 + 9x16^1 + Fx16^0$

$= 10x256 + 9x16 + 15x1$

$= 2560 + 144 + 15$

$= (2719)_{10}$

(f) $(B08.4)_{16}$ $= Bx16^2 + 8x16^0 + 4x16^{-1}$

$= 11x256 + 8x1 + 4x0.0625$

$= 2816 + 8 + 0.25$

$= (2824.25)_{10}$

(**Forouzan**, pp. 19-22, 24-26 (ed.4), pp. 21-24, 26-27 (ed.3))

QUESTION 3

(a) $(613.625)_{10}$ to binary:

First write down the powers of 2 until a number bigger than the one we want to convert, is reached. These numbers are used as 'column headers':

2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

.........1024 512 256 128 64 32 16 8 4 2 1

Ignore the biggest number (1024), because this number is bigger than 613. Now divide 613 into 'packages' the sizes of which are the numbers of the column headers.

First we make a package of 512. Now this part of the original number (613) is represented as a binary number:

512 256 128 64 32 16 8 4 2 1

1

613 - 512 = 101, so 101 of the original number still remains.

From 101 we cannot make a package of 256 or 128. Thus we put 0s in the corresponding columns. We can, however, use 64 and thus we write a 1 in the 64-column:

512 256 128 64 32 16 8 4 2 1

1 0 0 1

101 - 64 = 37, so 37 of the original number remains.

Continuing in this way, we get 37 = 32 + 4 + 1.

So we have 512 256 128 64 32 16 8 4 2 1

$(1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1)_2$

Thus $(613)_{10} = (1001100101)_2$

For the fractional part:

$0.625 \times 2 = 0.25 + 1$

$0.25 \times 2 = 0.5 + 0$

$0.5 \times 2 = 0.0 + 1$

So $(0.625)_{10} = (0.101)_2$

Thus $(613.625)_{10} = (1001100101.101)_2$

$(1001100101.101)_2$ to octal:

$(1001100101.101)_2 = (1 \quad 001 \quad 100 \quad 101.\ 101)_2$

$= (1 \quad 1 \quad 4 \quad 5. \quad 5)_8$

$(1001100101.101)_2$ to hexadecimal:

$(1001100101.101)_2 = (10 \quad 0110 \quad 0101. \quad 101\mathbf{0})_2$ (**add a zero** to form a group of 4.)

$= (2 \quad 6 \quad 5. \quad A)_{16}$

(b) $(110.25)_{10}$ to binary:

The integer part to binary:

110 / 2 = 55 rem. 0

55 / 2 = 27 rem. 1

27 / 2 = 13 rem. 1

13 / 2 = 6 rem. 0

6 / 2 = 3 rem. 0

3 / 2 = 1 rem. 1

1 / 2 = 0 rem. 1

Thus $(120)_{10} = (1100110)_2$

The fractional part to binary:

0.25 x 2 = 0.5 + 0

0.5 x 2 = 0.0 + 1

Thus $(0.25)_{10} = (0.01)_2$

Thus $(110.25)_{10} = (1100110.01)_2$

$(110.25)_{10}$ to octal:

$(1100110.01)_2 = (001100110.\ 01\mathbf{0})_2$ (Add a **zero** to form a group of 3.)

$(1 \quad 4 \quad 6\ .\ 2\ )_8$

$(110.25)_{10}$ to hexadecimal:

$(1100110.01)_2 = (0110\ 0110.\ 01\mathbf{00})_2$ (Add **zero** to form a group of 4.)

$= (\ 6 \quad 6\ .\ 4\ )_{16}$

(**Forouzan**, pp. 26-28 (ed.4), pp. 28-30 (ed.3))

QUESTION 4

The number 845.3 is not an octal number because we only use the digits 0 to 7 in the octal system. The digit 8 occurs in the given number.

(**Forouzan**, p. 23 (ed.4), p. 25 (ed.3))

QUESTION 5

(Subtraction in binary does not form part of the syllabus, but you can try to do it by first looking at the principles by which subtraction in decimal is executed. For example, borrow a 2 instead of a 10 as is the case in decimal.)

(a)		(b)	
	10111		110100
	1111 +		10011-
	100110		100001

(**Forouzan**, p. 83 (ed.4), p. 85 (ed.3))

QUESTION 6

The most important disadvantage of using sign-and-magnitude representation is that there are two ways of representing zero, namely -0 and +0. When testing to see whether a result is equal to zero or not, one has to test for -0 as well as for +0. Another disadvantage is the fact that the sign is not an integral part of the number and has to be treated separately in arithmetic operations.

(**Forouzan**, p. 44 (ed.4), p. 46 (ed.3))

QUESTION 7

(a) $(78.43)_{10}$ to binary.

