CSC 111
INTRODUCTION TO COMPUTER SCIENCE I

Course Development Team

Subject Matter Expert

Abimbola G. Akintola (Ph.D.)
Department of Computer Science
University of Ilorin, Nigeria

Content Editor

Bankole Ogechi Ijeoma
Centre for Open & Distance Learning,
University of Ilorin, Nigeria

Instructional Designers

Olawale Sunday Koledafe
Centre for Open & Distance Learning,
University of Ilorin, Nigeria

Samule Adekanye
Centre for Open & Distance Learning,
University of Ilorin, Nigeria

Graphics Designer

Jamiu Babatunde Bello
Department of Educational Technology,
University of Ilorin, Nigeria

Published by the Centre for Open and Distance Learning,
University of Ilorin, Nigeria

E-mail: codl@unilorin.edu.ng
Website: https://codl.unilorin.edu.ng

From the Vice Chancellor

Courseware development for instructional use by the Centre for Open and Distance Learning (CODL) has been achieved through the dedication of authors and the team involved in quality assurance based on the core values of the University of Ilorin. The availability, relevance and use of the courseware cannot be timelier than now that the whole world has to bring online education to the front burner. A necessary equipping for addressing some of the weaknesses of regular classroom teaching and learning has thus been achieved in this effort.

This basic course material is available in different electronic modes to ease access and use for the students. They are available on the University's website for download to students and others who have interest in learning from the contents. This is UNILORIN CODL's way of extending knowledge and promoting skills acquisition as open source to those who are interested. As expected, graduates of the University of Ilorin are equipped with requisite skills and competencies for excellence in life. That same expectation applies to all users of these learning materials.

Needless to say, that availability and delivery of the courseware to achieve expected CODL goals are of essence. Ultimate attention is paid to quality and excellence in these complementary processes of teaching and learning. Students are confident that they have the best available to them in every sense.

It is hoped that students will make the best use of these valuable course materials.

Professor S. A. Abdulkareem
Vice Chancellor

Forward

Courseware remains the nerve centre of Open and Distance Learning. Whereas some institutions and tutors depend entirely on Open Educational Resources (OER), CODL at the University of Ilorin considers it necessary to develop its own materials. Rich as OERs are and widely as they are deployed for supporting online education, adding to them in content and quality by individuals and institutions guarantees progress. Doing it in-house as we have done at the University of Ilorin has brought the best out of the Course Development Team across Faculties in the University. Credit must be given to the team for prompt completion and delivery of assigned tasks in spite of their very busy schedules.

The development of the courseware is similar in many ways to the experience of a pregnant woman eagerly looking forward to the D-day when she will put to bed. It is customary that families waiting for the arrival of a new baby usually do so with high hopes. This is the apt description of the eagerness of the University of Ilorin in seeing that the centre for open and distance learning [CODL] takes off.

The Vice-Chancellor, Prof. Sulyman Age Abdulkareem, deserves every accolade for committing huge financial and material resources to the centre. This commitment, no doubt, boosted the efforts of the team. Careful attention to quality standards, ODL compliance and UNILORIN CODL House Style brought the best out from the course development team. Responses to quality assurance with respect to writing, subject matter content, language and instructional design by authors, reviewers, editors and designers, though painstaking, have yielded the course materials now made available primarily to CODL students as open resources.

Aiming at a parity of standards and esteem with regular university programmes is usually an expectation from students on open and distance education programmes. The reason being that stakeholders hold the view that graduates of face-to-face teaching and learning are superior to those exposed to online education. CODL has the dual-mode mandate. This implies a combination of face-to-face with open and distance education. It is in the light of this that our centre has developed its courseware to combine the strength of both modes to bring out the best from the students. CODL students, other categories of students of the University of Ilorin and similar institutions will find the courseware to be their most dependable companion for the acquisition of knowledge, skills and competences in their respective courses and programmes.

Activities, assessments, assignments, exercises, reports, discussions and projects amongst others at various points in the courseware are targeted at achieving the objectives of teaching and learning. The courseware is interactive and directly points the attention of students and users to key issues helpful to their particular learning. Students' understanding has been viewed as a necessary ingredient at every point. Each course has also been broken into modules and their component units in sequential order.

At this juncture, I must commend past directors of this great centre for their painstaking efforts at ensuring that it sees the light of the day. Prof. M. O. Yusuf, Prof. A. A. Fajonyomi and Prof. H. O. Owolabi shall always be remembered for doing their best during their respective tenures. May God continually be pleased with them, Aameen.

Bashiru, A. Omipidan
Director, CODL

Course Guide

Our world today revolve around use of data. We all needs data everyday to help us make different life decision. Therefore, there is a need to process data at a faster rate. It may intrest you to note that the manual method of processing data is slow and comes with diverse challenges such as storage, processing time and duplication.

With the advent of computer systems, challenges of traditional data processing method has been optimally addressed. Therefore, this course will introduce you to the different types Computer files together with the way they are Organized, structured, manipulated and stored in the computer memory.

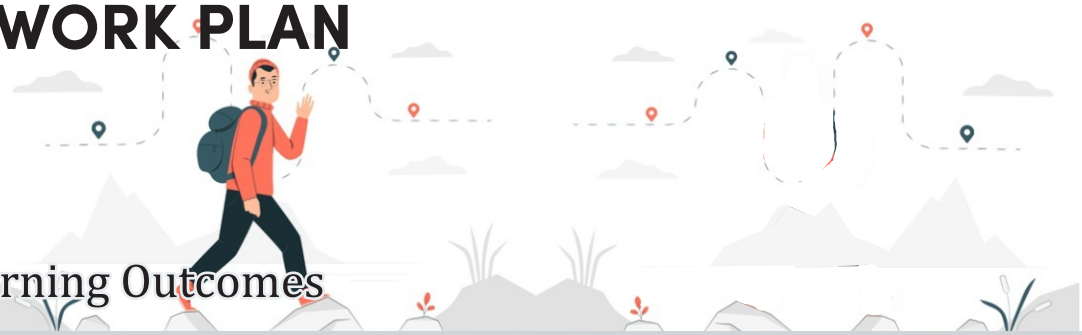
Furthermore, this course will introduce you to the Computer filing system concept since no data processing in the computer is possible without the use of files. Computer file processing and its manipulation will be discussed extensively

For your information, this course is made up of four modules and fourteen units. Each of the units explain in details the components of the computer file system. Module 1 will introduce you to data management files, while Module 2 & Module 3, will address the Input and Output systems and file allocation methods respectively. Finally in Module 4 I will extensively discuss data management facilities, file management system and file system architecture.

Course Goal

Your journey through this course will introduce you to the computer system. You will be able to classify computers in terms of the nature of data, generation, and purpose. You will also learn other aspects of computing like Boolean algebra, number system, and introductory programming.

WORK PLAN



Learning Outcomes

I. Trace the Historical Development of Computer Systems;

II. Mention the advantages, disadvantages and characteristics of Computer

Module 01

III. Differentiate among computer Software, Hardware and Human ware in relation to their functions.

IV. Perform various calculations in number systems (binary, decimal, octal decimal, and hexadecimal)

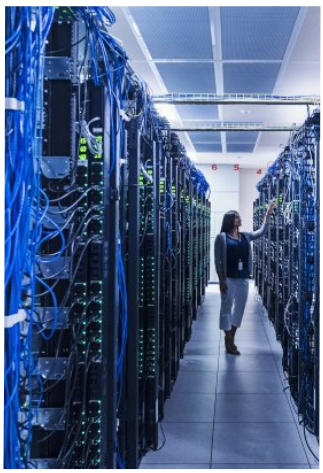
Module 02

Course Guide
Course Content

Module 1

Computers

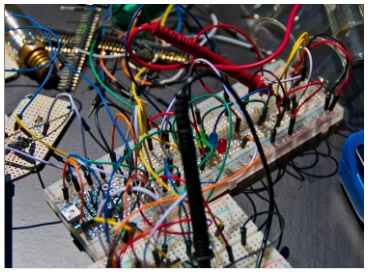
In this unit, I will explain the basics of computer systems, the historical development of computers, as well as their classification. I will also categorize computers based on sizes, nature of data they process, their generations, and purposes.



Module 2

Number Bases and Computer Arithmetic

Having learnt about the number base system, it is important that you also learn how numbers can be converted from one base to the other. In this unit, you will learn about how numbers can be converted from base 10 to other number bases such as base 2, base 8 and base 10, respectively.



Related Courses

There are no pre-requisites for this course, however this course is also required for the folloiwng courses.

CSC 112
Introduction to Computer Science II

CSC 230
Computer Architecture

CSC 321
Introduction to Digital Design and Microprocessors

V. Explain Boolean algebra, DeMorgan's Theorem and Karnaugh Map

VI. Explain Logic Gates, Combinatorial Logic Circuit and forms of Logic Gate

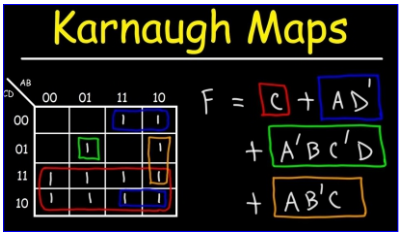
VII.Discuss generations of programming languages

Module 03 → Module 04 → Module 05

Module 3

Boolean Algebra and Karnaugh Map

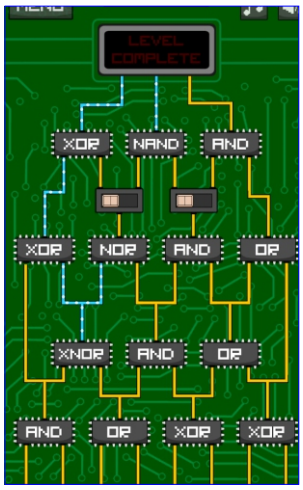
This unit covers Boolean Algebra and its operators as well as truth tables. as well as the use of Boolean operators such as AND operator represented with a dot (.), the OR operator represented with plus (+) and the NOT operator, which is an inverter.



Module 4

Logic Gates

In this Module, I will explain the basic logic gates and Combinatorial Logic Circuits. The basic gates are AND, OR, and the NOT gate. Other advanced gated are developed using combinations of the basic gates like NAND, NOR. EXOR, AND EXNOR



Module 5

Computer Programming Languages

Programming languages are known to be the medium through which human beings communicate with the computer. Different programming languages have evolved over the years, and each has its target and features. The features of the language are used to measure its strength, and this will determine how the public will accept it.

Course Requirements

Requirements for success

The CODL Programme is designed for learners who are absent from the lecturer in time and space. Therefore, you should refer to your Student Handbook, available on the website and in hard copy form, to get information on the procedure of distance/e-learning. You can contact the CODL helpdesk which is available 24/7 for every of your enquiry.

Visit CODL virtual classroom on http://codllms.unilorin.edu.ng. Then, log in with your credentials and click on CSC 111. Download and read through the unit of instruction for each week before the scheduled time of interaction with the course tutor/facilitator. You should also download and watch the relevant video and listen to the podcast so that you will understand and follow the course facilitator.

At the scheduled time, you are expected to log in to the classroom for interaction. Self-assessment component of the courseware is available as exercises to help you learn and master the content you have gone through.

You are to answer the Tutor Marked Assignment (TMA) for each unit and submit for assessment

Embedded Support Devices

Requirements for success

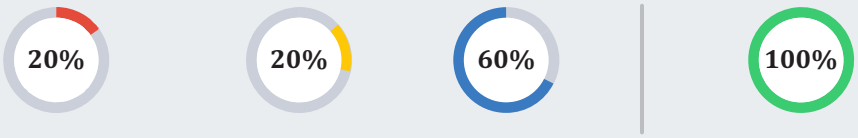
Throughout your interaction with this course material, you will notice some set of icons used for easier navigation of this course materials. We advise that you familiarize yourself with each of these icons as they will help you in no small ways in achieving success and easy completion of this course. Find in the table below, the complete icon set and their meaning.

 **Introduction**	 **Learning Outcomes**	 **Main Content**

 Summary	 Tutor Marked Assignment	 Self Assessment
 Web Resources	 Downloadable Resources	 Discuss with Colleagues
 References	 Futher Reading	 Self Exploration

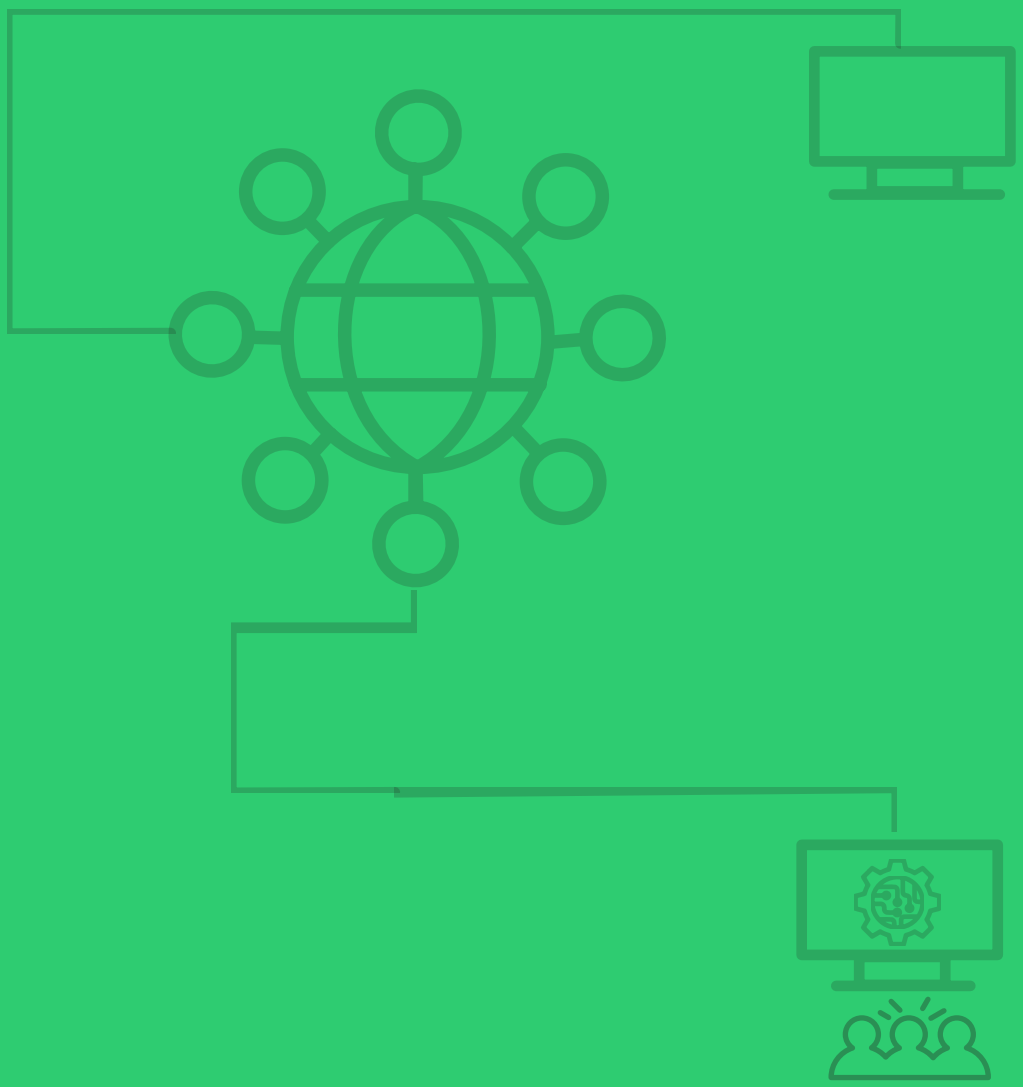
	SAQ (n) provides you with clues to answering equivalent Self Assessment Question(s). Where "n" serves as the corresponding SAQ number.
	T mins suggests the estimated amount of time it will take you to complete a block of instruction. This is based on the estimation of 300 words per minute for adults (see: https://irisreading.com/what-is-the-average-reading-speed)

Grading and Assessment

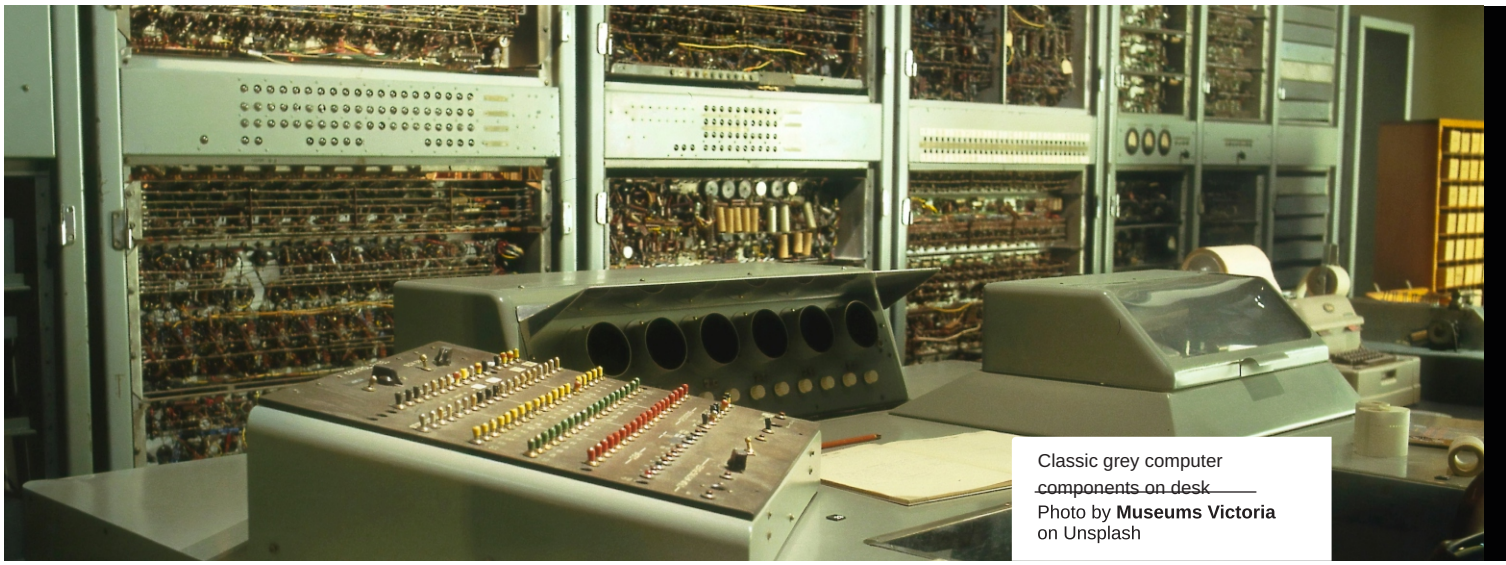


Module 1

COMPUTERS




85.00%



Classic grey computer components on desk
Photo by **Museums Victoria** on Unsplash

UNIT 1

Definition and History of Computers

Introduction

In this unit, I will explain the basics of computer systems, the historical development of computers, as well as their classification. I will also categorize computers based on sizes, nature of data they process, their generations, and purposes.

Learning Outcomes

At the end of this unit, you should be able to:

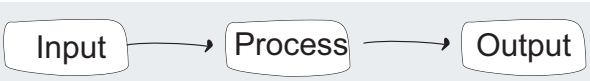
- Define a computer;
- Describe the historical development of computers; and
- Describe the nature of data computers can process.

Main Content

Basic Computing Concepts

05 mins

A computer can be described as an electronic device that accepts data as input, processes the data based on a set of predefined instructions called a program and produces the result of these operations as output called information. From this description, we can then say that"a computer can be referred to as an Input-Process-Output (IPO) system",



Data

Data are raw facts. For example, a score of 55 in an examination or Malik as the name of a student. There are three types of data – Numeric, Alphabetic, and Alphanumeric.

· Numeric Data: consists of digits 0 – 9 any combination of these digits are also numeric data. Data such as age, currency, etc. are numeric data because they contain numbers. My age is 35 years. 35 is a combination of digits; it is a numeric data.

· Alphabetic Data: these consist of any of the English language alphabets in upper and lower cases (e. g. Toyin).

· Alphanumeric Data: these can include a number, an alphabet or a special character, such as a vehicle plate number (e. g. AE731LRN).

An Exam score is an example of **Numeric data**

A Name tag is an example of **Alphabetic data**

02 | Information

As we have seen from the previous section, data makes no complete sense. However, when it is transformed into a more meaningful and useful form, it is called information.

This transformation process usually involves a series of operations to be performed by the computer on the raw data that is fed into the computer system. This operation can be arithmetic (such as addition, subtraction, multiplication, and division), logical comparison or character manipulation (as in text processing).

03 | Logical comparison

Logical comparison means testing whether one data item is greater than, equal to, or less than another. Based on the outcome of the comparison, a specified action can be taken. The output of the processing can be in the form of reports which can be displayed or printed.

Activity 1: Match the following data with their data-type

Data	Data Type
Tayo's Age	Alphanumeric
Unilorin Matriculation Number	Alphabetic
Nigeria's Car Plate Number	Numeric
The name of your CSC 111 E-tutor	Alphanumeric

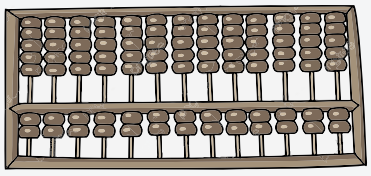
The History of Computers

In this section, I will quickly give a brief account of the history of the computer system. It may sound interesting to know that our forefathers, in the early days, before the advent of computers, used fingers and toes for counting. Later on, sticks and pebbles were used. They keep permanent records of the result of counting by putting marks on the ground, wall and so on using charcoal, chalk, and plant juice.

With the earlier mentioned human practice in mind, I will focus more on the historical development of the digital computer from the days of the abacus to the modern electronic computer. Some of the people whose contributions to the development of Computer have been widely acknowledged will be discussed.

The Abacus

It may interest you to know that the abacus qualifies as a digital instrument because it uses beads as counters to calculate in discrete form. It is made of a board that consists of beads that slide on wires. The abacus is divided by a wooden bar or rod into two zones. Perpendiculars to this rod are wires arranged in parallel, each one representing a positional value. Each zone is divided into two levels - upper and lower. Two beads are placed on each wire in the upper zone, while five beads are arranged on each wire in the lower zone.



Do you know that the abacus was invented to replace the old methods of counting?
It is an instrument known to have been used for counting as far back as 500 B.C. in Europe, China, Japan and India, and it is still being used in some parts of China today

The abacus can be used to perform arithmetic operations such as addition and subtraction efficiently.

"**Note** that the abacus is just a representation of the human fingers: the 5 lower rings on each rod represent the 5 fingers, and the 2 upper rings represent the 2 hands."

Watch Video
How Abacus Machine Works

| Blaise Pascal

Pascal was born at Clermont, France in 1623 and died in Paris in 1662. Pascal was a Scientist as well as a Philosopher. He started to build his mechanical machine in 1640 to aid his father in calculating taxes. He completed the first model of his

machine in 1642, and it was presented to the public in 1645.

It may interest you to note that the machine, called Pascal machine or Pascaline, was a small box with eight dials that resembled the analogue telephone dials. Each dial is linked to a rotating wheel that displayed the digits in a register window. Pascal's main innovative idea was the linkage provided for the wheels such that an arrangement was made for a carry from one wheel to its left neighbour when the wheel passed from a display of 9 to 0. The machine could add and subtract directly.