The integer part:

64	**32**	**16**	**8**	**4**	**2**	**1**
1	0	0	1	1	1	0

So $(78)_{10} = (1001110)_2$

The fractional part:

0.43 x 2 = 0.86 + 0

0.86 x 2 = 0.72 + 1

0.72 x 2 = 0.44 + 1

So $(0.43)_{10} = (0.011)_2$ to 3 binary places

From the above it follows: $(78.43)_{10}$ = $(1001110.011)_2$

= $(1.001110011)_2 \times (2^6)_{10}$

(b) $(1.39 \times 10^2)_{10} = (139)_{10}$ to binary:

139 / 2	=	69	remainder 1
69 / 2	=	34	remainder 1
34 / 2	=	17	remainder 0
17 / 2	=	8	remainder 1
8 / 2	=	4	remainder 0
4 / 2	=	2	remainder 0
2 / 2	=	1	remainder 0
1 / 2	=	0	remainder 1

Thus $(139)_{10}$ = $(10001011)_2$

= $(1.0001011)_2 \times (2^7)_{10}$

(**Forouzan**, pp. 52-54 (ed.4), pp. 54-56 (ed.3))

QUESTION 8

A logical right shift operation is applied to the bit pattern 11001111.

The operation shifts each bit one position to the right, so the result is:

01100111

(**Forouzan**, p. 79 (ed.4), p. 81 (ed.3))

QUESTION 9

Using an 8-bit allocation, use two's complement arithmetic to determine -15 + 12.

To obtain the two's complement number: ***Copy the bits from the right until a 1 is copied, then flip the rest of the bits****.*

15 in binary: 00001111. Apply two's complement because -15 is a negative number: 11110001

	0	0	0	0	1	1	0	0	binary number for 12
+	1	1	1	1	0	0	0	1	two's complement applied on 15
	1	1	1	1	1	1	0	1	**negative** number
	0	0	0	0	0	0	1	1	two's complement applied

Convert 00000011 to decimal: 3. Sign is added: -3

(**Forouzan**, pp. 47-49 (ed.4), pp. 49-51 (ed.3))

Solutions of the Self-assessment: Section B

NOTATION IN THIS SOLUTION: Two different fonts in this model solution are used. The steps for the model solutions are in normal font, while the extra bits of explanation are in Italics *like this*.

QUESTION 1 ***[4]***

(a) **Use the XOR operator on the bit patterns 100110101 and 101010011. (Determine 100110101 XOR 101010011.)**

(b) **Determine 1101101 + 1000110 in binary.**

(c) **A 6-bit digital counter can be made up of ______ T flip-flops. At the start the counter represents _________.**

Solution:

(a)

```
       100110101 Input 1
XOR    101010011 Input 2
       001100110 Output
```

Forouzan, (pp. 79, 564, 565 (ed.4), pp. 81, 544, 545 (ed.3))

(b) Addition in binary (the base is 2, i.e. two):

1101101

+ <u>1000110</u>

10110011

Explanation:

Place values:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Carry:		*1*			*1*	*1*		
Augent:	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*1*
Addent:	*0*	*1*	*0*	*0*	*0*	*1*	*1*	*0*
Sum:	*1*	*0*	*1*	*1*	*0*	*0*	*1*	*1*

Forouzan, p. 83 (ed.4), p. 85 (ed.3); Table 4.1; Tutorial Letter 102 pp. 40-42

(c) **A 6-bit digital counter can be made up of <u>6</u> T flip-flops. At the start the counter represents** <u>000000</u>.

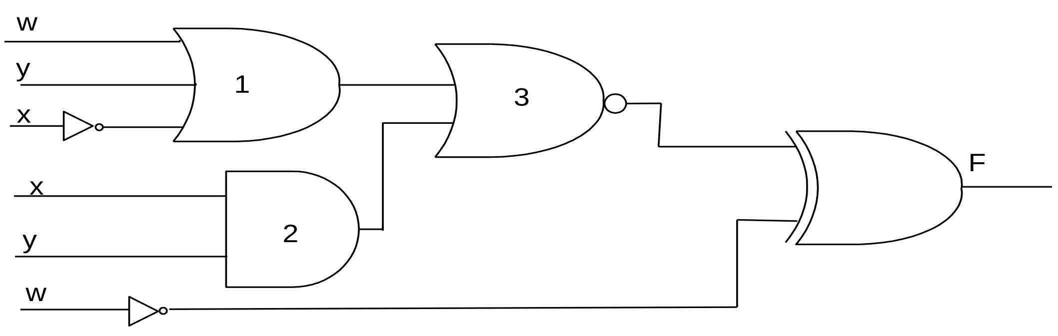
Forouzan, p. 580 (ed.4), p. 560 (ed.3)

QUESTION 2 ***[4]***

Draw the logic circuit for the following Boolean expression (do not simplify the expression):

F(x, y, w) = [(x' + y + w) + xy]' ⊕ w'

Solution:



Output Gate 1: x' + y + w; Output Gate 2: xy; Output Gate 3: [(x' + y + w) + xy]'

Final output at F using gate XOR and w' (starting with input w at the bottom): [(x' + y + w) + xy]' $\oplus$ w'

Forouzan, pp. 564-567, 574-580 (ed.4), pp. 544-547, 554 – 560 (ed.3); Tutorial Letter 102, pp. 73-76.