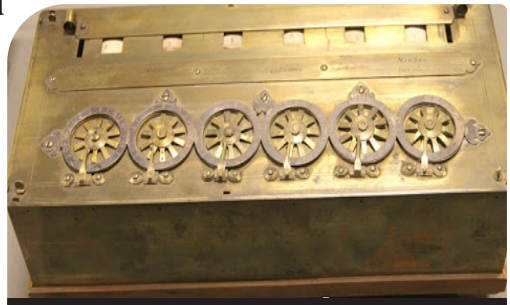


Figure 1.3: **Pascal's Pascaline**
Source: **© 2002 IEEE**

Watch Video 2
How Pascaline Machine works

Note: A Pascaline opened up so you can observe the gears and cylinders which rotated to display the numerical result

Joseph Marie Jacquard

In 1801, the Frenchman Joseph Marie Jacquard invented a power loom that could base its weave (and hence the design on the fabric) upon a pattern, automatically read from punched wooden cards, held together in a long row by rope. Descendents of these punched cards have been in use ever since.

Figure 1.4:
Jacquard's Loom showing the threads and the punched cards

Figure 1.5: By selecting particular cards for Jacquard's loom you defined the woven pattern [photo © 2002 IEEE]

| Charles Babbage

Charles Babbage was born in Totnes, Devonshire on December 26, 1792 and died in London on October 18, 1871. He was educated at Cambridge University where he studied Mathematics. In 1828, he was appointed Lucasian Professor at Cambridge. Charles Babbage started work on his analytic engine when he was a student. His objective was to build a program-controlled, mechanical, digital computer incorporating a complete arithmetic unit, store, punched card input and a printing mechanism.

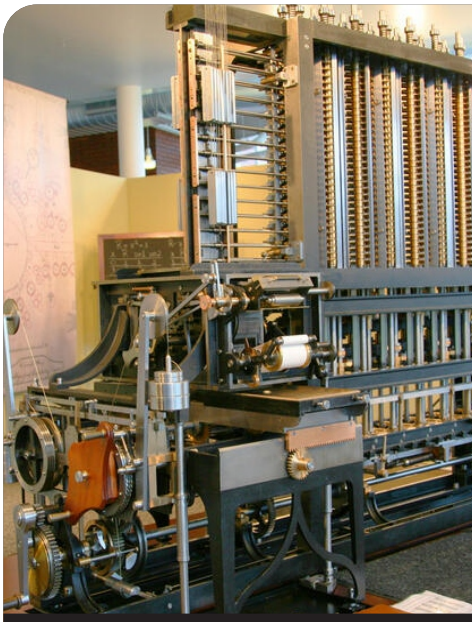


Figure 1.6:
Barbage's Analytics Engine

The program was to be provided by the set of Jacquard cards. However, Babbage was unable to complete the implementation of his machine because the technology available at his time was not adequate to see him through. Moreover, he did not plan to use electricity in his design. It is noteworthy that Babbage's design features are very close to the design of the modern computer. Babbage invented the modern postal system, cowcatchers on trains, and the ophthalmoscope, which is still used today to treat the eye

| Augusta Ada Byron

Ada Byron was the daughter of the famous poet Lord Byron and a friend of Charles Babbage, (Ada later became Countess Lady Lovelace by marriage). Though she was only 19, she was fascinated by Babbage's ideas, and through letters and meetings with Babbage, she learned enough about the design of the Analytic Engine to begin fashioning programs for the still unbuilt machine. Babbage refused to publish his knowledge for another 30 years. Ada, however, wrote a series of "Notes" in which she detailed sequences of instructions prepared for the Analytic Engine. The Analytic Engine remained unbuilt, but Ada earned her spot in history as the first computer programmer. Ada invented the subroutine and was the first to recognize the importance of looping.

| Herman Hollerith

Hollerith was born at Buffalo, New York in 1860 and died at Washington in 1929. Hollerith founded a company which merged with two other companies to form the Computing Tabulating Recording Company. In 1924, its name changed to International Business Machine (IBM) Corporation, a leading company in the

Watch Video
Hollerith desks
Photo courtesy:
The Computer Museum

manufacturing and sales of computer today.

Hollerith, while working at the Census Department in the United States of America, became convinced that a machine based on cards can assist in the purely mechanical work of tabulating population. This also made similar statistics feasible. He left the Census in 1882 to start work on the Punch Card Machine which is also called Hollerith desks.

This machine system consisted of a punch, a tabulator with a large number of clock-like counters and a simple electrically activated sorting box for classifying data in accordance with values punched on the card. The principle he used was simply to represent logical and numerical data in the form of holes on cards.

His system was installed in 1889 in the United States Army to handle Army Medical statistics. He was asked to install his machine to process the 1890 Census in the USA. This he did, and in two years, the processing of the census data which used to take ten years was completed. Hollerith's machine was used in other countries such as Austria, Canada, Italy, Norway and Russia.

John von Neumann

John von Neumann was born on December 28, 1903, in Budapest, Hungary and died in Washington D. C. on February 8, 1957. He was a great mathematician with significant contribution to the theory of games and strategy, set theory and the design of high-speed computing machines. In 1933, he was appointed one of the first six professors of the School of Mathematics in the Institute for Advanced Study at the Princeton University, USA, a position he retained until his death.

Neumann with some other people presented a paper titled “The Preliminary Discussion of the Logical Design of an Electronic Computing Instrument” popularly known as von Neumann machine. This paper contains revolutionary ideas on which the present-day computers are based. The machine has Storage, Control, Arithmetic and input/output units. The machine was to be a general-purpose computing machine. It was to be an electronic machine and introduced the concept of a stored program. The idea implied that operations in the computer were controlled by a programme stored in its memory. This program

was to consist of codes that intermixed data with instructions.

As a result of this, it became possible for computations to proceed at electronic speed, perform the same set of operations or instructions repeatedly and the concept of program counter, which implied that whenever an instruction is fetched, the program counter which is a high-speed register automatically contains the address of the instruction to be executed next.

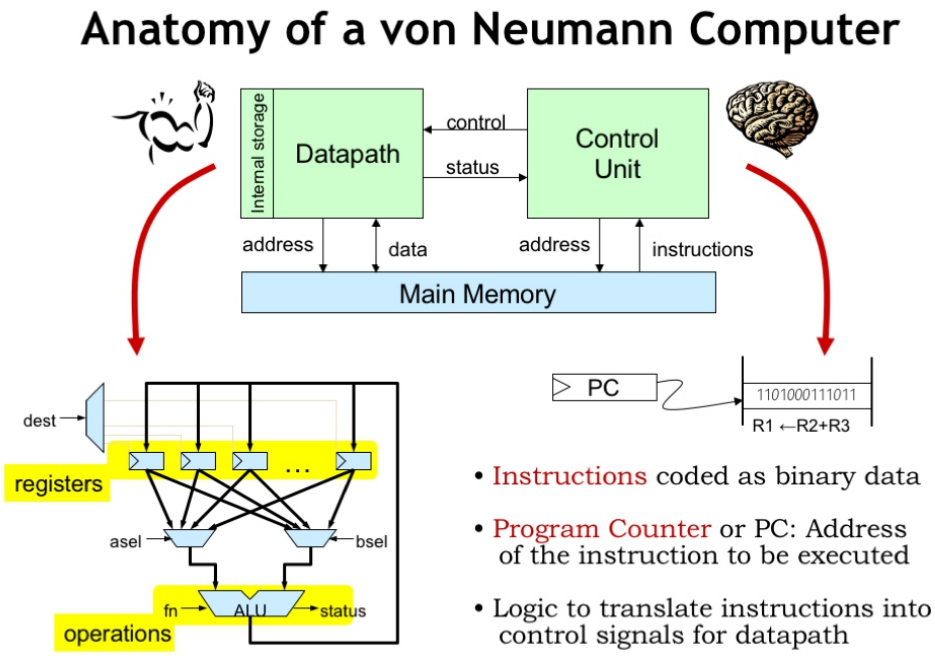


Figure 1.7:
Anatomy of a von Neumann Computer
Source: Slideplayer

J. V. Atanasoff

One of the earliest attempts to build an all-electronic digital computer occurred in 1937 by J. V. Atanasoff, a professor of Physics and Mathematics at Iowa State University. By 1941, he and his graduate student, Clifford Berry, had succeeded in building a machine that could solve 29 simultaneous equations with 29 unknowns. This machine was the first to store data as a charge on a capacitor, which is how today computers stored information is in their main memory. It was also the first to employ binary arithmetic. However, the machine was not programmable. It lacked a conditional branch, and its design was appropriate for only one type of mathematical problem. Moreover, it was not further pursued after World War II.

Figure1.8: The Atanasoff-Berry Computer
[photo © 2002 IEEE]

Watch Video 6: Atanasoff-Berry Computer

Source: Computer History Museum YouTube

Watch Video 06
Atanasoff-Berry Computer

Figure1.9: The Harvard Mark I:
An electro-mechanical computer

Watch Video 07
The Harvard Mark I

| Howard Aiken

Howard Aiken of Harvard was the principal designer of the Mark I. The Harvard Mark I computer was built as a partnership between Harvard and IBM in 1944. This was the first programmable digital computer made in the U.S., but it was not a purely electronic computer. Made up of switches, relays, rotating shafts, and clutches, the machine weighed 5 tons. It was 8 feet tall, 51 feet long and incorporated 500 miles of wire. It also had a 50ft rotating shaft running its length, turned by a 5-horsepower electric motor. The Mark I ran non-stop for 15 years.

Grace Hopper

Grace Hopper was one of the primary programmers for Mark I computer. Hopper found the first computer "bug": a dead moth that had gotten into the Mark I and whose wings were blocking the reading of the holes in the paper tape. The word "bug" had been used to describe a defect since at least 1889. Still, Hopper is credited with coining the term "debugging" to describe the work of eliminating program faults.

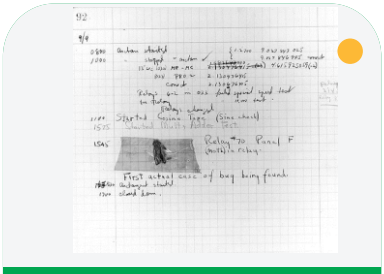


Figure 1.11:
The first computer bug

In 1953 Grace Hopper invented the first high-level language, "Flow-Matic". This language eventually became COBOL which was the language most affected by the infamous Y2K problem. A high-level language is designed to be more understandable by humans than is the binary language understood by the computing machinery. A high-level language is worthless without a program -- known as a compiler -- to translate it into the binary language of the computer. Hence, Grace Hopper also constructed the world's first compiler. Grace remained active as a Rear Admiral in the Navy Reserves until she was 79.

Figure 1.12:
The Microsoft windows 1.0

Bill Gates

William (Bill) H. Gates was born on October 28, 1955, in Seattle, Washington, USA. Bill Gates decided to drop out of college so he could concentrate all his time writing programs for Intel 8080 categories of Personal Computers (PCs). This early experience put Bill Gates in the right place at the right time once IBM decided to standardize on the Intel microprocessors for their line of PCs in 1981. Gates founded a company called Microsoft Corporation (together with Paul G. Allen) and released its first operating system called MS-DOS 1.0 in August 1981 and the last of its group (MS-DOS 6.22)in April 1994. Bill Gates announced Microsoft Windows on November 10, 1983.

Philip Emeagwali

Philip Emeagwali was born in 1954, in the Eastern part of Nigeria. He had to leave school because his parents couldn't pay the fees, and he lived in a refugee camp during the Nigerian civil war. He won a scholarship to university. He later migrated to the United States of America. In 1989, he invented the formula that used 65,000 separate computer processors to perform 3.1 billion calculations per second.

Philip Emeagwali is a supercomputer and Internet pioneer. He is regarded as one of the fathers of the internet due to his invention of an international network similar to, but predated the Internet. He also discovered mathematical equations that enabled the petroleum industry to recover more oil. Emeagwali won the 1989 Gordon Bell Prize, computation's Nobel Prize, for inventing a formula that lets computers perform the fastest computations, a work that led to the reinvention of supercomputers.

Inventors

Activity 2

Match the following inventions with their inventors

Inventor	Invention
Abacus	First Programmer
Grace Hooper	Punch Card
J. V. Atanasoff	Counting Machine
Augusta Ada Byron	Debugging
Herman Hollerith	Binary Arithmetic

Summary

In this unit, you have learnt that:

·A computer is an electronic device that accepts data as input, process the data through predefined instructions to produce information.
·The historical development of the computer was from the Abacus to the modern electronic computer.

Self-Assessment Questions

1. Define a computer;
2. Describe the historical development of computers; and
3. Describe the nature of data computers can process.

Tutor Marked Assessment

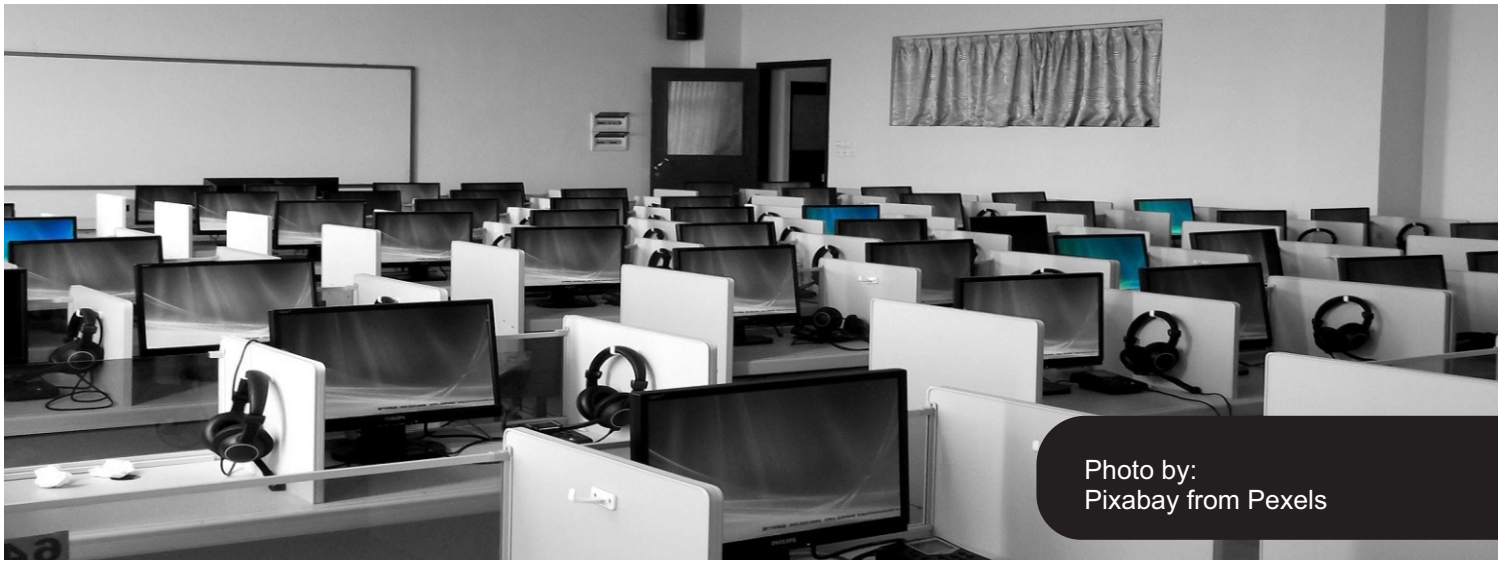
- State the contribution(s) of John von Neumann and two (2) other persons to the development of Computer.

Further Reading

- https://en.wikipedia.org/wiki/Herman_Hollerith
- http://www.byte-notes.com/types-computers-purpose
- https://www.webopedia.com/DidYouKnow/Hardware_Software/FiveGenerations.asp
- https://www.tutorialspoint.com/computer_fundamentals/computer_first_generation.htm
- https://en.wikipedia.org/wiki/File:Hollerith.jpg
- http://www.computerhistory.org/babbage/
- http://www.computerhistory.org/babbage/engines/
- https://www.computerhope.com/jargon/p/punccard.htm
- https://www.computerhistory.org/revolution/early-computer-companies/5/100

References

Stenzel, N. (n.d): Basic File Management and organization. The University of Maryland Extension. Retrieved on from http//extension.umd.edu



UNIT 2

Classification of Computers

Introduction

In unit 1, you learnt about the meaning and history of computers. Here, I will categorize computers based on their generations, sizes, nature of data they process, and their purposes.

Learning Outcomes

At the end of this unit, you should be able to:

- Compare generations of computers;
- Identify computers based on the nature of data they process;
- Classify computers by size; and
- Classify computers by purpose.

Main Content

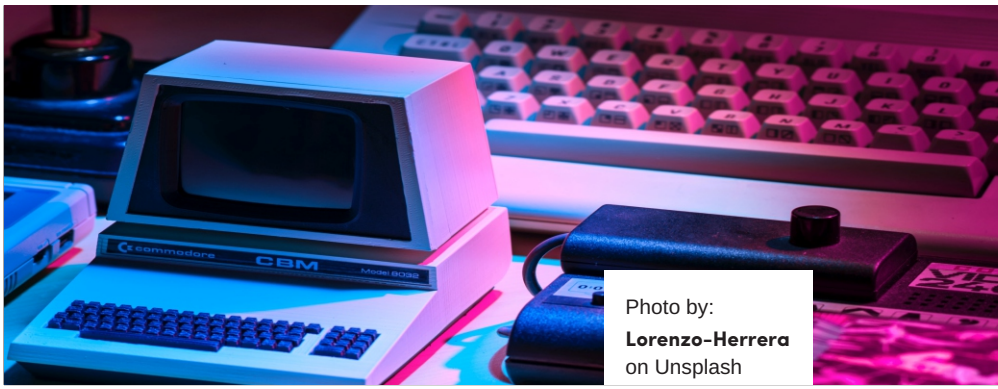


Photo by: **Lorenzo-Herrera** on Unsplash

Computer can be classified into various forms because of the complexity and diversification in their application. We will adopt four primary classifications, namely, generations, nature of data, size and purpose.

Classification By Generation

We have five generations to which computers can be classified based on year and components.

I. **First Generation Computer:** These were the early computers that were manufactured in 1940 and lasted till 1956. The first-generation computers were characterized by the use of vacuum tubes as its major components. These vacuum tubes generate enormous heat and consume much electricity. The first generation of computers introduces the concept of stored programs. Exclusively, computer experts can program the computer only in machine language, which makes it programmable. Examples are UNIVAC (Universal Automatic Computer), ENIAC (Electronic Numerical Integrator and Calculator) etc.

Figure 1.13 UNIVAC I supervisory

II. **Second Generation Computer:** These set of computers succeeded the first generation of computers. Their advent lasted from 1956 to 1963. The components of the second generation of computers were built around transistors which replaced the vacuum tube in the first generation. The resultant

effect of the transistors in place of the vacuum tube is a reduction in size compared with first-generation computers, less power consumption, generation of less heat and improved storage facility due to introduction of magnetic devices for a storage medium. The overall effects are the improved reliability and introduction of symbolic languages for programming. Examples are ATLAS, IBM 1400 series (International Business Machine), PDP I & II (Programmed Data Processor I & II) etc.

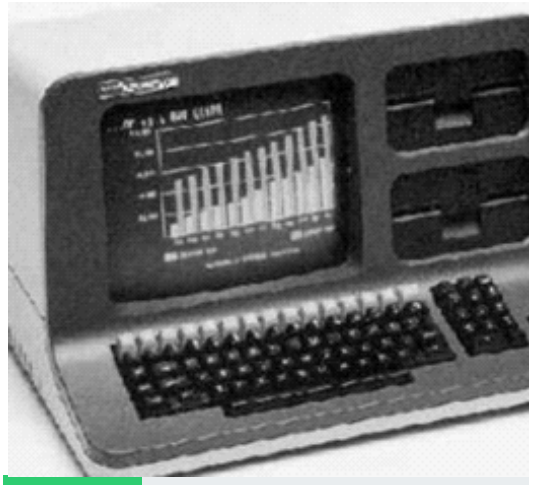


Figure 1.13 **Second Generation**

III. **Third Generation Computer:** This generation of computer came after the second generation of computers. Their advent was between 1964 to late 1971. Due to the technological advancement that had taken place in the industrial sector, many transistors had to be coupled into a single unit component. Hence, the major component that characterized the third-generation computer is the Integrated Circuitry (IC), which is a resultant effect of thousands of transistors combined into a single unit component. The integration of transistors into one component makes the computer smaller in size compared with first and second-generation computers. It also made the machines faster, consume less power and generates less heat. The concept of multi-programming was introduced in this generation of computers. Programming was made easier by the use of high-level languages. Examples include IBM 360/370 series, ICL 1900 series (International Computers Limited) etc.

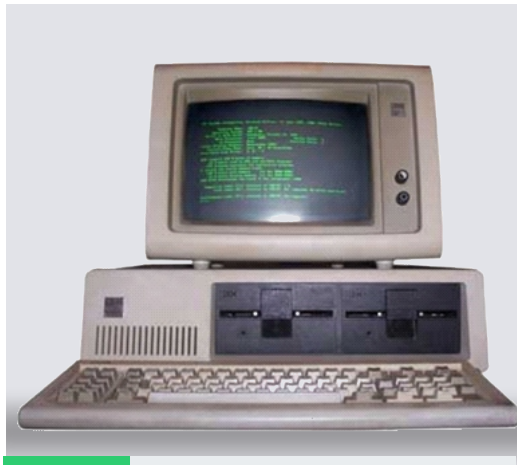


Figure 1.14 **Third Generation**

IV. Fourth Generation Computer:

Figure 1.15 Fourth Generation

The emphasis in the first three generations of computers has been on the development of a computer system that is less expensive, more portable and highly reliable. The fourth-generation computers were also developed with the above assertions in mind. This generation of computers was built around Very Large-Scale Integrated Circuitry (VLSIC). This allowed over ten thousand flip-flops to be placed in a single silicon chip, i.e. thousands of ICs were combined into a single chip. This was the era of microcomputers with the introduction of microprocessors as its main component. The system came into being in 1971 and is still in existence till date. Examples include COMPAQ 2000 series, DEC 10, STAR 1000, PDP 11,etc

Figure 1.16 Fifth Generation

V. **Fifth Generation Computer:** The development of the fifth-generation computers started in the 1980s, and classical researches are still ongoing on this generation of computers. It is regarded as the present and future of computers. Some of these machines are already in use. However, a lot of work is still being done to improve it. The objective of this computer system is to mimic human intelligence in domains such as medicine, law, education, criminal investigation, etc. This objective is achieved through the implementation of Artificial Intelligence and Expert Systems development.

Classification By Nature of Data

A computer can process two types of data- Digital and Analogue. Computer types can be classified into three based on the types of data they process. These are Analogue, Digital and Hybrid computers.

I. Analogue Computer: Analogue computer deals with quantities that vary continuously. It measures changes in current, temperature or pressure and translates these data into electrical pulses for processing. Examples are speedometer, electric meter, water meter, thermometer etc.

II. Digital Computer: This operates on data representation in form of discrete values or digits (e.g 0,1,2,3,X,Y,Z,...). They handle numbers discretely and precisely rather than approximately. These types of computers are prevalent in use both at home and offices.

III. Hybrid Computer: These types of computers combine the features of both analogue and digital computers. They handle data in both discrete quantity and variable quantities. They are mostly found in industrial processes for data acquisition and data processing purposes. In most cases, an analogue signal generated from the analogue computer needs to be converted to a digital signal which has to be processed by the digital computer. Hence, the need for Analogue-to-Digital Converter and Digital-to-Analogue Converter modulator/demodulator (Modem).

Classification By Size

Computer classifications based on sizes are as follows:

I. Mainframe Computer: Mainframe Computer is a system that has a very powerful Central Processing Unit (CPU) linked by cable to hundreds or thousands of terminals, this system is cable of accepting data simultaneously from all the terminals at the same time. This type of computers is very big and very expensive general-purpose computers with memory capacity more than

Mainframe Computer
Source: Pixabay

100 million bytes and processing power of well above 10 million instructions per second (MIPS). Mainframe computers are used in large organizations such as banks, oil companies, big hospitals, airline reservations companies, examination bodies such as WAEC, NECO, JAMB

etc. that have very large volumes of data to process which also need to be adequately secured. Examples include ICL 1900 and IBM 360/370 series, IBM 704 etc.

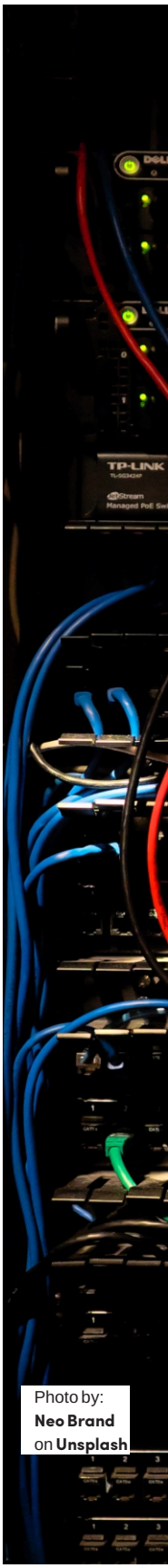


Photo by: **Neo Brand** on **Unsplash**

I. Minicomputer: This type of computer shares similar features with a mainframe computer, but it is smaller in physical size; generates lower heat, less instruction set and less expensive. It requires the same conditions for its operation like mainframe; such conditions are very cool environment because of the enormous heat being generated, raise or false floor, dust-free environment, high-secure office accommodation. Examples of minicomputers are IBM AS/400, NCR Tower 32, DEC System, PDP 7 etc.

II. Microcomputer: These types of computer are much smaller in size compare to mini and mainframe computers. They are ridiculously cheaper in terms of naira value compare to either mainframe or minicomputer. On these systems, various integrated circuits and elements of a computer are replaced by a single integrated circuit called “chip”. The microcomputer was first developed by companies like Apple Computers and later by IBM PC in 1981. It is also called the Personal Computer (PC).

III. Super Computers: The supercomputers are extraordinarily powerful computers and they are the largest and fastest computer systems in recent time. They provide a high level of accuracy, precision and speed for mathematical computations, meteorological, astronomical and oil exploration applications. In most of Hollywood's movies, it is used for animation purposes. It is also helpful for forecasting weather reports worldwide. Examples are Cray-1, Cyber series, Fujistu, ETA-10 system. Most of these machines are not available for commercial use.

IV. Notebook Computers: The have small size and low weight. The notebook is easy to carry anywhere. This is easy to carry around and preferred by students and businessmen and women to meet their assignments and other necessary tasks. The approach of this computer is also the same as the personal computer. It is a replacement of personal desktop or microcomputer. Also referred to as laptop. E.g. HP 530, Dell etc

Other types of computers based on size are palmtop and PDA (Personal Digital Assistant).

Classification By Purpose

There are two classifications to which computers are put according to their usage or function. These are special-purpose and general-purpose computers.

I **Special Purpose Computer:** This type of computer is specially developed to perform only one task. The system is highly efficient and economical but lacks flexibility. The program for the machine is built into the machine permanently. Some areas of usage include air traffic control system; military weapons control system, ship navigation system and industrial process controls.

II **General Purpose Computer:** These computers can handle a wide variety of programs and to solve many problems such as payroll, numerical analysis, software development for accounting, inventory system etc. It makes use of a stored program for switching from one application to another.

Computer Classification

Activity 3

Visit the following places and note a special purpose machine there, then discuss your findings with your colleagues

Places	Special Purpose Machine
Fuel Station	Fuel Dispensing Machine
Supermarket	
Bank	
Library	

Summary

In this unit, you have learnt that:

Computers can be classified based on generations, nature of data they can process, sizes as well as the purposes for which they are built.

Self-Assessment Questions

1. With appropriate examples, differentiate between Analogue and Digital Computers
2. Describe the classification of Computers by Purpose
3. Distinguish computers based on their generations;
4. Classify computers by size;
5. Classify computers by purpose.

Tutor Marked Assignment

- List two (2) examples of computers classified as special-purpose computers

  Differentiate between Mainframe Computer and Super Computers

  Notebook Computer is a replacement for microcomputer. Yes/No, Justify your answer.

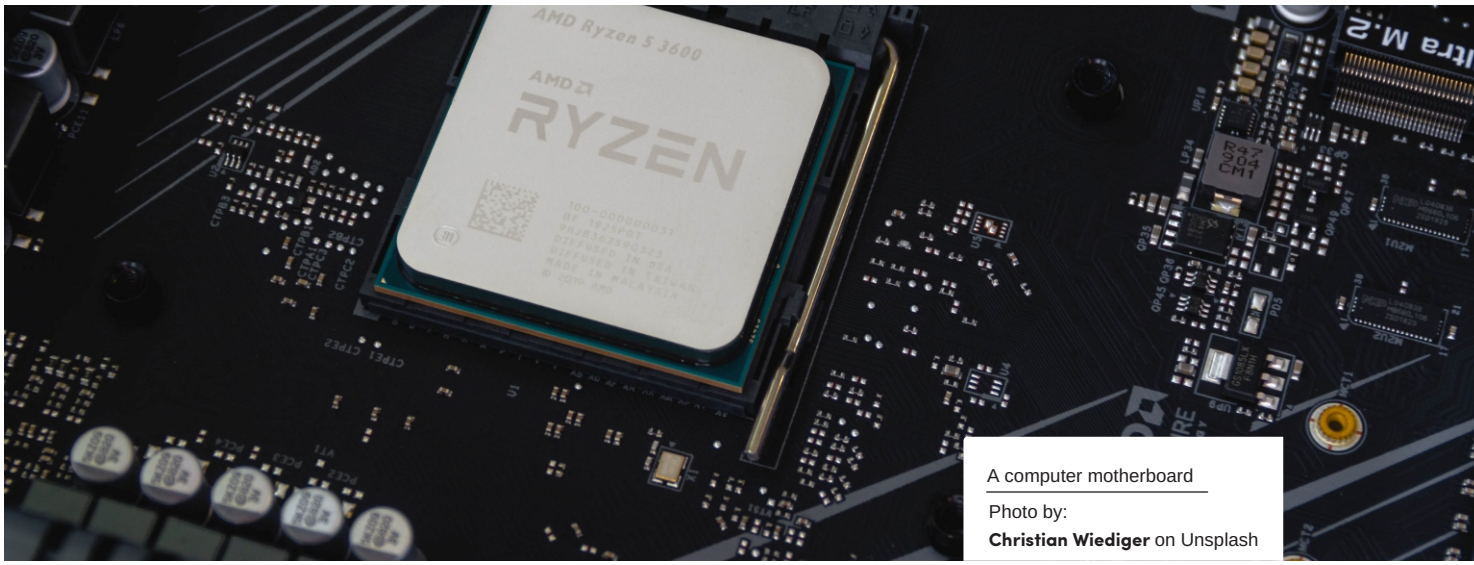
  Identify the major component or characteristic of the first four generations of Computers

References

- Egbewole, W. and Jimoh R. (Eds.). (2017) Digital Skill Acquisition. Ilorin, Nigeria: Unilorin
- Dale, N. (2005), Computer Science Illuminated. London: Jones and Bartlett.
- French C. (2001). Introduction to Computer Science. London: Continuum. z

Further Reading

- https://en.wikipedia.org/wiki/Herman_Hollerith
- http://www.byte-notes.com/types-computers-purpose
- https://www.webopedia.com/DidYouKnow/Hardware_Software/FiveGenerations.asp
- https://www.tutorialspoint.com/computer_fundamentals/computer_first_generation.htm
- https://en.wikipedia.org/wiki/File:Hollerith.jpg
- http://www.computerhistory.org/babbage/
- http://www.computerhistory.org/babbage/engines/
- https://www.computerhope.com/jargon/p/punccard.htm
- https://www.computerhistory.org/revolution/early-computer-companies/5/100



A computer motherboard

Photo by:
Christian Wiediger on Unsplash

UNIT 3

Components of a Computer

Introduction

In this unit, I will explain the basics of computer systems, the historical development of computers, as well as their classification. I will also categorize computers based on sizes, nature of data they process, their generations, and purposes.

Learning Outcomes

At the end of this unit, you should be able to:

- Differentiate among hardware, software and humanware; and
- Illustrate the organization structure of a computer installation

Main Content

picture: A Computer Desktop

Source: Unsplash.com

Basic Components of a Computer

(1Min)

Components of Computer refer to the physical and non-physical part of the system. A computer system can be divided into hardware, software and humanware. The hardware refers to the visible components of the system, while the software refers to the non-visible components of the computer system. The humanware deals with the human intervention that the computer needs to perform its tasks.

The Hardware

The hardware refers to the physical components and the devices which make up the visible parts of the computer. It can be divided into two:

Central Processing Unit (CPU) :

The CPU is responsible for all processing that the computer does. The CPU consists of

- Main storage: The main storage is used for storing data to be processed as well as the instructions for processing them.
- Arithmetic Logic Unit (ALU): ALU is the unit for arithmetic and logical operations.
- Control Unit (CU): The control unit ensures the smooth operation of the other hardware units. It fetches instruction, decodes (interprets) the

instruction and issues command to the units responsible for executing the instructions

The Peripherals

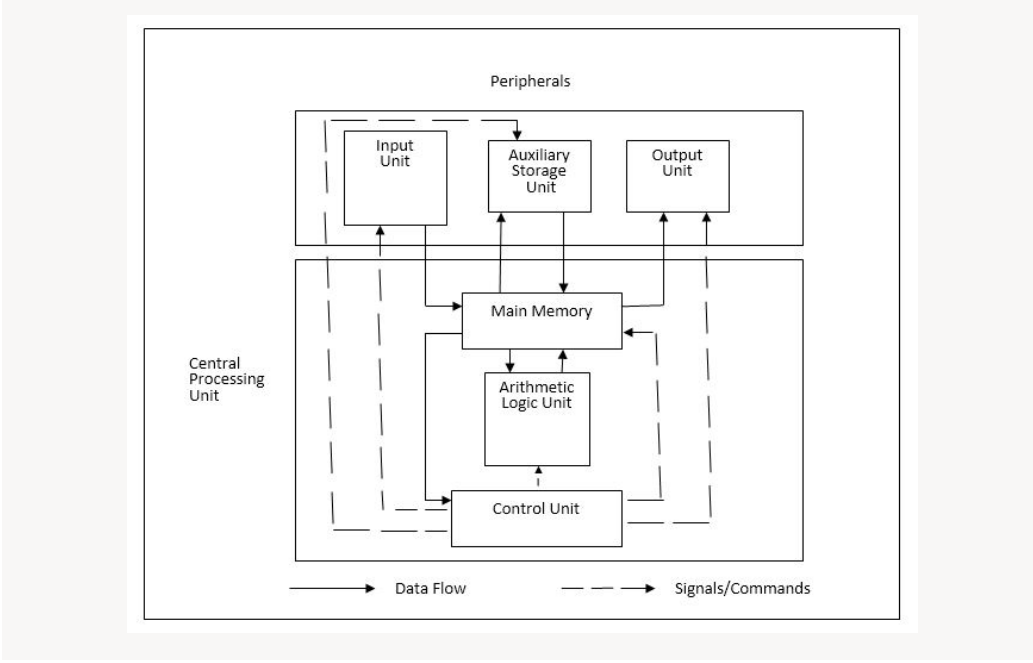
·These are responsible for feeding data into the system and for collecting information from the system. The peripherals are in three categories:

Input devices: Input devices are used for supplying data and instructions to the computer. Examples are terminal Keyboard, Mouse, Joystick, Microphone, Scanner, Webcam, etc.

Output devices: Output devices are used for obtaining result (information) from the computer. Examples are Printers, Video Display Unit (VDU), loudspeaker, projector etc.,

Auxiliary storage devices. Auxiliary Storage Devices are used for storing information on a long-term basis. Examples are hard disk, flash disk, magnetic tape, memory card, solid-state drive SDD etc.

A simple model of the hardware part of a computer system is shown below:



Model of Hardware Computer

Software is basically referred to as programs. A program consists of a sequence of instructions required to accomplish well-defined tasks. Examples of such tasks include:

System Software

System software are programs commonly written by computer manufacturers, which have a direct effect on the control, performance and ease of usage of the computer system. Examples are Operating System, Language Translators, Utilities and Service Programs, and Database Management Systems (DBMS).

Operating System

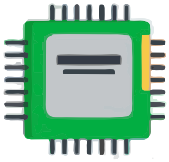
Operating System is a collection of program modules which form an interface between the computer hardware and the computer user. Its main function is to ensure judicious and efficient utilization of all the system resources (such as the processor, memory, peripherals and other system data) as well as to provide programming convenience for the user. Examples are Unix, Linux, Windows, Macintosh, and Disk Operating system.

Language Translators

Language Translators are programs which translate programs written in non-machine languages such as FORTRAN, C, Pascal, and BASIC into the machine language equivalent. Example of language translators are assemblers, interpreters, compilers and preprocessor.

Assemblers:

This is a program that converts program written in assembly language (low-level language) into machine language equivalent.

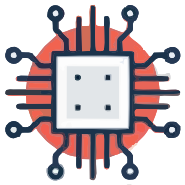


Interpreter:

This is a program that converts program written in a high-level language (HLL) into its machine language (ML) equivalent one line at a time. Language, like BASIC, is normally interpreted.

Compiler:

This is a program that translates program written in high-level language (HLL) into machine language (ML) equivalent all at once. Compilers are normally called by the names of the high-level language they translate. For instance, COBOL compiler, FORTRAN compiler etc.



Preprocessor:

This is a language translator that takes a program in one HLL and produces an equivalent program in another HLL. For example, there are many preprocessors to map structured version of FORTRAN into conventional FORTRAN

Database Management System (DBMS)

DBMS is a complex program that is used for creation, storage, retrieving, securing and maintenance of a database. A database can be described as an organized collection of related data relevant to the operations of a particular organization. The data are stored usually in a c central location and can be accessed by different authorized users.

Linker

Linker is a program that takes several object files and libraries as input and produces one executable object file. Loader is a program that places an executable object file into memory and makes them ready for execution. The operating system provides both linker and loader.

Utility and Service Programs

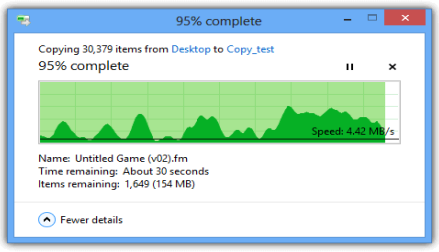
These are programs which provide facilities for performing common computing tasks of a routine nature. The following are some of the examples of commonly used utility programs:

Sort Utility:

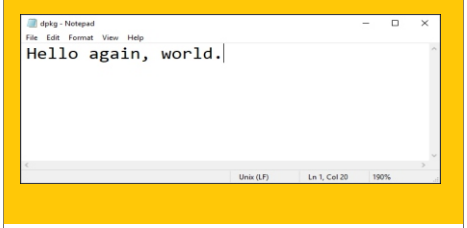
This is used for arranging records of a file in a specified sequence (alphabetic, numerical or chronological) of a particular data item within the records. The data item is referred to as the sort key.

Merge Utility:

This is used to combine two or more already ordered files together to produce a single file.



Utility Program for copying files on windows 10



Notepad Program for writing and editing texts on Microsoft Windows

Copy Utility:

This is used mainly for transferring data from one storage medium to another, for example, from disk to tape.

Debugging Facilities:

These are used for detecting and correcting errors in program.

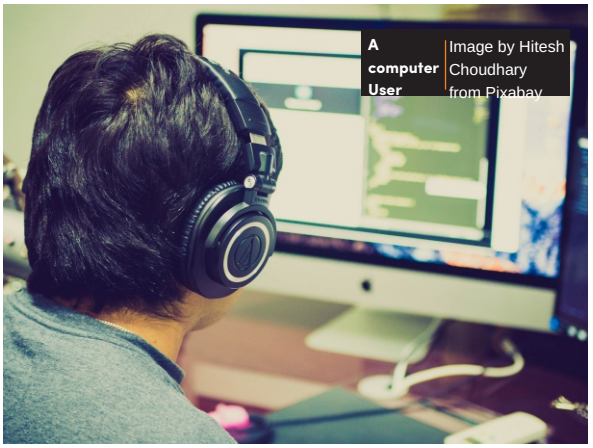
Text Editors:

These provide facilities for creation and amendment of a program from the terminal.

Benchmark Program: This is a standardized collection of programs that are used to evaluate hardware and software. For example, a benchmark might be used to compare the performance of two different computers on identical tasks, assess the comparative performance of two operating systems etc.

Application Software

These are programs written by a user to solve individual application problem. They do not have any effect on the efficiency of the computer system. An example is a program to calculate the grade point average of all the 100L students. Application software can be divided into two, namely: Application Package and User's Application Program. When application programs are written in a very generalized and standardized nature such that they can be adopted by several different organizations or persons to solve a similar problem, they are called **Application Packages**. There are a number of micro-computerbased packages. These include word processors (such as Ms-word, WordPerfect, WordStar); Database packages (such as Oracle, Ms-Access, Sybase, SQL Server, and Informix); Spreadsheet packages (such as Lotus 1-2-3 and Ms-Excel); Graphic packages (such as CorelDraw, Fireworks, Photoshop etc.), and Statistical packages (such as SPSS).



User's Application Program is a program written by the user to solve a specific problem which is not generalized in nature. Examples include writing a program to find the roots of quadratic equation, payroll application program, and program to compute students' results.

The Humanware

Although the computer system is automatic in the sense that once initiated, it can, without human intervention, continue to work on its own under the control of a

stored sequence of instructions (program). However, it is not automatic in the sense that it has to be initiated by a human being, and the instructions specifying the operations to be carried out on the input data are given by human beings. Therefore, apart from the hardware and software, the third element that can be identified in a computer system is the humanware. This term refers to the people that work with the computer system. The components of the human-ware in a computer system include the system analyst, the programmer, data entry operator, end users etc.

Organizational Structure of a Typical Computer Installation

The following diagram shows the organizational structure of a typical computer installation

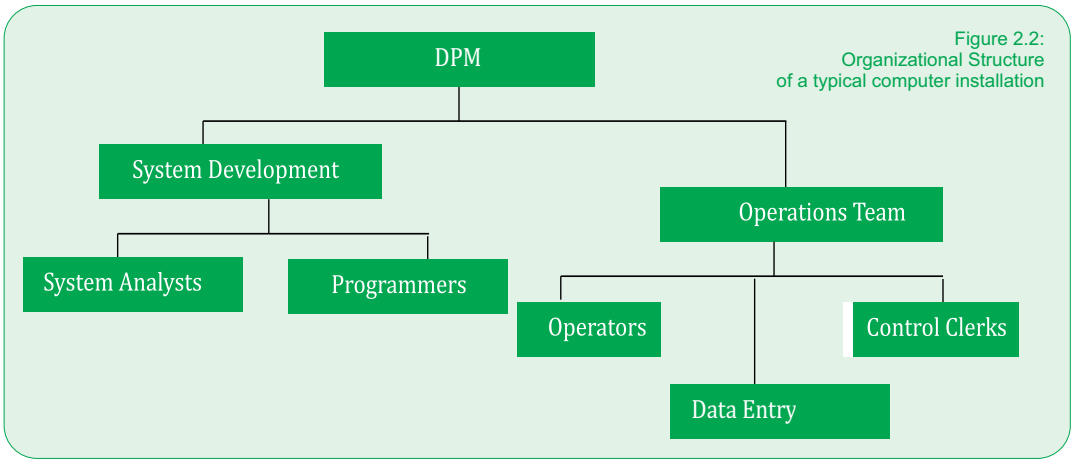


Figure 2.2:
Organizational Structure
of a typical computer installation

Data Processing Manager (DPM) supervises every other person that works with him and is answerable directly to the management of the organization in which he works.

A System Analyst is a person that understudies an existing system in operation to ascertain whether or not computerization of the system is necessary and/or cost-effective. When found necessary, he designs a computerized procedure and specifies the functions of the various programs needed to implement the system.

A Programmer is a person that writes the sequence of instructions to be carried out by the computer to accomplish a well-defined task. The instructions are given

in computer programming languages.

A Data Entry Operator: is a person that enters data into the system via keyboard or any input device attached to a terminal. Other ancillary staffs perform other functions such as controlling access to the computer room, controlling the flow of jobs in and out of the computer room.

An End-user is one for whom a computerized system is being implemented. The End-user interacts with the computerized system in their day-to-day operations of the organization. For example, a cashier in the bank who receives cash from customers or pays money to customers interacts with the banking information system.

Activity 

Visit a nearby computer firm, make a list of the computer professionals and experts in the firm and ask for their job descriptions.

Summary

In this unit, you have learnt that:

·Hardware are physical components of the system, Software are not physical and are also referred to as programs, while human ware is the interaction between man and computer

·The organizational structure includes the data processing analyst, data entry operator, end-user and so on.

Self-Assessment Questions

1. Differentiate between hardware and software
2. Draw the diagrammatic representation of a computer installation

Tutor Marked Assessment

- Explain the roles of the following personnel in an organization:
- Data Entry Operator
- Programmer
- System Analyst
- Data Processing Manager
- Define the following terms:
- Assemblerb)Compilerc)Interpreterd)Preprocessor

References

Egbewole, W. and Jimoh R. (Eds.). (2017) Digital Skill Acquisition. Ilorin, Nigeria: Unilorin

Further Reading

- https://en.wikipedia.org/wiki/Application_software
- http://dspace.mit.edu/bitstream/handle/1721.1/47060/computerizedmana00zann.pdf?sequence=1%20?iframe=true&width=100%&height=100%
- http://www.informit.com/articles/article.aspx?p=29470&seqNum=3

A computer motherboard

Photo by:
Christian Wiediger on Unsplash

UNIT 4

Characteristics and Advantages of Computer

Introduction

Objects and devices have features that help us in defining their use. In this unit, you will learn about the features or attributes that the machine must possess before it can be regarded as a computer. The advantages of computers are the reward the user or society at large get in the use of computers while the disadvantages are the negative aspect of computer systems. Today, people use computers to make work easier and faster, as well as to reduce the overall cost of completing a task.

Learning Outcomes

At the end of this unit, you should be able to:

- List the characteristics of Computer
- List the five (5) Advantages of Computer
- Mention five (5) Disadvantages of Computer

Main Content

picture: A Computer Motherboard

Source: Unsplash.com

Characteristics of Computers

The computer system is characterized by the following:

I. **Electronic in nature:** Data are represented in the form of electrical pulses. The basic component and the operation of computers are electronic, using integrated circuits (Ics).

II. **Speed:** Computers can perform millions of calculations in a few seconds when compared to man who will spend months doing the same task.

III. **Accuracy:** computers perform tasks with 100% accuracy provided that correct input has been given. It is garbage in garbage out (GIGO), i.e., given accurate input it generates accurate output, and given a wrong input, it gives a wrong output.

IV. **Consistency:** that is given the same set of input data, the same result will always be produced

V. **Iterative:** the ability to perform repetitive operations without getting bored or tired

VI. **Storage:** could store data/information on a long term basis. Variety of data/information such as text, images, audio and video.