QUESTION 3 ***[8]***

Use only Boolean algebra to simplify the Boolean expression F. (First determine F_1 and F_2, then simplify $F_1 + F_2$, showing all the steps. You need not provide the names of the Boolean rules that you apply.

$F_1 = x'(wy')' + x'wy'$
$F_2 = (xw + w')'$
$F(w, x, y) = F_1 + F_2$.

Solution:

F_1 = x'(wy')' + x'wy'

= x'[(<u>wy'</u>)' + <u>wy'</u>] *Distributive: a(b + c) = ab + ac*

= x'(1) *Complement: a' + a = 1 (Let a = <u>wy'</u>)*

= x' *Identity: a·1 = a*

F_2 = (x·w + w')'

= (x·w)' · w'' *de Morgan: (a + b)' = a'·b'*

= (x' + w') · w *de Morgan: (a · b) ' = a' + b'; Involution: a'' = a*

= x'w + w'w *Distributive*

= x'w + 0 *Complement: aa' = 0*

= x'w *Identity: a + 0 = a*

F = x' + x'w

= x' *Absorption: a + ab = a (Tut. Lettter 102, p. 38)*

Make sure that you understand how to apply de Morgan's theorem. Look at the explanation on pp. 48-50 in Tutorial Letter 102.

Note: The Boolean-algebra rules, given on pp. 47 in Tutorial Letter 102, can be expanded. We look at a few examples:

Distributive: $x + (yz) = (x + y)(x + z)$

Rule applied: $vw + mnz = (vw + mn)(vw + z)$

Complement: $x + x' = 1$

Rule applied: $v'wy + (v'wy)' = 1$

Null $x + 1 = 1$

Rule applied: $(vbr + gde + nm) + 1 = 1$

Absorption: $\underline{x} + (\underline{x}y) = \underline{x}$

Rule applied: $\underline{x} + (\underline{x}yw + \underline{x}vtg + \underline{x}de + \underline{x}m) = \underline{x}$ *(x stands alone and is present in each of the other terms.)*

Forouzan, pp. 568 – 571 (ed.4), pp. 548 – 551 (ed.3); Tutorial Letter 102, pp. 47-53

QUESTION 4 ***[5]***

Use a Karnaugh map to find the simplest form of
$H(A, B, C, D)) = m_0 + m_1 + m_2 + m_3 + m_5 + m_6 + m_8 + m_9 + m_{13}$.

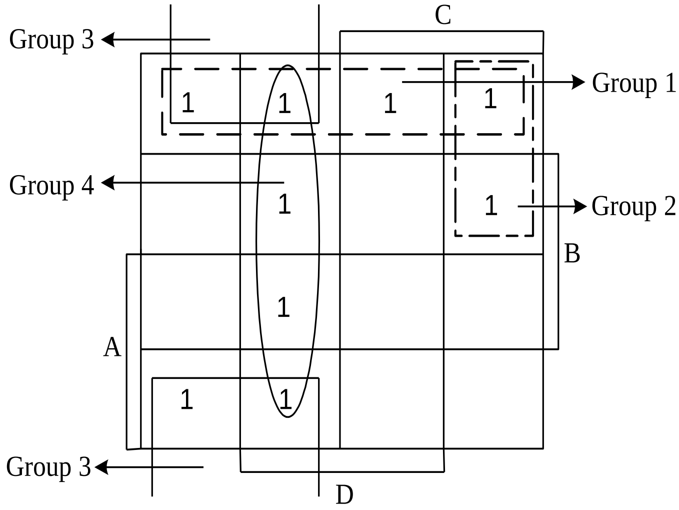
Derive the terms of H directly from the Karnaugh map without making use of algebraic manipulations or truth tables. Clearly show the groupings.
Use exactly the same order for the variables as given in the following diagram:

Solution:

First draw the Karnaugh diagram for H. Keep to the rules that are given on page 56 in Tutorial Letter 102 when you indicate the groups clearly in the diagram.

The 1s in the following Karnaugh diagram may not be placed in any other way.

Karnaugh diagram for H:



Note: We can assign the variables A, B, C and D in other ways to the rows and columns of the diagram, but then we must be careful to place the 1s in the relevant positions.

*The expression must be derived **directly** from the Karnaugh diagram. The minterms must not be grouped together and then be simplified algebraically.*

Determine the terms of the expression H:

Group 1 gives A'B'.

Group 2 gives A'CD'.

Group 3 gives B'C'.

Group 4 gives C'D.

We derive the simplified expression directly from the diagram: H = A'B' + A'CD' + B'C' + C'D.

Forouzan, pp. 571 – 573 (ed.4), pp. 551 – 553 (ed.3); Tutorial Letter 102, pp. 58-72

QUESTION 5 ***[9]***

Four types of package (A, B, C and D) with chemicals are supplied to research laboratories. Each package contains unique types of chemicals.
Package A contains 3 different types of chemicals,
Package B contains 6 different types of chemicals,
Package C contains 5 different types of chemicals and
Package D contains 2 different types of chemicals.

Suppose the input variables A, B, C and D in a truth table take on the value 1 whenever a laboratory receives a package with chemicals. For example, if A = 0, B = 1, C = 0 and D = 1, it means that a laboratory receives packages B and D.

Construct a truth table (use the same order for the variables as in the table given here) to determine the Boolean function F(A, B, C, D) that gives a 1 whenever a laboratory receives more than 11 different types of chemicals.

Solution:

NB: The Numbers of different types of chemicals are shown in brackets (2). This is optional.

A (3)	B (6)	C (5)	D (2)	F	minterms
0	0	0	0	0	m_0
0	0	0	1	0	m_1
0	0	1	0	0	m_2
0	0	1	1	0	m_3
0	1	0	0	0	m_4
0	1	0	1	0	m_5
0	1	1	0	0	m_6
0	1	1	1	1	m_7 Laboratory receives packages B, C and D, i.e. 13 types of chemicals, i.e. more than 11 types of chemicals.
1	0	0	0	0	m_8
1	0	0	1	0	m_9
1	0	1	0	0	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	1	m_{14} Laboratory receives packages A, B and C, i.e. 14 types of chemicals, i.e. more than 11 types of chemicals.
1	1	1	1	1	m_{15} Laboratory receives packages A, B, C and D, i.e. 16 types of chemicals, i.e. more than 11 types of chemicals.

From the table we obtain F in m-notation: $F = m_7 + m_{14} + m_{15}$.

Tutorial Letter 102, pp. 87 – 92

Multiple choice questions (solution)

QUESTION 6 ***Alternative D***

The <u>XOR-gate</u> has an output of 1 only if it has two inputs that are not equal (i.e. inputs 0 and 1). If the inputs are both 0 or both 1, the output is 0.

Forouzan, p. 564 (ed.4), p. 544 (ed.3)

QUESTION 7 ***Alternative C***

<u>Two</u> adjacent minterms must be grouped together in a four variable *Karnaugh* map to derive a simplified term consisting of three variables.

Tutorial Letter 102, p. 62

QUESTION 8 ***Alternative B***

Logic circuits are divided into two categories: combinational circuits and sequential circuits. A flip-flop falls in the category of <u>sequential circuits</u>.

Forouzan, p. 576 (ed.4), p. 556 (ed.3)

QUESTION 9 ***Alternative D***

A *n*-bit digital counter counts from 0 to $2^n – 1$. Thus a 3-bit digital counter counts from 0 to $2^3 – 1 = \underline{7}$.

Forouzan, p. 580 (ed.4), p. 560 (ed.3)

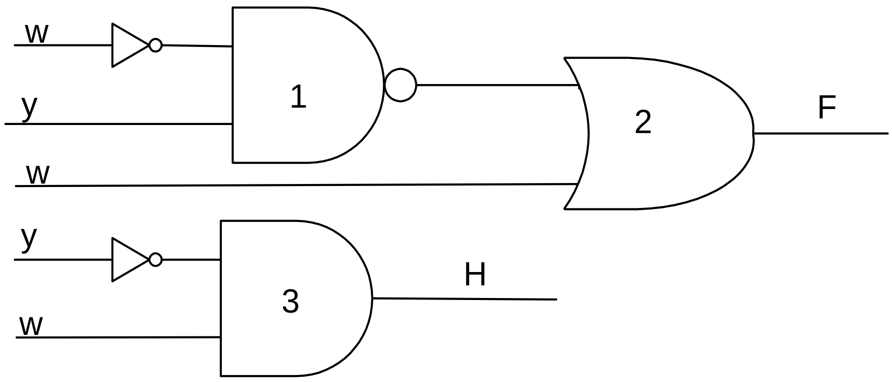
QUESTION 10 ***Alternative B***

A multiplexer is a combinational circuit that has *n* inputs and only 1 output.