VII. **Automatic control:** once initiated, it could operate on its own, without human intervention, under the control of stored sequences of instructions called program

VIII. **No Feelings:** Computer does not have emotions or feelings. Though some robots are now created to have feelings

Characteristics of Computers

I. Sort, Organize, and Search Information

The computer can use its stored information better than human. Information in the computer can be sorted or organized into different categories. The information can also be searched for faster.

Virtually every aspect of our life as a human has been affected by computers. We enjoy the use of computers due to many advantaged they offer. Below are some of the advantages we get from the use of computers:

II. Better Understanding of Data

Through different fields of computer technology, for example, data mining, computers understand data better than humans and can use the information learnt to predict future occurrences to a certain extent. For example, using the database, the computer can predict that at any time bread is sold, eggs or butter or beverage is sold along. This will help the business owner to know that s/he must not run out of eggs or butter or beverage when bread is available for

III. **Connection to the Internet**

The computer helps to connect to the internet, where choices available are endless once connected. Many advantages of the computer today are through connections to the internet.

IV. **Improves your abilities**

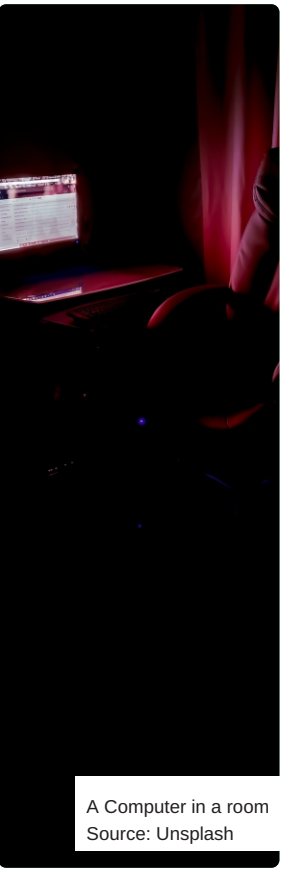
The computer can help its user to improve several aspects. For example, use of the spell checker, grammar corrector and so on. The user improves his/her abilities if s/he has a hard time learning.

V. **Assist the physically challenged**

Computers are very excellent tools that can be used to help the physically challenged. For example, Speech recognition, where the user gets to type and the computer reads it out. This can help a physically challenged user that cannot speak. Computers are also great tools for the blind with special software that can be installed to read what is on the screen.

VI. **Entertainment**

The computer can keep its user entertained. Different songs, videos, as well as games, can be stored on the computer for use.



A Computer in a room
Source: Unsplash

Disadvantages of Computers

Despite the numerous advantages that we derive from the use of computers, they also come with their disadvantages. However, the advantages overweigh the disadvantages. Some of the disadvantages are:

A. **Attack**

Attack can be in the form of virus and hacking. Virus usually spreads malicious substances on the system while hacking occurs through unauthorized access of the system.

B. **Online Cyber Crimes**

Online cyber-crime means that computer and network may be used to commit crime. Cyberstalking and Identity theft are at common cybercrimes of today.

C. **Less Human Interactions**

Computers have made it possible for people to work alone

on tasks on which they would need to collaborate with others if they were to have to do them manually, such as inputting figures in books of accounts. Therefore, the work environment in offices is such that all workers concentrate on just looking at the computer with little interaction with colleagues.

D. **Reduction in employment opportunity**

Activity 1 

Using the forum on the Learning Management System, discuss with your colleagues how you have experienced any of the disadvantages of the computer systems

Summary

In this unit, you have learnt that:

- The attributes that make a technology computer includes speed, storage, accuracy, consistency and many more
- Advantages of computer include but not limited to Entertainment, improve personal ability, a better understanding of data, aid to the physically challenged users and so on.
- Disadvantages of a computer include a reduction in employment opportunities, attacks, and less human interaction.

Self-Assessment Questions 

1. List five (5) advantages of the computer to the society
2. List five (5) characteristics of computer

Tutor Marked Assessment

- The computer is electronic. True/False. Justify your answer.
- Identify three (3) fields of computer technology that serves as an aid to humans.
- The computer is said to be garbage in garbage out, explain with appropriate example the meaning of the phrase

References

Egbewole, W. and Jimoh R. (Eds.). (2017) Digital Skill Acquisition. Ilorin, Nigeria: Unilorin

Further Reading

- http://oer.nios.ac.in/wiki/index.php/characteristics_of_computers
- http://ecomputernotes.com/fundamental/introduction-to-computer/what-are-characteristic-of-a-computer
- http://www.byte-notes.com/advantages-and-disadvantages-computers
- https://www.computerhope.com/issues/ch001798.htm

Photo by Gerd Altmann from Pixabay

Module 2

Number Bases and Computer Arithmetic

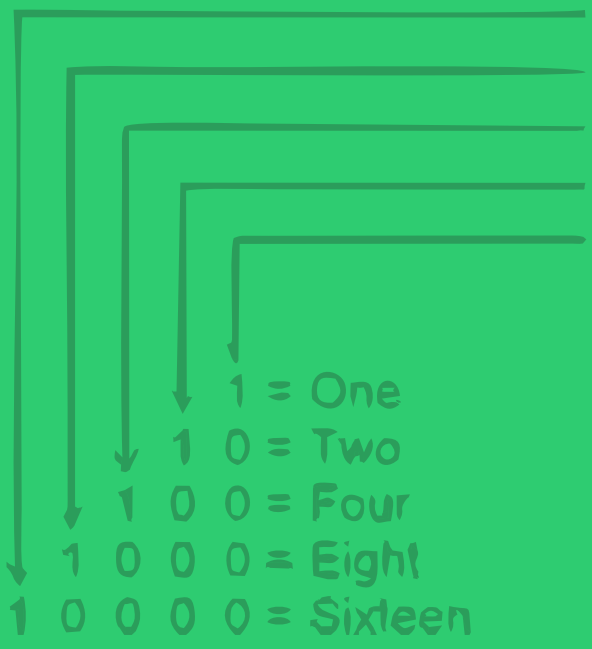


Photo by Gerd Altmann from Pixabay

UNIT 1

Number Base Arithmetic and Types

Introduction

I welcome you to the second module of CSC 111 – Introduction to Computer Science I. In the previous module, we have gone through the history, generations, classification, advantages and disadvantages of the computer system. In this module, we'll have an introduction to number bases and computer arithmetics.

Number base is a way of representing numbers in computing. There are different types of number base system which means numbers can be represented in any of the bases. This unit will expose you to the decimal number system, binary number system, Octal number system and hexadecimal system.

Learning Outcomes

At the end of this unit, you should be able to:

- Explain the concept of number base
- List the component digits of each type of number bases

Main Content

Number Base Arithmetic

05 mins

The number system is a writing system for representing numbers of a given set, consistently using digits or other symbols. The Numbers in the decimal system are represented by means of positional notation. That is, the value or weight of a digit depends on its location within the number. A number N, when expressed in positional notation in the base b, is written as:

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-m}$$

and defined as

$$a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots + a_{-m} b^{-m}$$
$$= \sum_{i=-m}^{n} a_i b^i \text{---------- (2.1)}$$

The "a" in the above equation are called digits and may have one of "b" possible values. Positional notation employs the radix point to separate the integer and fractional parts of the number. Decimal arithmetic uses the decimal point while binary arithmetic uses binary point.

Number Base Types

Now that you are familiar with the meaning of Number Base, I will go on to consider the four base types – decimal, binary, octal, and hexadecimal

Decimal: base, b = 10, digits, a = {0,1,2,3,4,5,6,7,8,9}
Binary: base, b = 2 digits, a ={0,1}
Octal: base, b = 8 digits, a = {0,1,2,3,4,5,6,7}
Hexadecimal: base, b = 16 digits, a = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

A **subscript** is used to indicate the base of a number when necessary, for example, 123_{10}, 456_{8}, 1011_{2}.

A **decimal number system** is the system we use in our day to day activities. It has the base 10 because it uses 10 digits (0-9). The successive positions to the left of the point represent units, tens, hundreds, thousands and so on. For example:

5372: 5 thousand, 3 is hundred, 7 is tens and 2 is unit. Each position represents a specific power of the base (10).

A **binary number system** is a number expressed in base 2. It uses only digits 0 and 1 to represent numbers. The binary system is the language the computer understands (known as machine language) and is used in modern computers and computer-based devices.

The **octal number system** has the base 8 because it uses 8 digits (0-7). Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of three (starting from the right).

The **hexadecimal number system** has the base 16 because it uses 16 digits (0-7, A-F). Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of four (starting from the right).

Activity 1: List the component digits of the following number bases

Octal Number	____|__|___|____|__|__|__|__
Binary Number	__|___
Hexadecimal	____|__|___|____|__|__|__|__|____|__|___|____|__|__|__|__
Decimal	__|___|____|__|__|__|__|__|___

Human beings normally work in decimal and computers in binary. The purpose of the octal and hexadecimal systems is as an aid to human memory since they are shorter than binary digits. Moreover, octal and hexadecimal numbers are more compact than binary numbers (one octal digit equals three binary digits, and one hexadecimal digit equals four binary digits), they are used in computer texts and core-dumps (a printout of part of the computer's memory). The advantage of binary number is in decoding electrical signals that switch on (logical one), or switch off (logical zero) a device.

Summary

In this unit, you have learnt that:

In this unit, you have learnt that the number base system is a way of representing numbers in different consistent forms. There are 4 major types of number base system, which are, decimal, binary, octal decimal, and hexadecimal number system.

Self-Assessment Questions

1. List the available number base systems
2. Differentiate between the digits used for each number base

Tutor Marked Assessment

- Differentiate between the four number base systems
- Why is decimal number base preferable for human beings and binary for computers?

Further Reading

- https://www.tutorialspoint.com/computer_fundamentals/computer_number_system.htm
- https://en.wikipedia.org/wiki/Binary_number

Photo by:
Gerd Altmann
from Pixabay

UNIT 2

Number Base Conversion

Introduction

Having learnt about the number base system, it is important that you also learn how numbers can be converted from one base to the other. In this unit, you will learn about how numbers can be converted from base 10 to other number bases such as base 2, base 8 and base 10, respectively.

Learning Outcomes

At the end of this unit, you should be able to:

- Convert from binary to decimal, to octal decimal, and hexadecimal
- Convert from decimal, and hexadecimal
- Convert from Octal decimal to decimal, to binary, and hexadecimal
- Convert from hexadecimal to binary, to decimal and octal decimal

Main Content

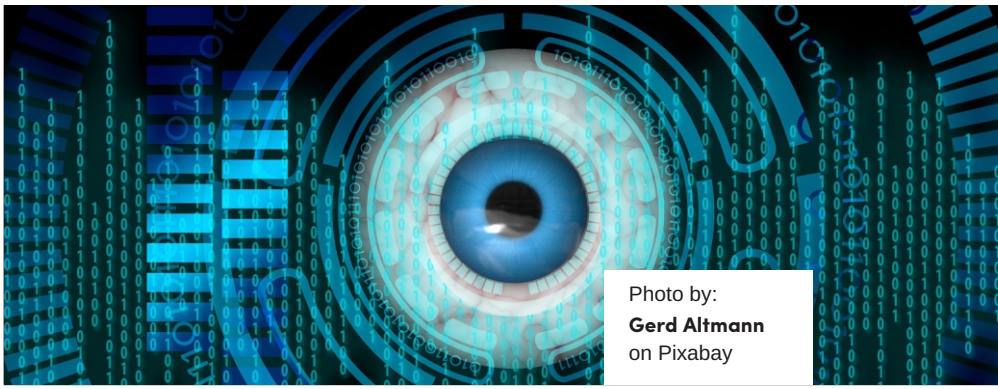


Photo by: **Gerd Altmann** on Pixabay

Conversion of Integers from Decimal to Binary

To convert a decimal integer to binary, divide the number successively by 2, and after each division record the remainder which is either 1 or 0. The process is terminated only when the result of the division is 0 remainder 1. The result is read from the most significant bit (the last remainder) upwards.

For example: to convert 123_{10} to binary, we have

Numerator	Divisor	Quotient	Remainder
123	2	61	1
61	2	30	1
30	2	15	0
15	2	7	1
7	2	3	1
3	2	1	1
1	2	0	1

Consequently, $123_{10} = 1111011_2$

Conversion of Integers from Decimal to Octal

Just as in binary numbers above except that the divisor is 8. The process of conversion ends when the final result is 0 remainder R (where $0 \leq R < 8$).

For example: to convert 4629_{10} to octal, we have

Numerator	Divisor	Quotient	Remainder
4629	8	578	5
578	8	72	2
72	8	9	0
9	8	1	1
1	8	0	1

Consequently, $4629_{10} = 11025_8$

Conversion of Integers from Decimal to Hexadecimal

Just as in binary and octal, the divisor is 16. The remainder lies in the decimal range 0 to 15, corresponding to the hexadecimal range 0 to F.

For example: to convert 53241_{10} to hexadecimal, we have

Numerator	Divisor	Quotient	Remainder
53241	16	3327	9
3327	16	207	15 = F
207	16	12	15 = F
12	16	0	12 = C

Therefore, $53241_{10} = CFF9_{16}$

Conversion of Integers from Other Bases to Decimal

Just express in the positional notation earlier stated above. But take note of the base.

For example, to convert 1010111_2, 6437_8, and $1AC16$ to decimal

i. $1010111_2 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
$= 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$
$= 64 + 0 + 16 + 0 + 4 + 2 + 1$
$= 87_{10}$

ii. $6437_8 = 6 \times 8^3 + 4 \times 8^2 + 3 \times 8^1 + 7 \times 8^0$
$= 6 \times 512 + 4 \times 64 + 3 \times 8 + 7 \times 1$

i. $1010111_2 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
$= 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$
$= 64 + 0 + 16 + 0 + 4 + 2 + 1$
$= 87_{10}$

ii. $6437_8 = 6 \times 8^3 + 4 \times 8^2 + 3 \times 8^1 + 7 \times 8^0$
$= 6 \times 512 + 4 \times 64 + 3 \times 8 + 7 \times 1$
$= 3072 + 256 + 24 + 7$
$= 3359_{10}$

iii. $1AC_{16} = 1 \times 16^2 + 10 \times 16^1 + 12 \times 16^0$
$= 1 \times 256 + 10 \times 16 + 12 \times 1$
$= 256 + 160 + 12$
$= 428_{10}$

Conversion from Binary Integer to Octal

Form the bits into groups of three starting at the binary point and moving leftwards. Replace each group of three bits with the corresponding octal digits (0 to 7).

For example, to convert 11001011101_2 to Octal

$11001011101_2 = 11 \quad 001 \quad 011 \quad 101$

$11 = 1 \times 2^1 + 1 \times 2^0 = 3; \quad 001 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1;$
$011 = 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3; \; 101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

Therefore $11001011101_2 = 3135_8$

Conversion from Binary Integer to Hexadecimal

The binary number is formed into groups of four bits starting at the binary point. A hexadecimal digit replaces each group from 0 to 9, A, B, C, D, E, F.

For example, to convert 11001011101_2 to hexadecimal

$11001011101_2 = 110 \quad 0101 \quad 1101$
$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6;$
$0101 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5;$
$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13 = D$

Conversion from Octal to Binary

Converting an octal number into its binary equivalent requires the reverse procedure of converting from binary to octal. Its binary equivalent simply replaces each octal digit.

For example, to convert 41357_8 to binary

413578 : converting each digit into binary, we have

4 = 100 1 = 001 3 = 011 5 = 101 7 = 111

Replacing each octal digit by its binary equivalent:

$41357_8 = 100\ 001\ 011\ 101\ 111 = 100001011101111_2$

Conversion from Hexadecimal to Binary

Each hexadecimal digit is replaced by its 4-bit binary equivalent. For example, to convert AB4C16 to binary

AB4C16: A = 10 =1010 B = 11 = 1011 4 = 0100 C = 12 = 1100

$AB4C16 = 1010\ 1011\ 0100\ 1100 = 1010101101001100_2$

Conversion from Hexadecimal to Octal

Conversion between hexadecimal and octal values is best performed via binary. For example, to convert 12BC16 to octal

12BC16 = 1 0010 1011 1100

Regrouping into three bits from the right-hand side

12BC16 = 1 001 010 111 100

Converting each group into octal digit

$12BC16 = 11274_8$

Conversion from Octal to Hexadecimal

For example, to convert 41357_8 to hexadecimal

41357_8 = 4 =100 1 = 001 3 = 011 5 = 101 7 = 111

Regrouping into four bits

$41357_8 = 100\ 0010\ 1110\ 1111$

= 4 2 14 15

= 42EF16

Conversion of Binary Fractions to Decimal

Treat all fractions as integers scaled by an appropriate factor. For example, to convert 0.01101_2 to decimal.

By expressing in standard form, we have:

$$0.01101_2 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$$

$$= 0 \times \frac{1}{2} + 1 \times 1/2^2 + 1 \times 1/2^3 + 0 \times 1/2^4 + 1 \times 1/2^5$$

$$= 0 \times \frac{1}{2} + 1 \times \frac{1}{4} + 1 \times \frac{1}{8} + 0 \times \frac{1}{16} + 1 \times \frac{1}{32}$$

$$= 0 + \frac{1}{4} + \frac{1}{8} + 0 + \frac{1}{32} = \frac{1}{4} + \frac{1}{8} + \frac{1}{32} = \frac{8+4+1}{32} = \frac{13}{32} = 0.40625$$

$$\therefore \quad 0.01101_2 = 0.40625_{10}$$

Conversion of Decimal Fractions to Binary

To convert a decimal fraction to a binary fraction, the decimal fraction is multiplied by two (2), and the integer part noted. The integer, which is either 1 or 0, is then stripped from the number to leave a fractional part. The new fraction is multiplied by two (2), and the integer part noted. The process is carried out repeatedly until it ends or a sufficient degree of precision has been achieved. The binary fraction is formed by reading the integer parts from top to bottom.

For example, to convert 0.687510 to binary

$$0.6875 \times 2 = 1.3750$$
$$0.3750 \times 2 = 0.7500$$
$$0.7500 \times 2 = 1.5000$$
$$0.5000 \times 2 = 1.0000$$

$\therefore 0.687510 = 0.1011_2$

We can convert from decimal fractions to octal or hexadecimal fractions by using the same algorithms used for binary conversions. We only need to change the base (that is: 2, 8, 16).

Conversion of Binary Fractions to Octal/Hexadecimal

First, split the binary digits into groups of three (four for hexadecimal), start grouping bits at the binary point and move to the right. Any group of digits remaining on the right containing fewer than three (four for hexadecimal) bit must be made up to three (four for hexadecimal) bit by the addition of zeros to the right of the least significant bit.

For example, to convert 0.10101100_2 and 0.10101111_2 to octal

$0.10101100_2 = 0.\quad 101\,011\,00(0)2 = 0.530_8$

$0.10101111_2 = 0.\quad 101\,011\,11(0)2 = 0.536_8$

To convert to hexadecimal

$0.10101100_2 = 0.\quad 1010\,1100 = 0.AC16$

$0.101011001_2 = 0.\quad 1010\,1100\,1(000) = 0.AC816$

Conversion of Octal/Hexadecimal Fractions to Binary

$0.4568 = 0.\,100\,101\,110 \quad = 0.100101110_2$

$0.ABC16 = 0.\quad 1010\,1011\,1100 = 0.101010111100_2$

Summary

In this unit, you have learnt the conversion of numbers from one base to another, either an integer or a fraction. You definitely should be able to:

Self Assessment

Convert the following:

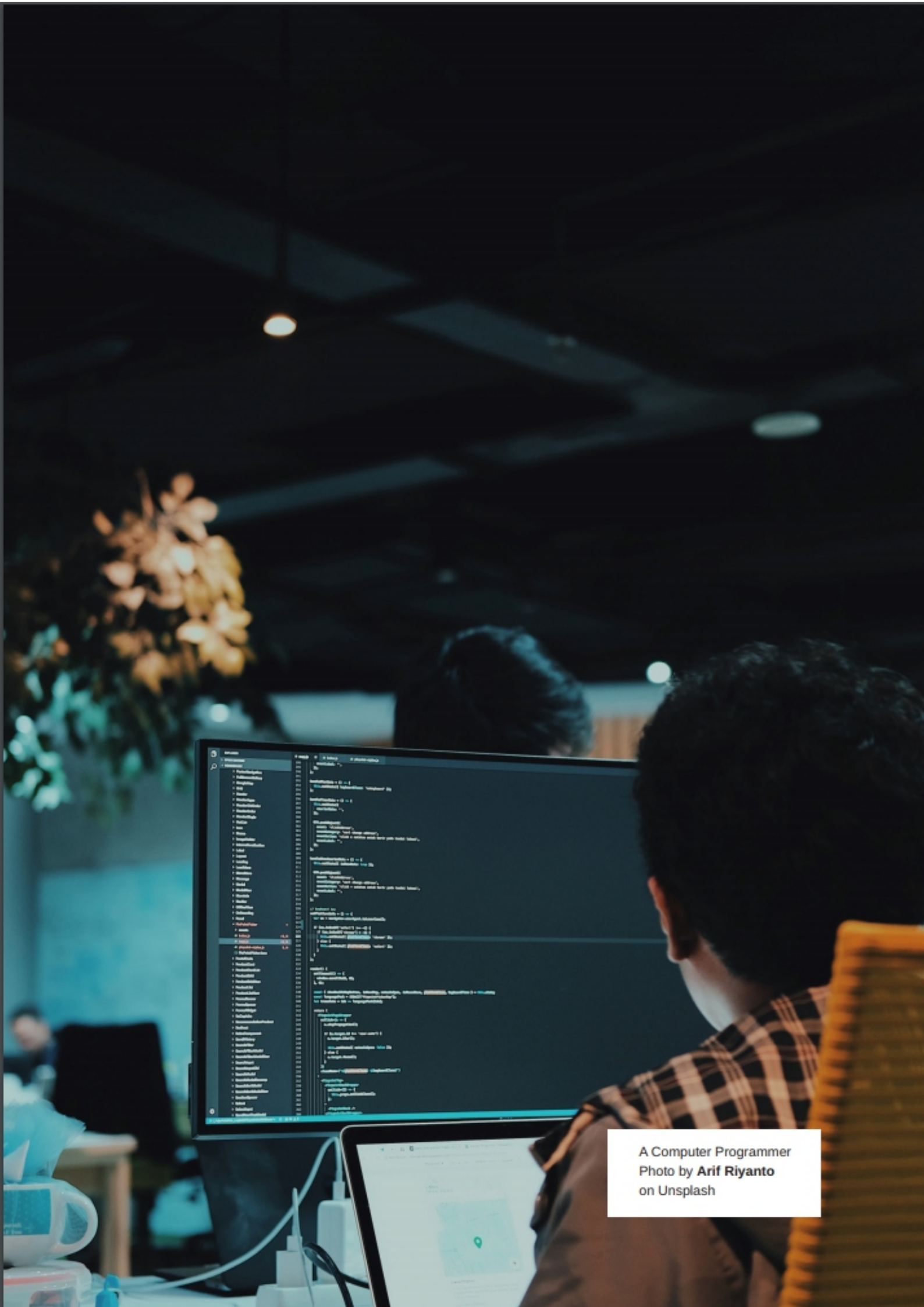
1. $11001100_2 = ?10, ?_8, ?16$
2. $4567810 = ?_2, ?10, ?_8, ?16$

Tutor Marked Assignment

- Convert the following:
  553.355_{10} = ?2, $?_8$ and ?16
  Ao716 = ?10, $?_8$, and ?2