Forouzan, p. 575 (ed.4), p. 555 (ed.3)

QUESTION 11 ***Alternative D***

Consider the following logic circuits:



The outputs of the given logic circuits are F = (w'y) '+ w and H = y'w. One of the four gates must be changed in order for the circuits to become equivalent.

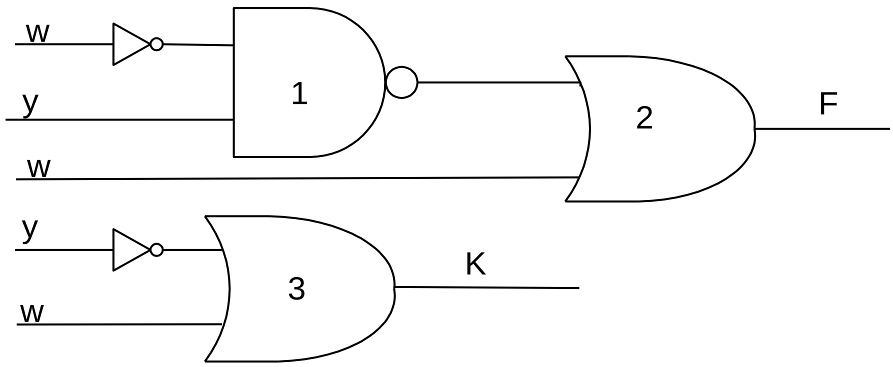
Solution:

Simplify: F = (w'y) '+ w = w'' + y' + w = w + w + y' = w + y' *de Morgan; Involution; Idempotence*

Simplify: H = y' · w

The outputs F and H of the two circuits are not equal, so they are not logically equivalent.

In the given question, Gate 3 is an AND-gate and its output is y' · w. If Gate 3 changes to an 'OR'-gate, the output for Gate 3 becomes w + y'. As a result, let K = w + y':



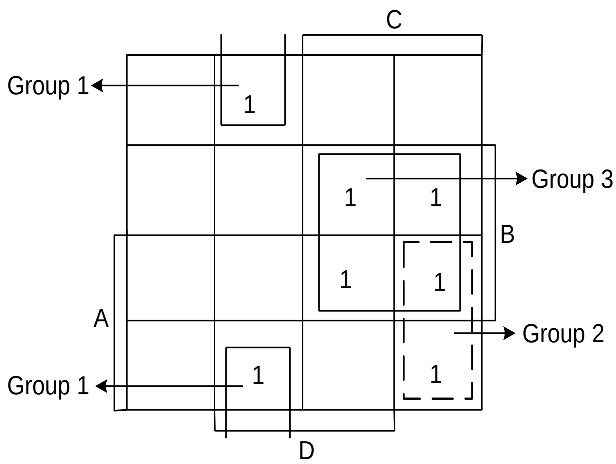
Because F = K, the circuits are logically equivalent.

Tutorial Letter 102, pp. 50 - 53, 73

QUESTION 12 ***Alternative B***

We use a *Karnaugh* map to find the simplest form of the following sum-of-minterm expression:

$$F(A, B, C, D) = m_1 + m_6 + m_7 + m_9 + m_{10} + m_{14} + m_{15}$$



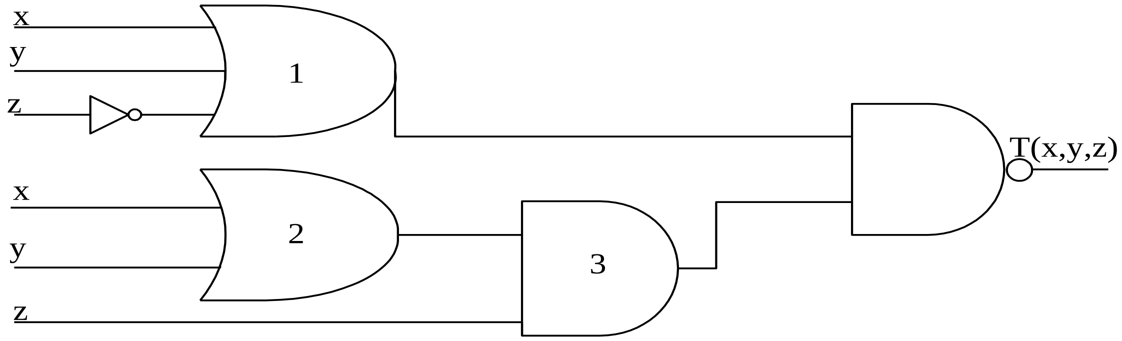
Group 1 gives B'C'D; Group 2 gives ACD'; Group 3 gives BC.

So F = B'C'D + ACD' + BC.

Tutorial Letter 102, pp. 58-72

QUESTION 13 ***Alternative D***

Consider the following logic circuit:



Output Gate 1: x + y + z'*; Output Gate 2:* x + y; *Output Gate 3:* (x + y) · z

The final output is T = [(x + y + z') · ((x + y) · z)]'.

Tutorial Letter 102, pp. 73 onwards

QUESTION 14 ***Alternative A***

Consider the expression F = (xy')' · **[**x'z**]** + (x'' + y').

We determine the values of (xy')', **[**x'z**]**, (x'' + y') and F if x = 1, y = 0 and z = 1:

F = (xy')' · **[**x'z**]** + (x'' + y')

= (1·0')' · **[**1'·1**]** + (1'' + 0')

= (1·1)' · **[**0·1**]** + (1 + 1)

= (1)' · **[**0**]** + (1)

= 0 · **[**0**]** + (1)

= 1

i.e. (xy')' = 0; **[**x'z**]** = **[**0**]**; (x'' + y') = (1) and F = 1.

Forouzan, pp. 564 – 565 (ed.4), pp. 543 – 545 (ed.3); Tutorial Letter 102, pp. 42-44

QUESTION* 15** ***Alternative D

We use only Boolean algebra to simplify the following Boolean expression: F(v, w, x) = vxw' + (vxw')'

F = vxw' + (vxw')' = 1 *Complement*

(Let y = vxw' then vxw' + (vxw')' = y + y' = 1.)