Further Reading

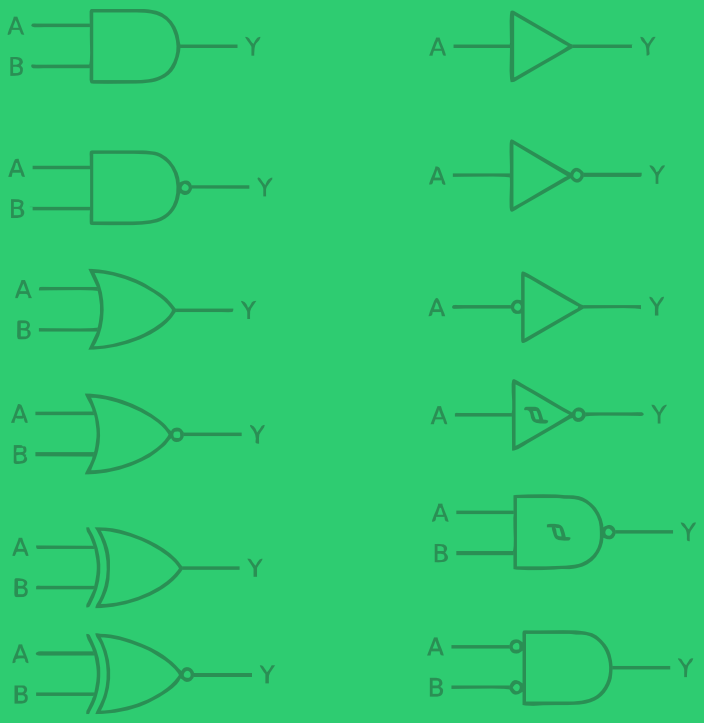
- https://code.tutsplus.com/articles/number-systems-an-introduction-to-binary-hexadecimal-and-more--active-10848
- https://www.talentsprint.com/blog/2018/01/number-system-i-what-is-number-syste.html
- https://www.varsitytutors.com/hotmath/hotmath_help/topics/number-systems

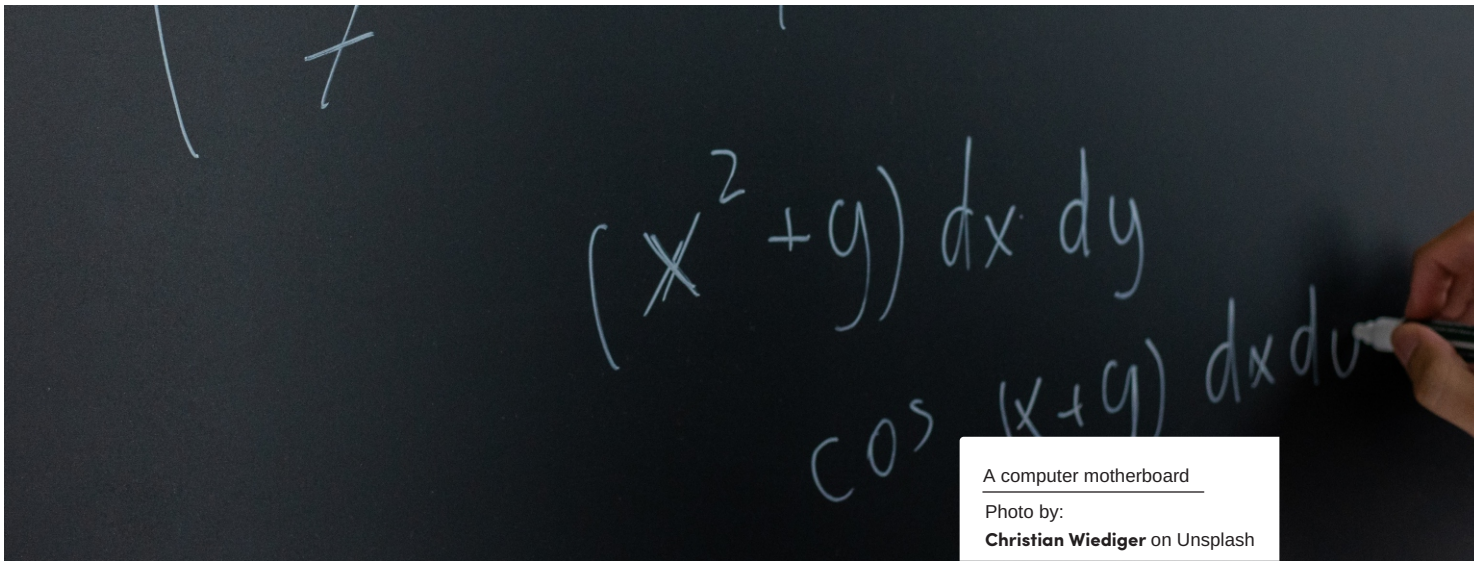


A Computer Programmer
Photo by **Arif Riyanto**
on Unsplash

Module 3

Boolean Algebra and Karnaugh Map





A computer motherboard

Photo by:
Christian Wiediger on Unsplash

UNIT 1

Boolean Algebra, Fundamentals of Truthtables and Precedence

Introduction

You are welcome to the third module of CSC 111- Introduction to Computer I. I hope you have learnt much about Computers and their operations. In this Module, you will learn more about Computer Arithmetic and Logical operations. This unit covers Boolean Algebra and its operators as well as truthtables. I will explain the use of Boolean operators such as AND operator represented with a dot (.), the OR operator represented with plus (+) and the NOT operator, which is an inverter. I will also give you information on how to represent the Boolean Operation through the use of truth tables.

Learning Outcomes

At the end of this unit, you should be able to:

- explain the concept of Algebra
- differentiate between among the Boolean operators
- reduce Boolean functions

Main Content

picture: A Computer Desktop

Source: Unsplash.com

Algebra

(2Min)

The word Algebra rings a bell, right? Algebra means reunion of broken parts. It is the study of mathematical symbols and rules for manipulating the symbols. We can also regard Algebra as elementary, abstract or modern depending on the level or field of study.

I want you to learn that algebra has computations similar to arithmetic but with letters standing for numbers which allow proofs of properties that are true regardless of the numbers involved. For example, quadratic equation: $ax2 + bx + c = 0$ where a, b, c can be any number ($a \neq 0$). Algebra is used in many studies, for example, elementary algebra, linear algebra, Boolean algebra, and so on.

Activity 1: Do you know that the word Algebra comes from an Arabic word al-jabr? Read more on the history of Algebra here...https://en.wikipedia.org/wiki/History_of_algebra

What about Polynomials?

A polynomial involves operations of addition, subtraction, multiplication, and non-negative integer exponents of terms consisting of variables and coefficients. For example, $x2 + 2x - 3$ is a polynomial in the single variable x. Polynomial can be rewritten using commutative, associative and distributive laws.

An important part of algebra is the factorization of polynomials by expressing a given polynomial as a product of other polynomials that cannot be factored any further. Another important part of algebra is the computation of polynomial greatest common divisors. $x2 + 2x - 3$ can be factored as $(x - 1)(x + 3)$.

Boolean Algebra

4 Mins

I want you to know that Boolean algebra is the branch of algebra in which the values of the variables are true values denoted by 1 and 0 or true and false respectively.

What you should also know is that Boolean algebra can be used to describe the logic circuit; it is also used to reduce the complexity of digital circuits by simplifying the logic circuits. Another name we can call Boolean algebra is Boolean logic. It may interest you to know that Boolean algebra was developed by George Boole sometime in the 1840s and is greatly used in computations and computer operations. As you would have guessed, the name Boolean comes from the name of the author.

Boolean algebra is a logical calculus of truth values. It somewhat resembles the arithmetic algebra of real numbers but with a difference in its operators and operations. Boolean operations involve the set {0, 1}, that is, the numbers 0 and 1. Zero [0] represents "false", or "off" and One [1] represents "true" or "on".

1 – True, on

0 – False, off

This has proved useful in programming computer devices, in the selection of actions based on conditions set.

Basic Boolean operations

1. AND

   The AND operator is represented by a period or dot in-between the two operands e.g

   - X.Y

   The Boolean multiplication operator is known as the AND function in the logic domain; the function evaluates to 1 only if both the independent variables have the value 1.

2. OR

   An addition sign represents the OR operator. Here the operation + is different from that defined in normal arithmetic algebra of numbers. E.g. X+Y

   The + operator is known as the OR function in the logic domain; the function has a value of 1 if either or both of the independent variables has the value of 1.

3. NOT

The NOT operator is represented by X' or $\bar{X}$.

This operator negates whatever value is contained in or assigned to X. It changes its value to the opposite value. For instance, if the value contained in X is 1, X' gives 0 as the result, and if the value stored in X is 0, X' gives 1 as the result. In some texts, NOT may be represented as $\bar{X}$

In order for you to better understand these operations, the **truth table** is presented for the result of any of the operations on any two variables.

Truth Tables

A truth table is a mathematical table used in logic to compute the functional values of logical expressions on each of their functional arguments. It is specifically in connection with Boolean algebra and Boolean functions. Truth tables can be used to tell if a proposition expression is logically valid. In a truth table, the output is completely dependent on the input. It is composed of a column for each input entry and another column the corresponding output. Each row of the truth table, therefore, contains one possible configuration of the input variables (for instance, X=true Y=false), and the result of the operation for those values.

Applications of Truth table

1. The truth table can be used in analysing arguments.
2. It is used to reduce basic Boolean operations in computing
3. It is used to test the validity of statements. In validating statements, the following three steps can be followed:
   a. Represent each premise (represented as inputs) with a symbol (a variable).
   b. Represent the conclusion (represented as the final result) with a symbol (a variable).
   c. Draw a truth table with columns for each premise (input) and a column for the conclusion (result).

Truth tables are a means of representing the results of a logic function using a table. They are constructed by defining all possible combinations of the inputs to a function in the Boolean algebra and then calculating the output for each combination in turn. The basic truth table shows the various operators and the result of their operations involving two variables only. More complex truth tables

can be built from the knowledge of the foundational truth table. The number of input combinations in a Boolean function is determined by the number of variables in the function, and this is computed using the formula 2^n

Number of input combinations $= 2^n$. Where n is several variables (s).

For example, a function with two variables has an input combination of $2^2 = 4$. Another with three variables has $2^3 = 8$ input combinations and so on.

AND

X	Y	X . Y
0	0	0
0	1	0
1	0	0
1	1	0

OR

X	Y	X + Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

X	Y
0	1
1	0

The NOT operation is a unary operator; it accepts only one input.

Example:

Draw a truth table for $A+BC$

A	B	C	BC	A+BC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Draw a truth table for $AB+BC$

A	B	C	AB	BC	AB+BC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	0	0	1
1	1	1	1	1	1

Draw a truth table for *A(B+D)*				
A	B	D	B+D	A(B+D)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

J= f(A,B,C) = $A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$A\bar{B}\bar{C}$	$\bar{A}\bar{B}\bar{C}$	$A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	0	1
1	0	1	0	1	0	0	0	0
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0

Summary

In this unit, you have learnt :

the operators of Boolean algebra, which are the AND operator represented with a dot (.), the OR operator represented with plus (+) and the NOT operator which is an inverter. The unit also shows how Boolean operators and variables can be represented using truth tables. The truth table can be applied to check the validity of a statement, express arguments as well as reducing complex boolean operators.

Self-Assessment Questions

1. Explain the concept of Algebra
2. Differentiate between among the Boolean operators
3. Reduce boolean functions
4. Solve the following Boolean functions

   a. J= f(A,B,C) = $\bar{A}B + B\bar{C} + BC + A\bar{B}\bar{C}$
   b. Z= f(A,B,C) = $\bar{A}B + B\bar{C} + BC + A\bar{B}\bar{C}$

Tutor Marked Assessment

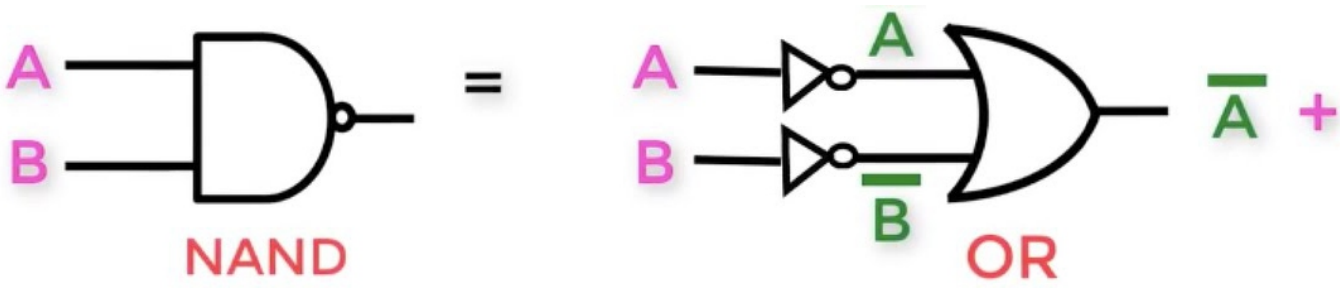
- Draw a truth table for (A+B)(A+C).
- Draw a truth table for W(X+Y)Z.

Further Reading

- https://en.wikipedia.org/wiki/Algebra
- https://en.wikipedia.org/wiki/Polynomial#Etymology
- http://www.mee.tcd.ie/~bfoley/2e6/digital/L3-2E6-Digital.docx

7.0 References

- Gupta A, Arora S. Industrial automation and robotics. Laxmi Publications; 2009.
- https://books.google.com.ng/books?hl=en&lr=&id=Y7rgCP7iC18C&oi=fnd&pg=PA1&dq=Industrial+Automation+and+Robotics+a.+K+Gupta+s.K.+Arora&ots=e4KP0Fl_g9&sig=5FeHKe3utUmUlfjaTLFQf-RbkMY&redir_esc=y#v=onepage&q=Industrial%20Automation%20and%20Robotics%20a.%20K%20Gupta%20s.K.%20Arora&f=false
- Wassell I. J. Digital Electronics: Part I – Combinational and Sequential
- Logichttps://www.cl.cam.ac.uk › teaching › DigElec ›
- Digital_Electronics_08_pdf



De Morgan's theorem

Boolean Alaebra

UNIT 2

DeMorgan's Theorem and Precedence

Introduction

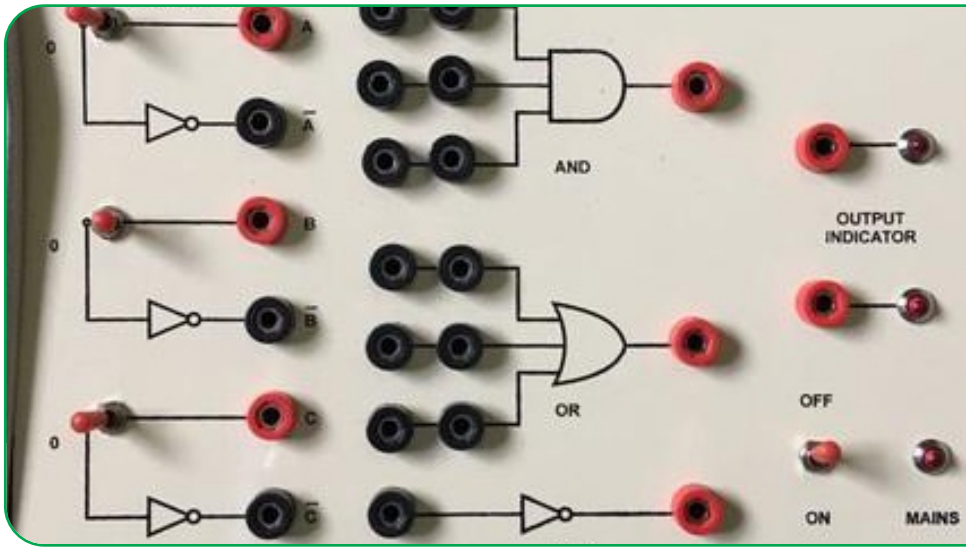
In the last unit, I attempted to explain Boolean Algebra. In this unit, I will also explain the use of DeMorgan's Theorem in solving different Boolean algebra expressions. The DeMorgan Theorem shows the relationship between the product and sums any two numbers. The theorem will help you to simplify Boolean Algebra

Learning Outcomes

At the end of this unit, you should be able to:

- explain the axiomatic relationships using truth tables;
- state the order of precedence of a boolean expression; and
- list the fundamental importance of boolean algebra

Main Content



Boolean Definition

3 Mins

Based on Boolean definitions, the following axiomatic relationships hold:

for $a,b,c \in B$

A1	Closure	(a) $a+b \in B$	(b) $a \,.\, b \in B$
A2	Identity	(a) $a+0=a$	(b) $a \,.\, 1 = a$
A3	Commutation	(a) $a+b=b+a$	(b) $a.b=b.a$
A4	Distribution	(a)) $a(b+c)=(a.b)+(a.c)$	(b) $a+(b.c)=(a+b).(a+c)$
A5	Inverse	(a) $a + a' = 1$	(b) $a.a' = 0$

To check that the axioms conform to the definitions, Properties A1 is obvious

A	a+0
0	0+0=0
1	1+0=1

a	a.1
0	0.1=0
1	1.1=1

A3 may also be verified by truth table:

A	b	a+b	b+a	a.b	b.a
0	0	0+0=0	0+0=0	0.0=0	0.0=0
0	1	0+1=1	1+0=1	0.1=0	1.0=0
1	0	1+0=1	0+1=1	1.0=0	0.1=0
1	1	1+1=1	1+1=1	1.1=1	1.1=1

We next consider A4(a)

A	b	C	b+c	a.(b+c)	a.b	a.c	(a.b)+(a.c)
0	0	0	0	0	0	0	**0**
0	0	1	1	0	0	0	**0**
0	1	0	1	0	0	0	**0**
0	1	1	1	0	0	0	**0**
1	0	0	0	0	0	0	**0**
1	0	1	1	1	0	1	**1**
1	1	0	1	1	1	0	**1**
1	1	1	1	1	1	1	**1**

We next consider A4(a)

A	b	C	bc	a+(bc)	a+b	a+c	(a+b)(a+c)
0	0	0	0	**0**	0	0	**0**
0	0	1	0	**0**	0	1	**0**
0	1	0	0	**0**	1	0	**0**
0	1	1	1	**1**	1	1	**1**
1	0	0	0	**1**	1	1	**1**
1	0	1	0	**1**	1	1	**1**
1	1	0	0	**1**	1	1	**1**
1	1	1	1	**1**	1	1	**1**

Finally, the A5 relations are verified from

A	a′	a+a′	a.a′
I 0	1	1	**0**
1	0	1	**0**

I want you to bear in mind that an important feature of Boolean algebra is duality. The set of (b) axioms are said to be duals of the (a) axioms, and vice versa, in that a (b) axiom can be formed from its (a) counterpart by exchanging operators and identity elements '+' to '.' and '1' to '0'. Thus for every theorem derived from one particular set of axioms, one can construct a dual theorem based on the corresponding set of dual axioms.

Any more complex functionality can be constructed from the three basic Boolean operators (And, Or, and Not) by using DeMorgan's Law:

I. The complement of a product is equal to the sum of complements

II. The complement of the sum is equal to the product of the complement

$$\overline{A + B + \ldots + N} = \overline{A}\,\overline{B}\,\overline{C} \ldots \overline{N}$$

Precedence

Order of precedence also exists in Boolean algebra as it exists in other areas of mathematics. This order should be followed in Boolean computations. The Boolean operators defined above have order precedence as defined here:

NOT operations have the highest precedence, followed by AND operations, followed by OR operations.

Also, brackets can be used. Example : X.(Y + Z), X+(Y+Z), X.Y+(X+Y)

X.Y + Z and X.(Y + Z) are not the same functions.

The brackets should be evaluated first to reduce the complexity of the Boolean operation.

Boolean operations are foundational tools used in building computers and electronic devices.

Consider this practical application of a Boolean operation:

"I will take an umbrella with me if it is raining or the weather forecast is bad".

This statement functionally has two propositions which are:

a. It is raining; and

b. The weather forecast is bad.

Let "It is raining" be variable X , "The weather forecast is bad" be Y, and the result (taking an umbrella) be Z.

We can generate truth values in a truth table from this problem statement.

From the statement, if either of the conditions is true, an umbrella would be taken.

In functional terms, we can be considered the truth value of the umbrella proposition as the output or result of the truth values of the other two.

X (Raining)	Y (Bad weather)	Z(Take umbrella)
False	False	false
False	True	true
True	False	true
True	True	true

Another practical application of Boolean logic is this:

"I will sweep the class only if the windows are opened, and the class is empty".

From this statement, we can get two propositions which are "Windows opened" and "Class empty". These two propositions are the variables X and Y, respectively.

"Windows opened" – X

"Class empty" – Y

X (Windows opened)	Y (Class empty)	Z (Sweep)
False	False	false
False	True	false
True	False	false
True	True	true

Fundamental Importance of Boolean Algebra

2 Mins

1. Boolean logic forms the basis for computation in modern binary computer systems.
2. They are used in the development of software, in selective control structures (if and if...else statements).
3. They are used in the building electronic circuits. For any Boolean function, you can design an electronic circuit and vice versa.
4. A computer's CPU is built up from various combinatorial circuits. A combinatorial circuit is a system containing basic Boolean operations (AND, OR, NOT), some inputs, and a set of outputs.

We consider the Boolean function in two variables A and B.

$Z=f(A,B)= \overline{A}B + A\overline{B}$

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}B$	$A\overline{B}$	$\overline{A}B+A\overline{B}$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

Reducing complex Boolean functions

Very complex Boolean functions may result, and this can be simplified in two ways:

1. The Boolean algebra method
2. Karnaugh map (discussed in the next unit)

Boolean algebra method

The basic rules of Boolean algebra are logical in nature. These rules are followed in simplifying any Boolean function. As stated in the axioms above, the rules are:

1. A+0=A
2. A+1=1
3. A.0=0
4. A.1=A
5. A+A=A
6. A+ =1
7. A.A=A
8. A. =0
9. $\bar{\bar{A}}$ =A (double bar)
10. A+AB= A(1+B)=A(1)=A
11. A (B + C) = A B + A C
12. A + (B C) = (A + B) (A + C)

De-Morgan's theorem

De-Morgan's Theorem states that the complement of the product of two or more variables is equal to the sum of the complements of the variables.

$\overline{(A+B)} = \bar{A}\,\bar{B}$

$\overline{(A\,B)} = \bar{A} + \bar{B}$

For example:

1. Using the above laws, simplify the following expression:

Q = (A + B)(A + C)

AA + AC + AB + BC (law 11)
A + AC + AB + BC (law 7)
A(1 + C) + AB + BC (law 10)- (1 + C = 1)
A.1 + AB + BC
A(1 + B) + BC (1 + B = 1)
A.1 + BC (A.1 = A)
Q =A + BC

This implies that the expression: (A + B)(A + C) can be simplified to A + BC

2. Another example can be considered:

ØõŭJ e ũœÁ Ż^λ ²A^η +)(A+ C)

Z=AA+A C+A + C+A + C
Z=A+A C+A +A + C +0
Z=A(1+ C+ +)+ C
Z=A(1+ (C+1)+)+ C
Z=A(1+ +)+ C
Z=A(1)+ C
Z=A+ C

Summary

In this unit, you have learnt that:

DeMorgan's Theorem states that the complement of the product of two or more variables is equal to the sum of the complements of the variables. I also covered axiomatic relationship, truth table and its precedence.