Forouzan, pp. 568 – 571 (ed.4), pp. 548 – 551 (ed.3); Letter 102, pp. 47-54

~~~~~~~~~~~~~~~~~~~~~~~~~~End of self- assessment solution~~~~~~~~~~~~~~~~~

Transformation of a Boolean function to a sum of products (minterms)

Please note that only the sum of products method is included in the syllabus. (**Forouzan**, pp. 572, 573 (ed.4), pp. 552, 553 (ed.3); examples E.2 and E.3; Tutorial Letter 102, Section 1.6, p. 53)

The product of sums method explained in **Forouzan**, pp 573, 574 (ed.4), pp. 553, 554 (ed.3) is excluded. This means that the product of sums functions in Figure E.7 and examples E.4 and E.5 are also excluded.

Appendix I

Engels / Afrikaans woordelys

WOORDELYS: ENGELS - AFRIKAANS

AC adapter	wisselstroom aansluitstuk
access control	toegangsbeheer
access privileges	toegangsvoorregte
access time	toegangstyd
address	adres
address bus	adresstam
Analog	analoog
analysis phase	ontledingsfase
anti-spam program	anti-spamprogram / anti-sproeiposprogram
antivirus	antivirus
applet	miniprogram
application generator or program generator	programgenerator
application service provider (ASP)	toepassingdiensvoorsiener
application software	toepassingsagteware
arithmetic/logic unit (ALU)	rekenkunde en logika-eenheid (RLE)
arrays	skikkings
arrow keys	pyltjies
artificial intelligence (AI)	kunsmatige intelligensie
assembler	saamsteller
assembly language	saamsteltaal
asymmetric digital subscriber line (ADSL)	asimmetriese digitale intekenaarlyn
asynchronous	asinchrone
audio	oudio
Authentication	waarmerk
back up	rugsteun
backbone	ruggraat
backup procedures	rugsteunprosedures
backup utility	rugsteunnutsprogram
bandwidth	bandwydte
batch processing	bondelverwerking
binary	binêr
biometric identifier	biometriese identifiseerder
bit or binary digit	bis of binêre syfer
bitmap	biskaart
black box	swartkis
booting	selflaai
browser routine	blaaier-roetine
bubble	borrel
bucket hashing	houerhutsing

buffer	buffer
bugs	foute
bus	bus of stamlyn
bus width	busgrootte
byte	greep
capacity	kapasiteit
cathode ray tube (CRT)	katode straalbuis
CD-R (compact disc-recordable)	kompakte skyf (skryfbaar)
CD-ROM (read only memory)	kompakte skyf (lees alleen geheue)
CD-ROM drive or CD-ROM player	kompakte skyf aandrywer of speler
CD-RW (compact disc-rewritable)	compakte skyf (herskryfbaar)
central processing unit (CPU) or processor	sentrale verwerkingseenheid (SVE)
check digit	kontrolesyfer
chip	vlokkie
CISC-type computers (Complex Instruction Set Computers)	CISC-tipe rekenaars
ciphertext	syferteks
clock cycle	kloksiklus
clock speed or clock rate	klokspoed
coaxial cable or Coax	koaksiale kabel
commands	bevele
common gateway interface (CGI)	gemeenskaplike deurgangspoortkoppelvlak
communications software	kommunikasiesagteware
compact disc (CD)	kompakte skyf
complementary metal-oxide semiconductor or CMOS	komplimentêre metaaloksied-halfgeleier KMOH
computer-aided design (CAD) software	rekenaar-gesteunde ontwerp sagteware
computer-aided software engineering (CASE)	rekenaargesteunde sagteware-ingenieurswerk
connector	aansluiter
construct	konstruk
control structure or construct	konstruk
control unit	beheereenheid
controller	beheerder
cookie	koekie
coupling	koppeling
coprocessor	medeverwerker
cracker or hacker	kraker of inbreker
CRT monitor or monitor	katodestraalbuis (KSB)
custom software	doelgemaakte sagteware
data bus	datastamlyn
data conversion	data-oorskakeling
data-link layer	dataverbindingslae
data processing	dataverwerking
data transfer rate	data-oordragtempo
data type	datatipe
data warehouse	datastoor
database management system (DBMS)	databasisbestuurstelsel
database server	databasisbediener
database software or a database management system (DBMS)	databasissagteware of 'n databasis bestuurstelsel (DBBS)
deadlock	dooiepunt
debug utility or debugger	ontfouter
decision support system (DSS)	besluitnemingsondersteuningstelsel
decision table	beslissingstabel
decision tree	beslissingsboom

declarative	deklaratiewe
decrypt	ontsyfer
default value	verstekwaarde
deliverable	afgelewerde item
design phase	ontwerpfase
design tool	ontwerphulpmiddel
desk checking	handontfouting
desktop	skerm oppervlakte
desktop computer	tafelrekenaar
desktop publishing (DTP) software	tafelpublikasie sagteware
device driver or driver	toesteldrywer
digital	digitaal
digital certificate or public-key certificate	digitale sertifikaat of openbare-sleutel sertifikaat
digital divide	digitale kloof
digital signature or digital ID	digitale merkteken of digitale ID
disk	skyf
disk controller	skyfkontroleerder
disk defragmenter	skyfdefragmenteerder
distributed	verspreide
DNS server	DNS-bediener
documentation	dokumentasie
domain name	domeinnaam
domain name system (DNS)	domeinnaamstelsel
do-until control structure	doen-tot beheerstruktuur
do-while control structure	doen-terwyl beheerstruktuur
download	aflaai
drive bays	aandrywer gleuwe (skyf gleuf)
DVD-ROM (digital video disc-ROM)	digitale videoskyf-LAG
dynamic RAM or DRAM	dinamiese LAG
e-mail or electronic mail	e-pos of elektroniese pos
encryption key	enkripsie sleutel
entity	entiteit
entity-relationship diagram (ERD)	entiteitsverhoudingsdiagram
execute	verwerk of uitvoer
expansion slot	uitbreidingsvak
expert system	ekspertstelsel
field	veld
file	lêer
floating-point coprocessor	wisselpuntverwerker
fragmented	gefragmenteer
frame	raam
FTP (File Transfer Protokol) server	lêer-oordrag-protokol bediener
Gantt chart	Gnatt kaart
graphic or graphical image	grafikabeeld
graphical user interface or GUI	grafiese gebruikerskoppelvlak
groupware	groepware
hard disk	harde skyf
hardware	hardeware
host computer	gasheerrekenaar
hub	nodus; spil
hyperlink or link	hiperkoppeling of koppeling
hypertext markup language (HTML)	hiperteksmarkeertaal
hypertext transfer protocol	hiperteks protokol
icon	ikoon

image editing	beeldredigering
image processing or imaging	beeldverwerking
implementation phase	implementeringsfase
indexed files	geïndekseerde leers

information	inligting
information hiding	inligtingsverskansing
information processing cycle	inligtingsverwerkingsiklus
information system	inligtingstelsel
inheritance	oorerwing
Input	toevoer
input device	toevoertoestel
instance	instansie
instant messenger	onmiddellike boodskapper
instruction time or i-time	instruksietyd of i-tyd
integrated circuit (IC)	geïntegreerde stroombaan
Internet service providers (ISPs) or online service providers (OSPs) or content portals or online malls	Internet diensvoorsieners of gekoppelde diensvoorsiener of inhoudelike poorte of gekoppelde koopsentrums
join operation	verbindingsbewerking
IP address or Internet protocol address	IP adres of Internetprotocol adres
kernel	kern
key field or primary key	sleutelveld of primêre sleutel
keyboard	sleutelbord
keyword	sleutelwoord
laptop computer or notebook computer	skootrekenaar
linked list	geskakelde lys
log on	aanlog (aanteken)
logic	logika
logical design	logika ontwerp
machine cycle or instruction cycle	masjiensiklus of instruksietyd
machine language	masjientaal
macro	makro
mainframe	hoofraam
management information system or MIS	bestuursinligtingstelsel
memory	geheue
memory cache	kasgeheue
message board or discussion board	boodskapbord of besprekingsbord
metropolitan area network (MAN)	metropolitaanse area-netwerk
microbrowser or minibrowser	mikroblaaier of miniblaaier
modularity	modulariteit
monitoring	monitering
motherboard or system board	moederbord of stelselbord
mouse pad	muiskussing
multimedia	multimedia
multimedia authoring software	multimedia-outeursagteware
multiplexer or MUX	multiplekser
multiplexing	multipleksering
multiprocessing	multiverwerking
multitasking	multitaakverwerking
nonrepudiation	nie-repudiëring
object	objek
object code or object program	objekkode
object query language (OQL)	objeknavraagtaal
object-oriented	objekgeoriënteerd
object-oriented programming (OOP) language	objek-georiënteerde programmeringstaal
object-relational data model	objekrelasionele datamodel
OCR devices	optiese karakterherkennings toestelle
operability	werkbaarheid
operating system	bedryfstelsel

output	afvoer
output device	afvoertoestel
paging	paginering
parallel processing	parallelle verwerking
parity bit	pariteitsbis
partitions	partisies (afbakenings)
physical design	fisiese ontwerp
pie-chart	sirkelgrafiek
pipelining	pyplynwerking
pixel	beeldelement
plaintext privacy	gewone teks-privaatheid
planning phase	beplanningsfase
port	poort
portability	oordraagbaarheid
private key encryption or symmetric key encryption	private sleutelenkripsie of simmetriese sleutelenkripsie
procedural	prosedurele
programmable read-only memory (PROM)	programmeerbare lees-alleen geheue
public key encryption or Asymmetric key encryption	openbare sleutelenkripsie of Assimetriese sleutelenkripsie
public switched telephone network (PSTN)	publiekgeskakelde telefoonnetwerk
public-domain software	publieke domein sagteware
quality review	kwaliteitsoorsig
queue	Tou
raster graphics	rastergrafika
RISC-type computers (Reduced Instruction Set Computers)	RISC-tipe rekenaars
registers	registers
repeater	herhaler
reset	terugstel
resolution	resolusie
resources	hulpbronne
ROM - read-only memory	lees-alleen geheue (LAG)
router	roeteerder
saved	gestoor
scope	bestek
screen saver	skermskut of skermbespaarder
search engine	soekenjin
secure server	veilige bediener
secure site	veilige tuiste
selection sort	seleksiesortering
server	bediener
signature	kenteken
signature verification system	kenteken verifikasiestelsel
single user or single tasking	enkeltaakwerking
software	sagteware
source document	brondokument
source program	bronprogram
speech recognition or voice recognition	spraakherkenning
stand-alone	alleenstaande
starvation	uithongering
state diagram	toestandsdiagram
static RAM or SRAM	statiese ETG
storage	stoor
storage device	stoortoestel

storage device	stoortoestel
streaming audio or Streaming sound	stromende oudio of stromende geluid
structure chart	struktuurkaart
structured design	gestruktureerde ontwerp
Structured Query Language (SQL)	gestruktureerde navraagtaal
syntax	sintaksis
tape drive	bandaandrywer
terminal	terminaal
test data	toetsdata
usability	bruikbaarheid
video capture card	video-vaslê kaart
video card or video adapter or graphics card	videokaart of video aansluitstuk of grafika kaart
video digitizer	video-versyferaar
video editing software	webblaaier of blaaier
video editing software	video-redigeringsagteware
video input or video capture	video-toevoer
video conference	videokonferensie
virtual memory (VM)	virtuele geheue
walking pointer	navolgingswyser
Web appliances or Internet appliances	Web-toestelle of Internet-toestelle
Web bar codes	Web strepieskode
Web browser or browser	Webblaaier of blaaier
Web filtering software or Internet filtering software	Web filtreringsagteware of Internet filtreringsagteware
Web hosting services	Webgasheer dienste
Web page authoring software	Webbladsy outeursagteware
Web page authors	webbladsy-outeurs
Web publishing	Webpublisering
Web server	Webbediener
Webmaster	Webmeester
Webware or a Web application	Webware of 'n Webtoepassing
word processing software	woordverwerkingsagteware
word size	woordlengte
World Wide Web (WWW) or Web	Wêreldwye Web (WWW) of Web
World Wide Web Consortium (W3C)	World Wide Web Consortium (W3C)
World Wide Web or WWW or Web	wêreldwye web
zipped file	kompaklêer