Self-Assessment Questions

1.List and explain the axiomatic relationships using truth tables
2.State the order of precedence available in any boolean expression
3.List 3 fundamental importance of boolean algebra

Tutor Marked Assignment

- Using Boolean algebra techniques, simplify this expression:

$$a.\bar{b} + a.(\overline{b+c}) + b.(\overline{b+c})$$

(a.b.(c+b.d) + a.b).c.d

References

Egbewole, W. and Jimoh R. (Eds.). (2017) Digital Skill Acquisition. Ilorin, Nigeria: Unilorin

Further Reading

- http://oer.nios.ac.in/wiki/index.php/characteristics_of_computers
- http://ecomputernotes.com/fundamental/introduction-to-computer/what-are-characteristic-of-a-computer
- http://www.byte-notes.com/advantages-and-disadvantages-computers
- https://www.computerhope.com/issues/ch001798.htm

A computer motherboard

Photo by:
Christian Wiediger on Unsplash

UNIT 3

Karnaugh Map and Minimization of Expressions

Introduction

You learnt about DeMorgan's Theorem in the last unit. Now, I will discuss the Karnaugh map popularly called k-map. Karnaugh map is a diagram that has a rectangular array of squares, each representing a different combination of the variables of a Boolean function. It is a way of reducing boolean expressions' complexity. This unit covers the karnaugh map with description of how grouping can be done, labelling, and boolean function reduction.

Learning Outcomes

At the end of this unit, you should be able to:

- define k-map;
- list the k-map grouping rules; and
- to reduce expressions

Main Content

Karnaugh Map

01 Min

I want you to bear in mind that Boolean expression can be reduced to its simplest form through some steps involved in Karnaugh mapping. What then is a Karnaugh map? let us look at that. A Karnaugh map is a graphical method of Boolean logic expression reduction.

At this point, you can apply the theorems and laws of Boolean algebra to simplify logic expressions to produce simpler Boolean functions. It is important for you to note that simplifying a logic expression using Boolean algebra, though not terribly complicated, is not always the most straightforward process. There isn't always a clear starting point for applying the various theorems and laws, nor is there a definitive end in the process. The Karnaugh map is also known as Veitch diagram (K-map for short). It is a tool to facilitate the simplification of Boolean algebra integrated circuit expressions. The Karnaugh map reduces the need for extensive calculations by taking advantage of human pattern-recognition.

It may interest to know that the Karnaugh map was originally invented in 1952 by Edward W.

It was further developed in 1953 by Maurice Karnaugh, a physicist at Bell Labs, to help simplify digital electronic circuits. In a Karnaugh map, the Boolean variables are transferred (generally from a truth table) and ordered according to the principles of Gray code in which only one variable changes in between squares. Once the table is generated, and the output possibilities are transcribed, the data is arranged into the largest even group possible, and the minterm is generated. The K-map is a more straight-forward process of reduction. In the reduction process using a k-map, 0 represents the complement of the variable (e.g. B), and 1 represents the variable itself (e.g. B).

Karnaugh Map

03 Mins

I want you note very importantly that the truth table for the Boolean function

should be drawn with the result or output and the karnaugh map drawn with the number of boxes being equal to the number of outputs =number of input combinations=$2n$

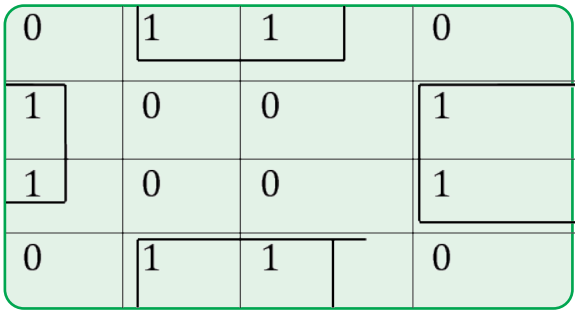
Where n is the number of variables in the function.

In grouping in the K-map, we get the simplified sum-of-products logic expression.

The rules you have to follow are:

1. Consider boxes with ones only. Boxes containing zeros would not be considered.
2. Group 1s in powers of 2. That is 2, 4, 8... ones.
3. Grouping can only be done side to side or top to bottom, not diagonally.
4. Using the same one in more than one group is permissible.
5. The target is to find the fewest number of groups.

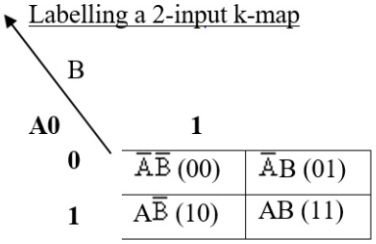
The top row may wrap around to the bottom row to form a group



Labelling a K-map

In labelling the Karnaugh map, we make use of the principle of the "gray code".

Labelling a 2-input k-map



For the 2-input k-map, the values change from 0 to 1 along both axes.

Labelling a 3-input K-map

AB	C0	1
00	$\bar{A}\bar{B}\bar{C}$(000)	$\bar{A}\bar{B}C$ (001)
01	$\bar{A}B\bar{C}$(010)	$\bar{A}BC$ (011)
11	$AB\bar{C}$(110)	ABC(111)
10	$A\bar{B}\bar{C}$(100)	$A\bar{B}C$(101)

In the case of the 3-input k-map, we have A and B on one side of the map and C on the other side of the map. Using gray code, we start with $\bar{A}\bar{B}$ (00), keeping A constant and changing B, we have $\bar{A}B$ (01). Now, if we still keep A constant and change B, we will have $\bar{A}\bar{B}$ (00) which already exists in the map, so, the next thing to do is to keep B constant and then change A. With this, we will have AB (11) next and then, $A\bar{B}$ (10).

For minimization using the k-map, the value 0 in the truth table, corresponding to a variable is taken as its complement. For instance, if the variable A has the value 0 in the truth table, it is taken as $\bar{A}$ to fill in the k-map.

It is important to note that in k-map, grey code and in general BCD, only one-bit change at a time. For example, 0000 has 4 bit only 1 bit can change at a time.

C \ AB	00	01	11	10
0	m0	m2	m6	m4
1	m1	m3	m7	m5

B \ AC	00	01	11	10
0	m0	m1	m5	m4
1	m2	m3	m7	m6

AB \ C	0	1
00	m0	m1
01	m2	m3
11	m6	m7
10	m4	m5

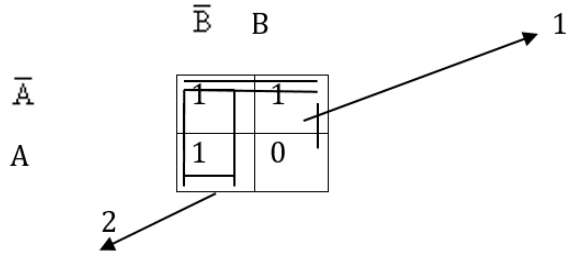
BC \ A	0	1
00	m0	m4
01	m1	m5
11	m3	m7
10	m2	m6

AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Consider the k-map:

0	1
1	0

The diagonal 1s cannot be grouped together.



There are two possible groupings here.

Example 1 Given $Z = AB + A\overline{B}$

A	B	$\overline{A}$	$\overline{B}$	AB	$A\overline{B}$	Z= $AB+A\overline{B}$
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	1	1
1	1	0	0	1	0	1

Put the output Z into a k-map:

	$\overline{B}$	B
$\overline{A}$	0	0
A	1	1

In k-map, the variable that remains constant across the group is retained. Since the variable B varies in value (looking at column $\overline{B}$ and B, the variable changed) and A remains constant, the constant value across the group is $\overline{A}$. A is not used even though it is constant because the value is 0.

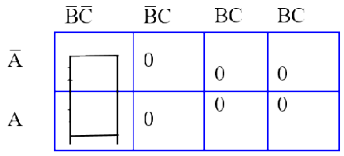
$Z = AB + A\overline{B}$

$Z = A$

Example 2

$J = f(A,B,C) = A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$

A	B	C	$\overline{A}$	$\overline{B}$	$\overline{C}$	$A\overline{B}\overline{C}$	$\overline{A}\overline{B}\overline{C}$	$A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	0	1
1	0	1	0	1	0	0	0	0
1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0



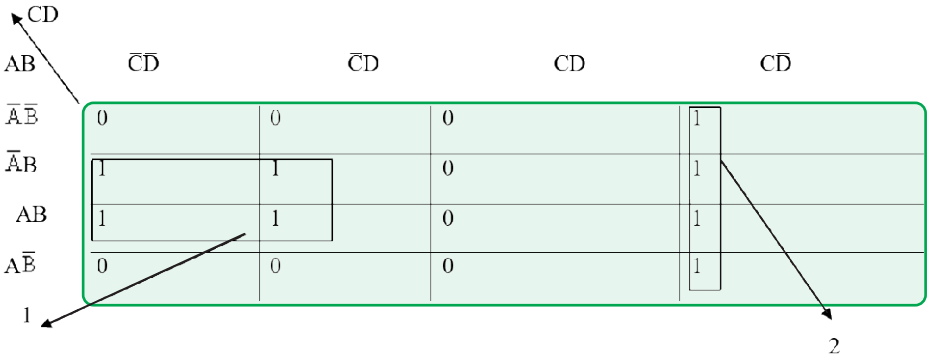
From the diagram, the value of A changes across the group and the value of $B\bar{C}$ remains the same.

$$J = A B\bar{C} + \bar{A} B\bar{C} = B\bar{C}$$

I want you to bear in mind the advantage of the k-map over the Boolean algebra method of function reduction is that the k-map has a definite process which is followed, unlike the boolean algebra method which may not have a particular starting and ending point.

Example 3

Consider the already filled k-map below,



The final answer here, after the grouping, is derived by looking across the group and eliminating the variable that changes in value.

For group 1, looking horizontally, D changes in value while C has a constant value of $\bar{C}$ So, D is eliminated and $\bar{C}$ retained. Looking vertically, A changes across the group while B remains constant. So, A is eliminated, and B retained.

For the group1, the answer is the AND of the retained variables after elimination, and this is $B\bar{C}$. We do the same for group 2. Our answer there is $C\bar{D}$ since vertically across the group, both and A and B change values.

After doing this for all the groups in the k-map, we then OR the individual results of each group.

So, for this k-map, we have $B\bar{C} + C\bar{D}$ as the minimum expression

Summary

In this unit, we have covered:

the boolean expression reduction using karnauph map; that is a diagram with a rectangular array of squares, each representing a different combination of the variables of a Boolean function.

Self-Assessment Questions

1.Reduce $J=f(A,B,C) = \bar{A}B + B\bar{C} + BC + AB\bar{C}$ using a k-map

2. Simplify the following Boolean functions:

$ABC + ABC + ABC + ABC + ABC$

$A\bar{B} + \bar{A}B + \bar{A}\bar{B}$

3.Consider a Boolean function represented by the truth table below and simplify the expression using k-map

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Tutor Marked Assessment

Consider the truth table given below and simplify the expression using k-map

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Further Reading

- https://www.allaboutcircuits.com/worksheets/boolean-algebra/
- https://studylib.net/doc/6803447/2.2.1.a-k
- https://www.allaboutcircuits.com/textbook/digital/chpt-8/karnaugh-maps-truth-tables-boolean-expressions/
- https://web.iit.edu/sites/web/files/departments/academic-affairs/academic-resource-center/pdfs/kmaps.pdf

References

- Gupta A, Arora S. Industrial automation and robotics. Laxmi Publications; 2009.
- https://books.google.com.ng/books?hl=en&lr=&id=Y7rgCP7iC18C&oi=fnd&pg=PA1&dq=Industrial+Automation+and+Robotics+a.+K+Gupta+s.K.+Arora&ots=e4KP0Fl_g9&sig=5FeHKe3utUmUlfjaTLFQf-RbkMY&redir_esc=y#v=onepage&q=Industrial%20Automation%20and%20Robotics%20a.%20K%20Gupta%20s.K.%20Arora&f=false
- Wassell I. J. Digital Electronics: Part I – Combinational and Sequential Logichttps://www.cl.cam.ac.uk › teaching › DigElec › Digital_Electronics_08_pdf

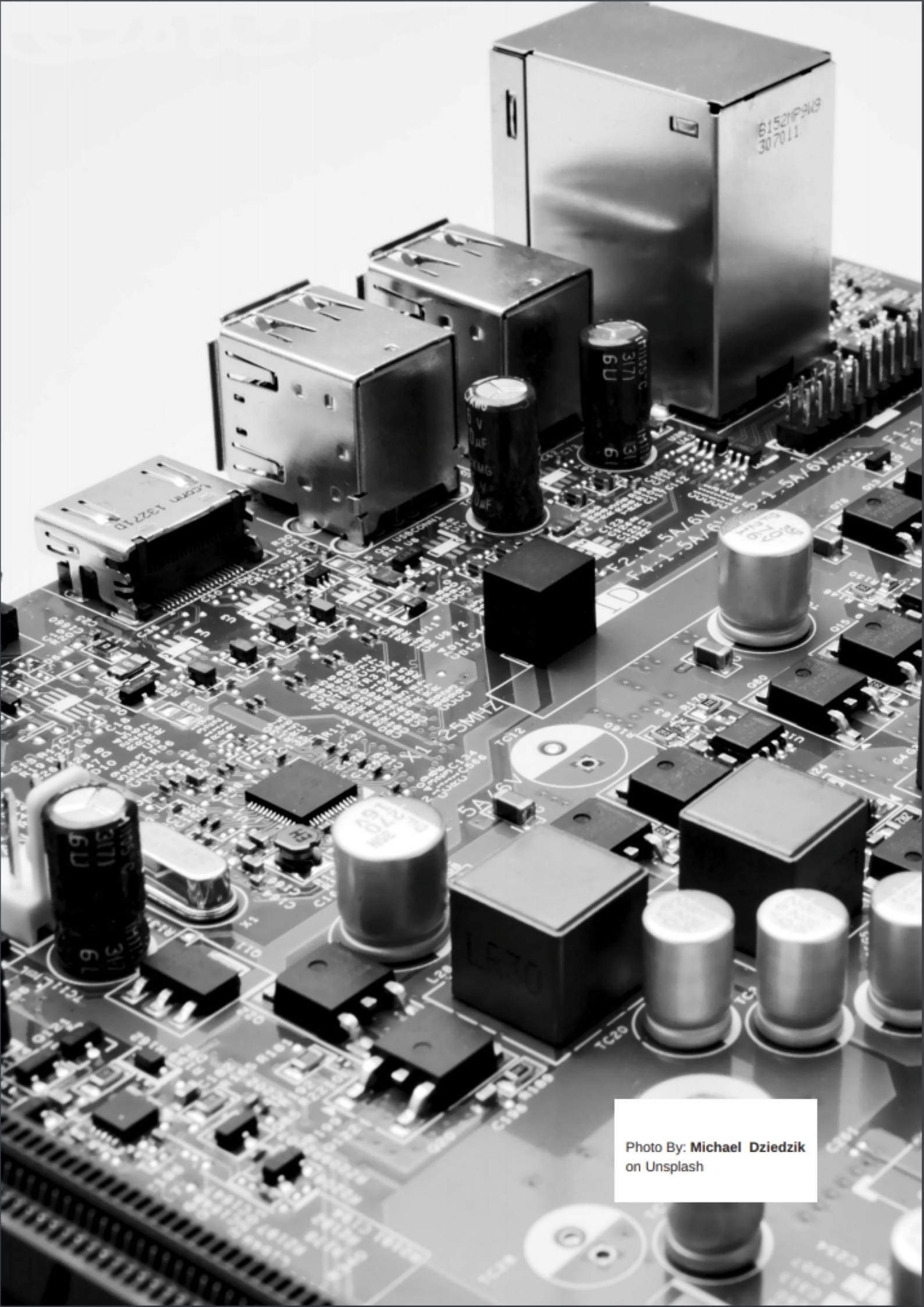
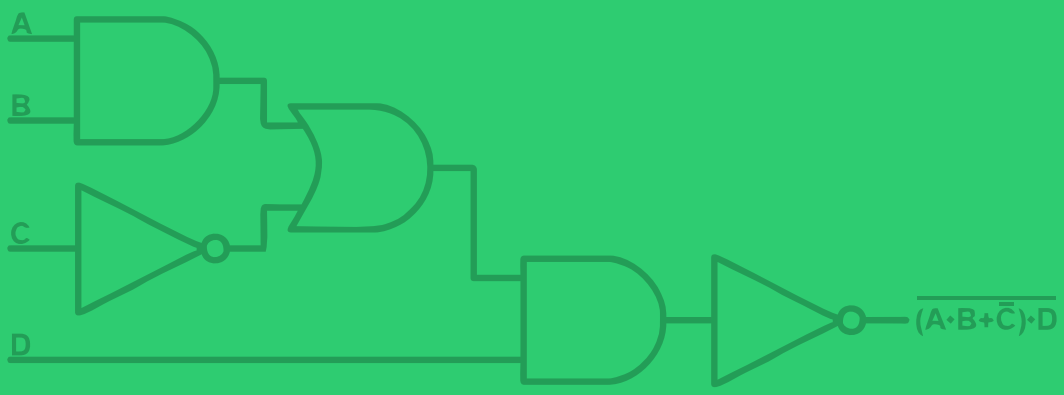
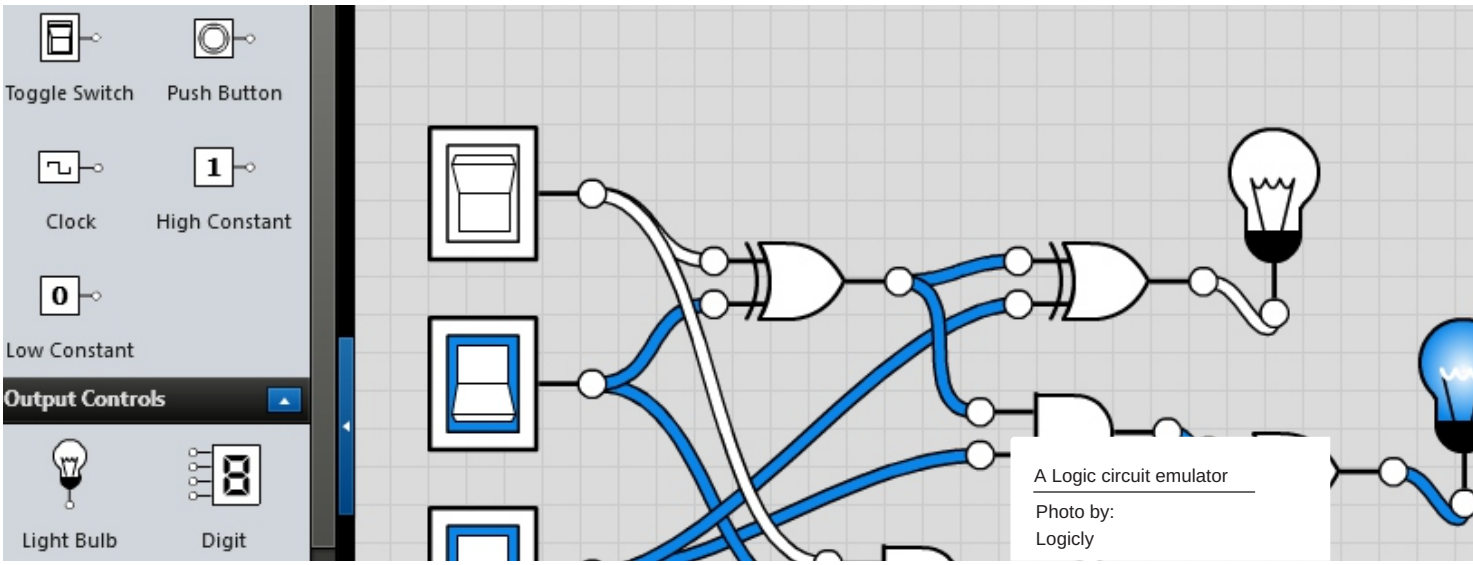


Photo By: **Michael Dziedzik** on Unsplash

Module 4

Logic Gates





A Logic circuit emulator

Photo by:
Logicly

UNIT 1

Basic Logic Gates

Introduction

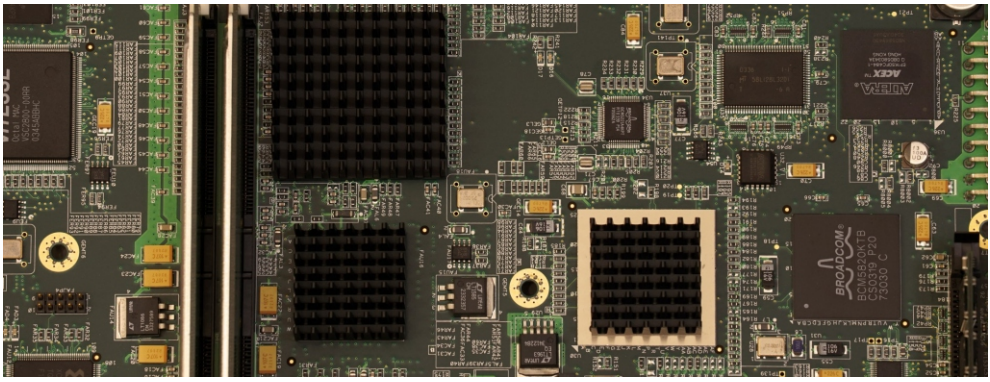
You are gradually coming to the end of your journey through CSC 111. In the previous modules, we have explained the Arithmetic and Boolean operations. In this Module, I will explain the basic logic gates and Combinatorial Logic Circuits. You need to pay close attention because Logic gates are building blocks of the digital circuit. Electronic circuits are built using various types of gate. The basic gates are AND, OR, and the NOT gate. Other advanced gated are developed using combinations of the basic gates like NAND, NOR. EXOR, AND EXNOR

Learning Outcomes

At the end of this unit, you should be able to:

- list at least five (5) types of gates;
- mention the logic function associated with each gate; and
- draw the truth table associated with each gate.

Main Content



Basic Logic Gates

03 Mins

We can view logic as black boxes with binary input (independent variable) and binary output (dependent variable). It also refers to both the study of modes of reasoning and the use of valid reasoning. In the latter case, logic is used in most intellectual activities. Logic in computer science has emerged as a discipline, and it has been extensively applied in the fields of Artificial Intelligence, and Computer Science and these fields provide a rich source of problems in formal and informal logic.

Boolean logic has been considered as a fundamental part to computer hardware, particularly, the system's arithmetic and logic structures, relating to operators AND, NOT, and OR.

Logic Gates

A logic gate is an elementary building block of a digital circuit. Complex electronic circuits are built using the basic logic gates. At any given moment, every terminal of the logic gate is in one of the two binary conditions low (0) or high (1), represented by different voltage levels.

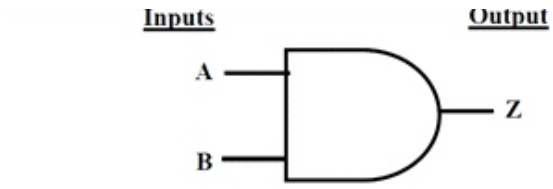
There are 3 basic logic gates: AND, OR, NOT.

Other gates- NAND, NOR, XOR and XNOR, are based on the 3 basic ones.

The AND Gate

The AND gate is so-called because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate.

The output is "true" when both inputs are "true." Otherwise, the output is "false."



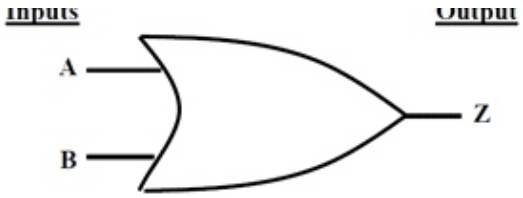
Logic Function: $Z = AB$

Truth Table:

Inputs		Output
A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

The OR Gate

The OR gate gets its name from the fact that it behaves after the way of the logical "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false."



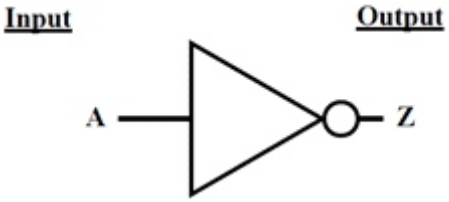
Logic Function: $Z = A + B$

Truth Table:

Inputs		Output
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

The NOT Gate

A logical inverter, sometimes called a NOT gate to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state (i.e., its input).



Logic Function: $Z = \overline{A}$

Truth Table:

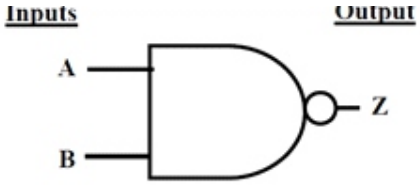
Input	Output
A	Z
0	1
1	0

As previously considered, the AND, OR and NOT gates' actions correspond with the AND, OR and NOT operators.

More complex functions can be constructed from the three basic gates by using DeMorgan's Law.

The NAND Gate

The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true". It finds the AND of two values and then finds the opposite of the resulting value.



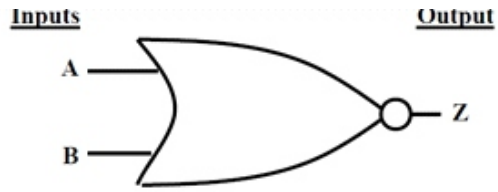
Logic Function: $Z = \overline{AB}$

Truth Table:

Inputs		Output
A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

The NOR Gate

What NOR gate simply stands for is a combination of an OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false". It finds the OR of two values and then finds the complement of the resulting value.



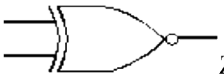
Logic Function: $Z = \overline{A + B}$

Truth Table:

Inputs		Output
A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

The XOR Gate

The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs, are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.



Z= $\overline{(A \oplus B)}$

XNOR gate

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

Summary

In this unit, you have learnt :

logic reasoning, logic gates as well as the logic function and truth tables for each of the gates in this unit. The basic gates are AND, OR and NOT gates; others are NOR, NAND, EXOR, and EXNOR.

Self-Assessment Questions

1. a.List at least five (5) types of gates.
2. b.Mention the logic function associated with each gate.
3. c.Draw the truth table associated with each gate.

Tutor Marked Assessment

- Draw the physical representation of the AND, OR, NOT and XNOR logic gates.
- Draw the logic circuit and truth table for
  - I. Z= ABC,
  - II. W= (P.Q̄) (R+S̄)

Further Reading

- https://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR
- https://www.electronics-tutorials.ws/logic/logic_1.html
- http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/

References

- Gupta A, Arora S. Industrial automation and robotics. Laxmi Publications; 2009.
- https://books.google.com.ng/books?hl=en&lr=&id=Y7rgCP7iC18C&oi=fnd&pg=PA1&dq=Industrial+Automation+and+Robotics+a.+K+Gupta+s.K.+Arora&ots=e4KP0Fl_g9&sig=5FeHKe3utUmUlfjaTLFQf-RbkMY&redir_esc=y#v=onepage&q=Industrial%20Automation%20and%20Robotics%20a.%20K%20Gupta%20s.K.%20Arora&f=false
- Wassell I. J. Digital Electronics: Part I – Combinational and Sequential Logichttps://www.cl.cam.ac.uk › teaching › DigElec › Digital_Electronics_08_pdf

A computer motherboard

Photo by:
Christian Wiediger on Unsplash

UNIT 2

Combinatorial Logic Circuits

Introduction

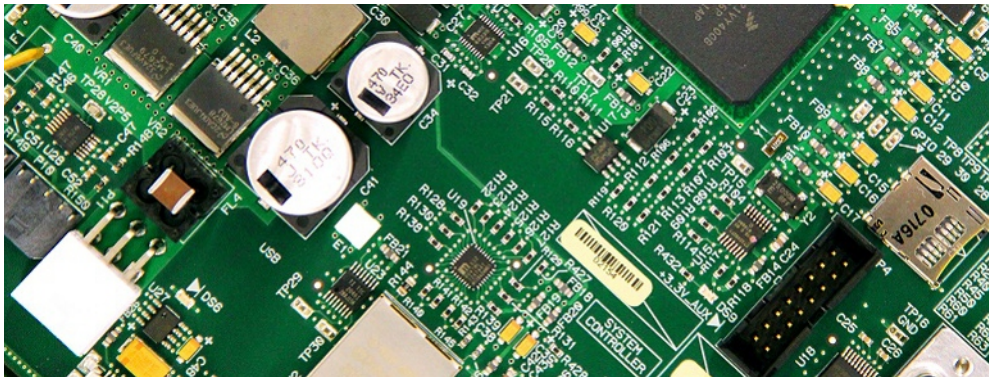
We can combine different gates to build digital circuits. As you learnt in the previous module, algebraic functions can be reduced using k-map or Boolean reduction. The reduced logic can relate to a decrease in the cost of building a circuit.

Learning Outcomes

At the end of this unit, you should be able to:

- combine different gates to form a logic circuit; and
- draw the associated truth table for the logic circuit.

Main Content



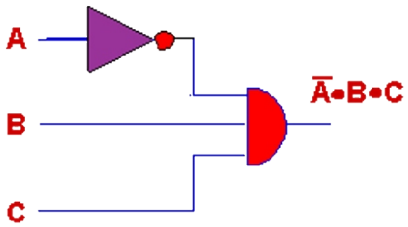
Combinatorial Logic Circuits

02 Mins

With the combinations of several logic gates, complex operations can be performed by electronic devices. Arrays (arrangement) of logic gates are found in digital integrated circuits (ICs).

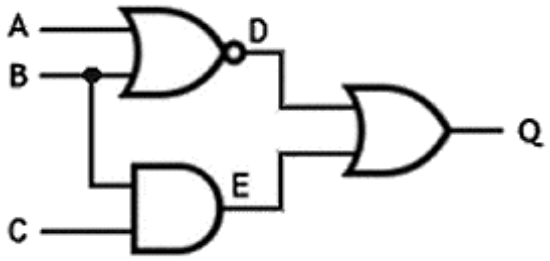
As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing much-more-complicated operations at an increased speed.

Combination of gates



A	B	C	Ā	ĀBC
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

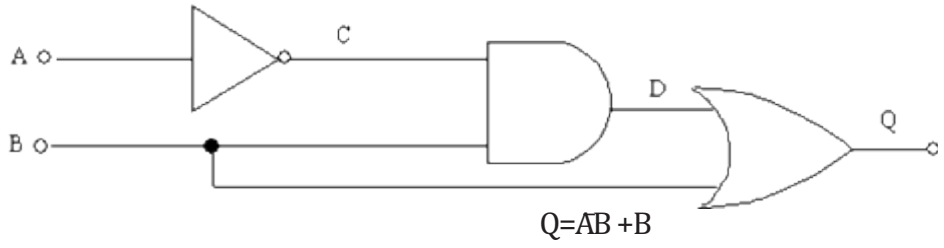
A goes into the NOT gate and is inverted; after this, it goes into the AND gate along with the variables B and C. The final output at the output terminal of the AND gate is ĀBC. More complex circuitry can be developed using the symbolic representation in this same manner.



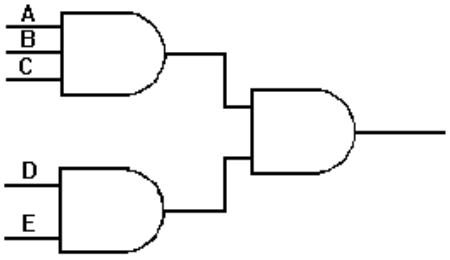
$Q = \overline{A + B} + BC$

A	B	C	D	E	Q
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Basically, there are 3 variables A, B, and C, do not be confused by the presence of D, E. Variables A and B goes into a NOR gate, B goes into AND gate along with variable C. The B is reused from the earlier defined one so as not to waste resources or have repetition. The output of the Nor and And gates serves as input to the Or gate.



$Q = \bar{A}B + B$



Q=(ABC)(DE)

Summary

In this chapter, you have learnt how to combine different gates

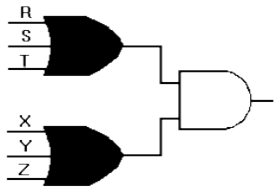
Self-Assessment Questions

1. i.Combine gates to draw 4 logic circuits, combining at least 3 gates in each.
2. ii.Draw the logic gate and associated logic circuits for the following functions
   A X = ABCD + FG
   B Z= ABC + CDE + ACF

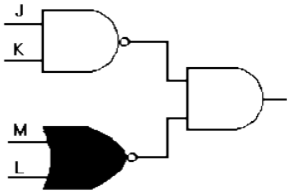
Tutor Marked Assessment

Write out the logic function of the gates below:

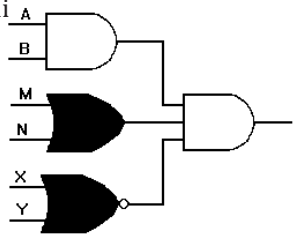
i



ii



iii



Further Reading

- https://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR
- https://www.electronics-tutorials.ws/logic/logic_1.html
- http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/

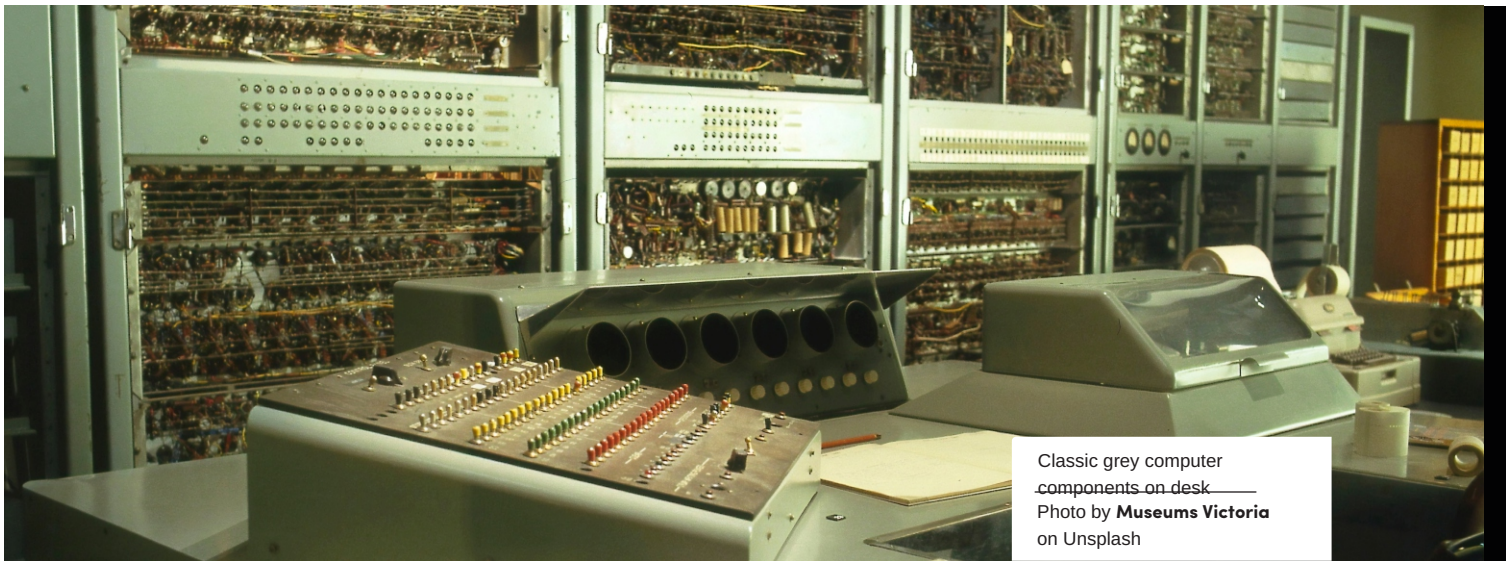
References

- Gupta A, Arora S. Industrial automation and robotics. Laxmi Publications; 2009.
- https://books.google.com.ng/books?hl=en&lr=&id=Y7rgCP7iC18C&oi=fnd&pg=PA1&dq=Industrial+Automation+and+Robotics+a.+K+Gupta+s.K.+Arora&ots=e4KP0Fl_g9&sig=5FeHKe3utUmUlfjaTLFQf-RbkMY&redir_esc=y#v=onepage&q=Industrial%20Automation%20and%20Robotics%20a.%20K%20Gupta%20s.K.%20Arora&f=false
- Wassell I. J. Digital Electronics: Part I – Combinational and Sequential Logichttps://www.cl.cam.ac.uk › teaching › DigElec › Digital_Electronics_08_pdf

Photo By: **Alex Motoc** on Unsplash

Module 5

Computer Programming Languages



Classic grey computer components on desk
Photo by **Museums Victoria** on Unsplash

UNIT 1

Program and Evolution of Languages

Introduction

I welcome you to the fifth and final module of CSC 111- Introduction to Computer Science I. In the previous modules, I have introduced you to different operations of the computers. You will recall that we defined information as data that is processed in line with a given instruction. Such instruction is regarded as a program. Programming is our concern in this first unit of Module 5. We use a program to execute tasks on computers. Programming languages are known to be the medium through which human beings communicate with the computer. Different programming languages have evolved over the years, and each has its target and features. The features of the language are used to measure its strength, and this will determine how the public will accept it.

Learning Outcomes

At the end of this unit, you should be able to:

- list ten (10) different programming languages and their authors;
- state the features of each programming languages; and
- state the features of a good programming language.

Main Content

Program

01 min

A program is a list of instructions in a logical sequence which are needed to be performed to accomplish a given task or solve a given problem on a computer. The process by which a user specifies to the computer in a particular programming language what s/he wants the computer to do is referred to as programming. Since the computer cannot think on its own, it is the programmer that will give the detailed steps, as well as the sequence in which steps are to be taken, in solving the problem, do you understand?

Programming Language

Programming Language is a set of specialized notations for communicating with the computer system.

Evolution of Programming Languages

01 min

May I let you know that hundreds of programming languages have been developed in the last fifty years? Many of them remained in the labs, and the ones which have good and more general features got recognized. Every language that is introduced comes with features upon which its success is judged. In the initial years, languages were developed for specific purposes, which limited their scope. However, as the computer revolution reached the common man, the language needed to be moulded to suit all kinds of applications. Every new language inherited certain feature from existing languages and added its features. The chronology of developments in programming languages is given below:

I. Lady Lovelace Ada Augusta made the first computer program in 1843 for an analytical engine application.

II. Konrad Zuse, a German, started a language design project in 1943. He finally developed plankalkul, programming calculus, in 1945. The language supported bit, integer, floating-point scalar data, arrays, and record data structures.

III. In the early 1950s, Grace Hopper and his team developed A-O Language. During this period, assembly language was introduced.

IV. The major milestone was achieved when John Backus developed FORTRAN (Formula Translator) in 1957. The FORTRAN data is oriented around numerical calculations. It was a major step towards the development of full-fledged programming language including control structures, conditional loops, and input and output statements

V. ALGOL was developed by GAMM (German Society of Applied mathematics) and ACM (Association of Computing Machinery) in 1960

VI. COBOL (Common business-oriented Languages) was developed for business purposes by the US department of defence in 1960.

VII. BASIC (beginner's All-purpose symbolic instruction code) was developed by John Kemeny and Thomas Kurtz in the 1960s

VIII. Niklaws around the 1970s developed PASCAL. PASCAL was named after French philosopher, Blaise pascal.

IX. In early '70s Dennis Ritchie developed C at Bell laboratories using some of the B languages. Features.

X. C++ was developed by Bjarne Stroustrup in early 1980s extending features of C and introducing object-oriented features

XI. Java, originally called Oaks, was developed by Sun Microsystems of USA in 1991 as a general-purpose language. Java was designed for the development of software for consumer electronic devices. It was a simple, reliable, portable and powerful language.

I want you to bear in mind that a language may be extremely useful for one type of application. For example, a language such as cobol is useful for business application but not for embedded software. Based on application, programming languages can be broadly classified as =

BUSINESS = COBOL

SCIENTIFY = FORTRAN

INTERNET = JAVA

SYSTEM = C, C++

Artificial intelligence (AI): LISP and PROLOG

Activity	Who was the first programmer? Do you remember her contribution to the evolution of computer system? (See Module 1)

Features of Good Programming Languages

04 mins

Let us now examine the qualities of a good programming language. The features of one programming language may differ from another. One may be easy and simple, while another may be difficult and complex. The program written for a specific task may have few lines in one language and many lines in another. The success and strength of a programming language are judged concerning standard features. To begin the language selection process, it is important to establish some criteria that make a language good. A good language choice should provide a path into the future in several important ways.

(a) Ease of use: this is the most important factor in choosing a language. The language should be easy in writing codes for the programs and executing them. The ease and clarity of a language depend upon its syntax. It should be able to provide a clear, simple, and unified set of concepts. The vocabulary of the language should resemble English (or some other natural language). Any concept that cannot easily be explained to amateurs should not be included in the language. Part-time programmers do not want to struggle with difficult concepts; they just want to get a job done quickly and easily.

(b) Portability: the language should support the construction of code in a way that it could be distributed across multiple platforms (operating systems). Computer languages should be independent of any particular hardware or operating systems, that is, programs written on one platform should be able to be tested or transferred to any other computer or platform, and there it should perform accurately.

(c)Reliability: the language should support the construction of components that can be expected to perform their intended functions satisfactorily throughout its lifetime. Reliability is concerned with making a system failure-proof and thus is concerned with all possible errors. The language should have the support of error detection as well as prevention. It should make some kinds

(d) Safety: safety is concerned with the extent to which the language supports the construction of critical safety systems, yielding systems that are fault-tolerant, fail-safe or robust in the face of systemic failures. The system must always do what is expected and be able to recover from any situation that might lead to a mishap or actual system hazard. Thus, safety tries to ensure that any failure that occurs result in minor consequences, and even potentially dangerous failures are handled in a fail-safe fashion. Language can facilitate this through such features as built-in consistency checking and exceptional handling.

(e) Performance: In some applications, performance is a big issue. By performance, we mean that the language should not only be capable of interacting with the end-users, but also with the hardware. The language should also support software engineering mechanism, discouraging or prohibiting poor practices and supporting maintenance activities. This is the main reason why C language is used for developing operating systems.

(f) Cost: Cost component is a primary concern before deploying a language at a commercial level. It includes several costs such as; program execution and translation cost, program creation, testing and use, program maintenance

(g) Compact Code: A good language should also promote compact coding; that is, the intended operations should be coded in a minimum number of lines. Even if the language is powerful, and is not able to perform the task in a small number of codes, then it is bound to be unpopular. This is the main reason for the C language's popularity over other languages in developing complex applications. Larger codes require more testing and developing time, thereby increasing the cost of developing an application.

(h) Maintainability: creating an application is not the end of the system development. It should be maintained regularly so that it can be modified to satisfy a new requirement or to correct deficiencies. Maintainability is facilitated by most of the languages, which makes it easier to understand and then change the software. Maintainability is closely linked with the structure of the code. If the original code were written in an organized way (Structural Programming), then it would be easy to modify or add new changes.

(I) Provides Interface to Other Language: From the perspective of the language, interface to other language refers to the extent to which the selected language supports interfacing feature to other languages. This type of support can have a significant impact on the reliability of the data, which is exchanged between applications, developed with different languages. In case of data exchange between units of different languages, without specific language

(j) Concurrency Support: Concurrency support refers to the extent to which inherent language supports the construction of code with multiple threads of control (also known as parallel processing). For some applications, multiple threads of control are very useful or even necessary. This is particularly true for real-time systems and those running on architecture with multiple processors. It can also provide the programmer with more control over its implementation. Other features include Reusability and Standardization.

Summary

In this unit, you have learnt :

i.five (5) different programming languages and their authors.

ii. Five (5) features of each programming languages.

iii. Features of a good programming language.

Self-Assessment Questions

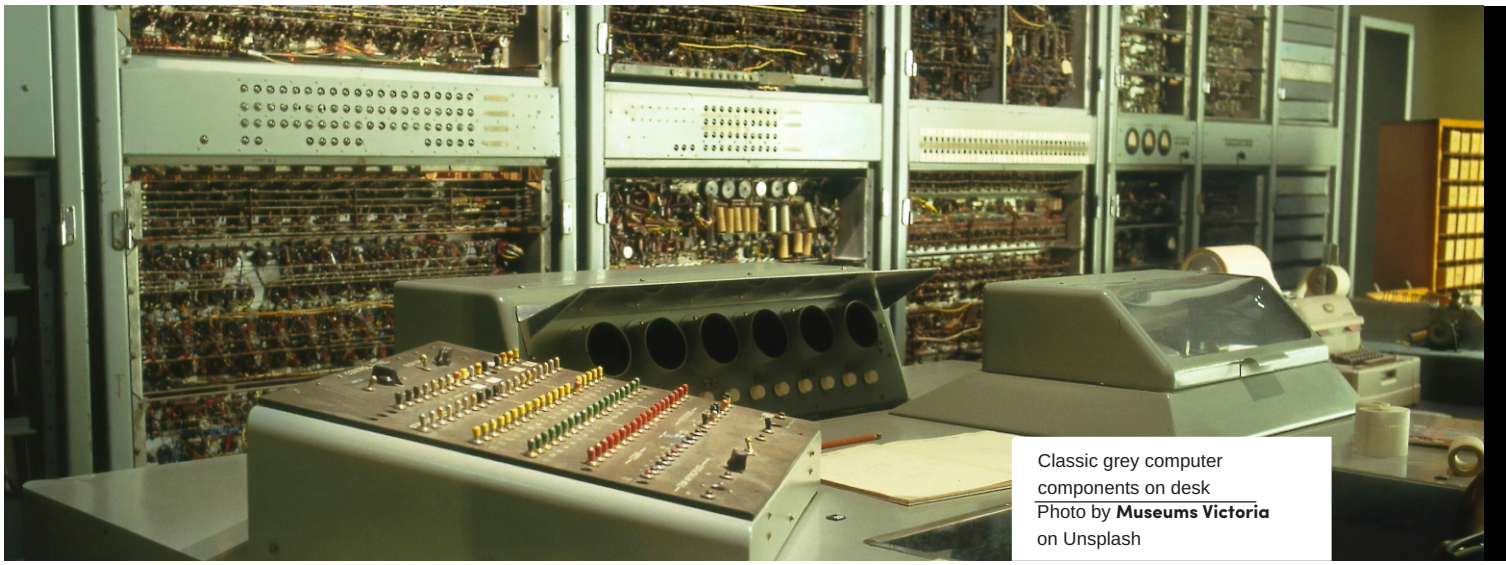
1. What is a program?
2. Discuss the evolution of programming language from Ada Lovelace to Java.

Tutor Marked Assessment

- "An action is to occur at a particular time, is a program" True/False. Justify your answer.
- If Cobol is good for business, identify the applications for Java, Pascal, C++, and Basic
- List and explain five (5) features of a good programming language

Further Reading

- https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading13.htm
- https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Program_Development
- http://interactivepython.org/runestone/static/CS152f17/GeneralIntro/Glossary.html



Classic grey computer components on desk
Photo by **Museums Victoria** on Unsplash

UNIT 2

Classification and Generations of Programming Languages

Introduction

Welcome to the second unit of the fifth module. In this unit, I will take you through the classifications and generations of computer programming languages. The primary computer language is a machine language that consists of 0s and 1s. Communication with the computer is through the use of machine language. This language is cumbersome and not easy to remember, which led to the development of assembly language and high-level language that is more like English in nature. The classification and generation of computers are based on machine language, assembly language and High-level language

Learning Outcomes

At the end of this unit, you should be able to:

- mention programming languages according to their generations; and
- differentiate between the generations of languages

Main Content

Classification of Programming Languages

04 mins

Computers understand only one language, and that is binary language (the language of 0's and 1's) also known as machine language. In the early years of computer programming, all the instructions were given in binary form only. Although the computer easy understood these programs, it proved too difficult for a human being to remember all the instructions in the form of 0's and 1's. Therefore, the computer remained a mystery to a common man until other languages such as assembly and high –level languages were developed, which were easier to learn and understand. These languages use commands that have some degree of similarity with English (such as if else, exit)

Programming languages can be grouped into three major categories: machine language, assembly (low-level) language and high–level languages.

1. Machine language: Machine language is the native language of computers. It uses only 0s and 1s to represent data and the instructions written in this language, consists of a series of 0s and 1s. Machine language is the only language understood by the computer. The machine language is peculiar to each type of computer.

2. Assembly (low-level) language: Assembly language correspondences are symbolic instructions and executable machine codes and were created to use letters instead of 0s and 1s to run a machine. It is called low-level because of its closeness to the machine language.

3. High-level language: these languages are written using a set of words and symbols following some rules similar to a natural language such as English. The

Generations of Programming Language

16 mins

I have observed that since the early 1950s, programming languages have evolved tremendously. This evolution has resulted in the development of hundreds of different languages. It may interest you to know that with each passing year, the languages become user-friendlier and more powerful than their predecessors. We can illustrate the development of all the language

First Generation: Machine Language

It may interest you to know that the first language was binary, also known as machine language, which was used in the earliest computers and machines. We know that computers are digital devices, which have only two states, ON and OFF (1and 0). Hence, computers can understand only two binary codes, 1 and 0. Therefore, every instruction and data should be written using 0s and 1s. Machine language is also known as the computer's 'native' language because the computer directly understands this system of codes.

Advantages of machine language: Even though machine language is not a human-friendly language, it offers certain advantages, as listed below:

i. Translation free: Machine language is the only language that the computer can directly execute without the need for conversion. It is the only language that computer can understand. Even an application using high-level language has to be converted into a machine-readable form so that the computer can understand the instruction.

ii. High speed: Since no conversion is needed, the application developed using machine languages are extremely fast. It is usually used for complex application such as space control system, nuclear reactors, and chemical processing.

Disadvantages of Machines Languages: There are many disadvantages to using machines languages to develop a program.

i. machine-dependence: Every computer type differs from the other, based on its architecture. Hence, an application developed for a particular type of computer may not run on a different type of computer. This may prove to be both costly as well as difficult for the organization. E.g., program written for one machine, say IBM 370 cannot be executed by another machine say HP 530.

ii. Complex languages: as I have consistently mentioned, machine language is very difficult to read and write. Since all the data and instruction must be

iii. Error-prone: Since the programmer has to remember all the opcodes (Operation Codes) and the memory location, it is bound to be error-prone. It takes a superhuman effort to keep track of the logic of the problems and, therefore, result in frequent programming errors.

iv. Tedious: Machine language poses real problems while modifying and correcting a program. Sometimes the programming becomes too complex to modify, and the programmer has to re-program the entire logic again. Therefore, it is very tedious and time-consuming, and since time is a precious commodity, programming using machine languages tends to be costly.

Due to these overwhelming limitations, machine languages are rarely used nowadays.

Second Generation: Machine Language

The complexities of machine languages led to the search for another language, and the assembly language was developed. It was developed in the early 1950s, and the main developer was IBM. Assembly language allows the programmers to interact directly with the hardware. This language assigns mnemonic codes to each machine language instruction to make it easier to remember or write. It allows a better human-readable method of writing program as compared to writing in binary bit patterns.

Unlike other programming languages, assembly language is not a single language but a group of languages. Each processor family (and sometimes individual processors within a processor family) has its assembly languages.

An assembly language provides mnemonic instructions, usually three letters long, corresponding to each machine instruction. The letters are usually abbreviated indicating what the instruction does: For example, ADD is used to perform an addition operation, MUL for multiplication, and so on. Assembly languages make it easier for humans to remember how to write instruction to the computer, but an assembly language is still a representation of the computer's native instruction set. Since each type of computer uses a different native instruction set, assembly languages cannot be standardized from one machine to another, and instructions from one computer cannot be expected to work on another.

Assembler:

Assembly language is nothing more than a symbolic representation of machine code, which also allows a symbolic designation of memory location. However, no matter how close assembly language is to machines codes, the computer still cannot understand it. The assembly language programs must be translated into machine codes by a separate program called Assemblers. The assembler program recognizes the character strings that make up the symbolic names of the various machine operations and substitute the required machine code for each instruction. At the same time, it also calculates the required address in memory for each symbolic name of a memory location. It substitutes those addresses for the names resulting in a machine language program that can run on its own at any time. An assembler converts the assembly codes into binary codes, and then it assembles the machine-understandable code into the main memory of the computer, making it ready for execution.

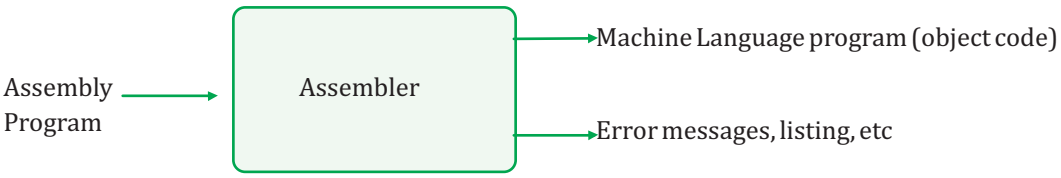


Figure 5.1: The working of an Assembler.

The original assembly language program is also known as the source code, while the final machine language program is designated the object code. If an assembly language program needs to be changed or corrected, it is necessary to make the changes to the source code and then re-assemble it to create a new object program. The functions of an assembler are given below:

a. It allows the programmer to use mnemonics while writing source code programs, which are easier to read and follow.

b. It allows the variable to be represented by symbolic names, not as memory locations.

c. It translates mnemonic operations codes to machine code and corresponding register addresses to system addresses.

d. It checks the syntax of the assembly program and generates diagnostic messages on syntax errors.

e. It assembles all the instructions in the main memory for execution.

f. In case of large assembly programs, it also provides linking facility among the subroutines.

g. It facilitates the generations of output on required output medium.

Advantages of Assembly Language:

The advantages of using assembly language to develop a program are:

i. Easy to Understand and Use: Assembly language uses mnemonics instead of using numerical opcodes and memory locations used in machine language. Hence, the programs written in assembly language are much easier to understand and use when compared with machine language. Being a more user-friendly language as compared to machine language, assembly programs are easier to modify.

ii. Less Error-Prone: Since mnemonic codes and symbolic addresses are used, the programmer does not have to keep track of the storage locations of the information and instruction. Hence, there is bound to be less error while writing an assembly language program. Even in cases of errors, assembly programs provide better facility to locate and correct them as compared to machine language programs.

iii. Efficiency: Assembly programs can run much faster and use less memory and other resources than a similar program written in a high-level language. Speed increment of 2 to 20 times faster is common, and occasionally, an increase of hundreds of times faster is also possible. Apart from speed, assembly programs are also memory efficient; that is, the memory requirement of a program (size of code) is usually smaller than a similar program written in high-level language.

iv. More Control on Hardware: Assembly language also gives direct access to key machine features essential for implementing certain kinds of low-level routines such as an operating system kernel or micro-kernel, device drivers, and

Disadvantages of Assembly Language:

The disadvantages of using assembly language to develop a program are:-

i. Machine dependence: Different computer architectures have their machine and assembly languages, which means that programs written in these languages are not portable to others (incompatible systems). If an assembly program is to be shifted to a different type of computer, it has to be modified to suit the new environment.

ii. Harder to Learn: The source code for an assembly language is cryptic (has hidden meaning) and in a very low machine-specific form. Being a machine-dependent language, every type of computer architecture requires different assembly languages, making it nearly impossible for a programmer to remember and understand every dialect of assembly. More skilled and highly trained programmers, who know all about the logical structure of the computer, can only create applications using assembly language.

iii. Slow Development Time: Even with highly skilled programmers, assembly generated application is slower to develop as compared to high-level language-based applications. In case of assembly language, several lines of assembly code are required for a line of high-level code, and the development time can be 10 to 100 times in comparison to the high-level language generated application.

iv. Less Efficient: A program written in assembly language is less efficient as compared to an equivalent machine language program because every assembly instruction has to be converted into the machine. Therefore, the execution of an assembly language program takes more time than its equivalent machine language program. Moreover, before executing an assembly program, the assembler has to be loaded in the computer's memory for translation, and it occupies a sizeable memory of the computer.

V. Not Standardized: Assembly language cannot be standardized because each type of computer has a different instruction set and, therefore, a different assembly language.

vi. No Support for Modern Software Engineering Technology:-Assembly

languages provide no inherent support for software engineering technology. They work with just machine-level specifics, not with abstractions. Assembly language does not provide inherent support for safety-critical systems. It provides very little opportunity for reuse, and there is no object-oriented programming support. There is also no specific support for distributed systems. The tools available for working with assembly language are typically very low-level tools.

Third Generation: Machine Language

In the 1960s, computers started to gain popularity and it became necessary to develop languages that were more like natural languages such as English so that a common user could use the computer sufficiently. Since assembly language required deep knowledge of computer architecture, it demanded programming as well as hardware skills to use computers. Due to computer's widespread usage, the early 1960s saw the emergence of the third programming languages (3GL). Languages such as COBOL, FORTRAN, BASIC, and C are examples of 3GLs and are considered high-level languages.

Using a high-level language, programs are written in a sequence of statements that impersonate human thinking to solve a problem. For example, the following BASIC code snippet will calculate the sum of two numbers.

```
LET X = 10
LET Y = 20
LET sum = X + Y
PRINT SUM
```

The first two statements store 10 in variable X (memory location name) and 20 in variable Y, respectively, the third statement again creates a variable named sum, which will store the summation of X and Y value.

Finally, the output is printed, that is the value store in sum is printed on the screen. From this simple example, it is evident that even a novice user can follow the logic of the program.

Activity	Consider different computer programmes on your PC and discuss their Language of development using the LMS forum.

Translating High-level Language To Machine Language

Since computers understand only machine language, it is necessary to convert the high-level programs into machine language codes. This is achieved by using language translators or language processors, generally known as compilers, interpreters or other routines that accept statements in one language and produces equivalent statements in another language.

A. COMPILER: A compiler is a kind of translator that translates a program into another program, known as a target language. Usually, the term compiler is used for language translator of high-level languages into machine language. The compiler replaces a single high-level statement with a series of machine language instruction. When a program is to be compiled, its compiler is loaded into main memory. The compiler stores the entire high-level program scans it and translates the whole program into an equivalent machine language program. During the translation process, the computer reads the stored program and checks the syntax (grammatically) errors. If there is any error, the compiler generates an error message, which is usually displayed on the screen. In case of errors, the compiler will not create the object code until all the errors are rectified.

Once the program has been compiled, the resulting machine code is saved separately, and can be run on its own at any time, that is, once the object code is generated, there is no need for the actual source code. However, if the source code is modified, then it is necessary to recompile the program to effect the changes.

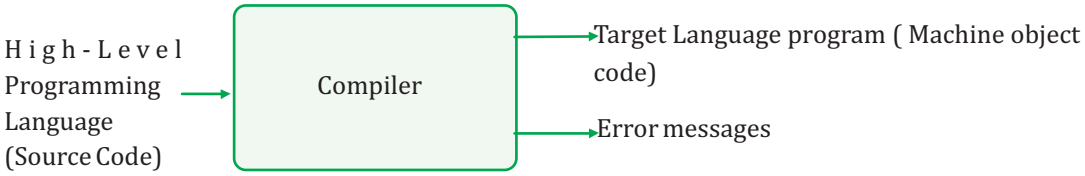


Figure 5.2: Compiler

NOTE: for each high-level language a separate compiler is required. For example, a compiler for C language cannot translate a program written in FORTRAN. Hence, to execute both language programs, the host computer must have the compilers of both languages

B. INTERPRETER: An interpreter is also a language translator and translates high-level language into machine language. However, unlike compilers, it translates a statement in a program and executes the statement immediately, that is, before translating the next source language statement. When an error is encountered in the program, the execution of the program is halted, and an error message is displayed. Similar to compilers, every interpreted language such as BASIC and LISP, has its interpreters.

Once the program has been compiled, the resulting machine code is saved separately, and can be run on its own at any time, that is, once the object code is generated, there is no need for the actual source code. However, if the source code is modified, then it is necessary to recompile the program to effect the changes.

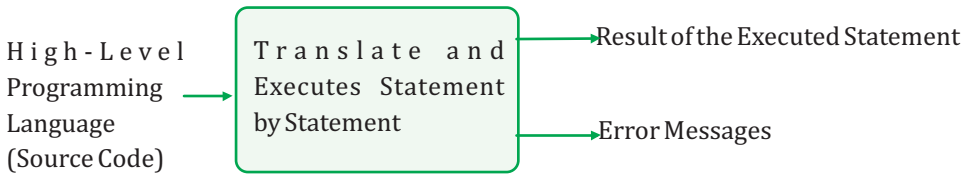


Figure 5.3: Working of an Interpreter

There are fundamental similarities in the functioning of interpreter and compiler. However, there are certain dissimilarities also, as given in Table 5.1 below

Table 5.1: Similarities and dissimilarities between the functions of Interpreter and Compiler

Basic	Compiler	Interpreter
Object Code	A compiler provides a separate object program	An interpreter does not generate a permanent saved object code file

Translation Process	A compiler converts the entire program into machine code at once	An interpreter translates the source code line-wise, that is it would execute the current statement before translating the next statement
Debugging Ease	Removal of errors (debugging is slow.)	Debugging becomes easier because the errors are pointed out immediately
Implementation	By nature, compilers are complex programs. Hence, they require hard-core coding. They also require more memory to execute a program	Interpreters are easier to write because they are less complex programs. They also require less memory for program execution
Execution Time	Compilers are faster as compared to interpreters because all statements are translated only once and saved in object files which can be executed anytime without translating again	Interpreters are slower as compared to compilers because each statement is translated every time it is executed from the source program.

Nowadays, many languages use a hybrid translator having the characteristics of a compiler as well as interpreter. In such a case, the program is developed and debugged with the help of interpreters, and when the program becomes bug-free, the compiler is used to compile it.

Advantages of High-Level Languages: High-level languages (HLL) are useful in developing complex software, as they support complex data structures. It increases the programmer's productivity (the number of lines of code generated per hour), unlike assembly language, the programmer does not need to learn the instruction set of each computer being worked with. The various advantages of using high-level languages are discussed below:-

(a) Readability: Since high-level languages are closer to natural languages, they are easier to learn and understand. In addition, a programmer does not need to be aware of computer architecture; even a common man can use it without much difficulty. This is the main reason for HLL's popularity.

(b) Machine Independence: High-level language is machine independent in the sense that a program created using HLL can be used on different platforms with very little or no change at all.

(c) Easy Debugging:- High-level languages include the support for ideas of abstraction so that programmers can concentrate on finding the solution to the problem rapidly, rather than on low-level details of data representation, which results in fewer errors. Moreover, the compilers and interpreters are designed in such a way that they detect and point out the errors instantaneously. Hence, the programs are free from all syntax errors.

(d) Easier to Maintain:- As compared to machine and low-level language, the program written in HLL are easily modified because HLL programs are easier to understand.

(e) Low Development Cost: High-level language permits faster development of programs although a high-level program may not be as efficient as an equivalent machine and low-level programs. But the savings in programmer time generally outweigh the inefficiencies of the application.

(f) Easy Documentation: Since the statements are written in HLL are similar to natural languages, human beings can easily understand them. As a result, the code is obvious; that is, there is few or no need for comments to be inserted in programs.

Disadvantages of High-Level Languages: There are two main disadvantages of high-level language

i. Poor Control on Hardware: High-level language is developed to ease the pressure on programmers so that they do not have to know the intricacies (complexity) of hardware. As a result, sometimes the applications written in high-level languages cannot completely harness the total power available at hardware level.

ii. Less Efficient: The HLL programs are less efficient as far as computation time is concerned. This is because, unlike machine language, high-level languages must be created and sent through another processing program known as a compiler. This process of translation increases the execution time of application

programs written in high-level language. They take more time to execute and require more memory space.

Some Popular High-level Language

Although several languages evolved in the last five decades, only a few languages were considered worthwhile to be marketed as commercial products. Some of the commonly used high-level languages are discussed as follows:-

(a). **FORTRAN:** FORTRAN, or FORMULA TRANSLATOR, was developed by John Backus for IBM 704 mainframes in 1957. The IBM 704 machines were considered as inseparable with FORTRAN: They were the first machines to provide indexing and floating-point instruction in hardware. FORTRAN gained immense popularity as compared to any of its counterparts and is still extensively used to solve scientific and engineering problems.

The main feature of FORTRAN is that it can handle complex numbers very easily. However, the syntax of FORTRAN is very rigid. A FORTRAN program is divided into sub-programs, each sub-program is treated as a separate unit, and they are compiled separately. The compiled programs are linked together at load time to make a complete application. It is not suitable for a large amount of data as well and, hence, it is not often used for business applications.

(b) COBOL: COBOL, or Common Business Oriented Language, has evolved after many design revisions. Grace Murray Hopper, on behalf of US Defence, was involved in the development of COBOL as a language. She showed for the first time that a system could use the English Language like syntax, suiting to the business notations rather than scientific notations. The first version was released in 1960 and later revised in 1974 and 1984. COBOL was standardized with revisions by ANSI in 1968.

COBOL is considered a robust language for the description of Input/ Output formats. It could cope with large volumes of data. Due to its similarity with English, COBOL programs are easy to read and write. Since it uses English words rather than short abbreviations, the instructions are self-documentary and self-explanatory. However, due to its large vocabulary, the programs created using

©. BASIC: Beginner's All-Purpose Symbolic Instruction Code, was developed by John Kemeny and Thomas Kurtz at Darmouth College in the year 1960. It was the first interpreted language made available for general use. It is now in such widespread use that most people see and use this language before they deal with others. Presently many advanced versions of BASIC are available and used in a variety of fields as business, science and engineering.

Basic program was traditionally interpreted. This meant that each line of code had to be translated as the program was running. BASIC programs, therefore, ran slower than FORTRAN programs. However, if a BASIC program crashed because of a programming error, it was much easier to identify the source of the problem, and in some cases, the program could even be restarted at the point where it has broken down. In the BASIC program, each statement is prefixed by a line number, which serves a dual purpose to provide a table for every statement and to identify the sequence in which the statement will be executed. BASIC is easy to learn as it uses common English words. Therefore, it is a good language for beginners to learn their initial programming skills.

(d). PASCAL: Named after Blaise Pascal, a French philosopher, mathematician, and physicist, PASCAL was specifically designed as a teaching language. Niklaus Wirth developed the language at the Federal Institute of Technology of Zurich in the early 1970s.

PASCAL is a highly structured language, which forces programmers to design programs very carefully. Its object was to force the student to learn the techniques and requirement of structured programming correctly. PASCAL was designed to be platform-independent, that is a PASCAL program could run correctly on any other computer, even with a different and incompatible type of processor. The result was a relatively slow operation, but it did work in its fashion.

(e). C: C was initially developed as an experimental language called A. Later on, it was improved, corrected and expanded until it was called B. This language, in turn, was improved, upgraded, and debugged and was finally called C. C was developed by Dennis Ritchie at Bell Labs in the mid1970s. It was originally

C consists of a rich collection of standard functions useful for managing system resources. It is flexible, efficient, and easily available. Having syntax close to English words, it is an easy language to learn and use. The applications generated using C language programs are portable; that is, the programs written in C can be executed on multiple platforms. C works on a data structure, which allows a simple data storage. It has the concept of pointers, the memory addresses of variables and files.

(f). C++: Bjarne Stroustrup developed this language in the early 1980s. It is the superset of C and supports object-oriented features. This language is used effectively in developing system software as well as application software. As it was an extension of C, C++ maintained the efficiency of C and added the power of inheritance. C++ works on classes and objects as a backbone of object-oriented programming. Being a superset of C, it is an extremely powerful and efficient language. However, C++ is much harder to learn and understand than its predecessor C. The salient features of C++ are:-

- Strongly typed
- Case-sensitive
- Compiled and faster to execute
- Platform independent

(g). JAVA:- This language was developed by Sun Microsystems of the USA in 1991. It was originally called 'Dak'. Java was designed for the development of software for consumer electronic devices. As a result, Java came out to be a simple, reliable, portable, and powerful language. This language truly implements all the object-oriented features. Java was developed for internet and contributed a lot to its development. It handled certain issues like portability, security, networking, and compatibility with various operating systems. It is immensely powered on the web and is used for creating scientific and business applications:-

The features of Java includes;

- Simple and robust language.
- Secure and safe language.
- Truly object-oriented language.
- Portable and platform-independent
- Multithreaded, distributed, and dynamic.

Fourth Generation: 4 Gl

Fourth-generation language (4GLs) have simple, English-like syntax rules, commonly used to access databases. The third-generation programming language is considered as procedural languages because the programmer must list each step and must use logical control structures to indicate the order in which instructions are to be executed. 4GLs, on the other hand, are non-procedural languages. The non-procedural method is simply to state the needed output instead of specifying each step one after another to perform a task. In other words, the computer is instructed WHAT it must do rather than HOW a computer must perform a task.

The non-procedural method language is easier to write, but has less control over how each task is performed. When using non-procedural languages, the methods used and the order in which each task is carried out is left to the language itself; the user does not have any control over it. Also, 4GLs sacrifice computer efficiency to make programs easier to write. Hence, they require more computing power and processing time. However, with the increase in power and speed of hardware and with diminishing costs, the use of 4GLs have spread.

Fourth-generation languages have a minimum number of syntax rules. Hence, people who have not been trained as programmers can also use such languages to write applications programs. This saves time and allows professional programmers for more complex tasks. The 4GLs are divided into three categories:

1. Query Languages: they allow the user to retrieve information from databases by following simple syntax rules. For example, the database may be requested to locate details of all employees drawing a salary of more than $10000. Examples of

2. Report Generations:- They produce customized reports using data stored in a database. The user specifies the data to be in the reports format, and whether any subtotals and totals are needed. Often report specifications are selected from pull-down menus, making report generations very easy to use. Examples of report generators are Easytrieve plus by Pansophic and R&R Relational Report Writer by Concentric Data systems.

3. Application Generations: with application generations, the user writes programs to allow data to be entered into the database. The program prompts the user to enter the needed data. It also checks the data for validity. Cincom Systems MANTIS and ADS by Cullinet are examples of application generation.

Advantages of 4GLs:

The main advantage of 4GLs is that a user can create an application in a shorter time for development and debugging than with other programming languages. The programmer is only interested in what has to be done and that too at a very high level. Being non-procedural, it does not require the programmers to provide the logic to perform a task. Therefore, a lot of programming effort is saved as compared to 3GLs. Use of procedural templates and data dictionaries allow automatic type checking (for the programmer and user input), and this results in fewer errors. Using application generations, routine tasks are automated.

Disadvantages of 4GLs;

Since programs written in 4GL are quite lengthy, they need more disk space and a large memory capacity as compared to 3GLs. These languages are inflexible also because the programmer's control over language and resources is limited as compared to other languages. These languages cannot directly utilize the computing power available at the hardware level as compared to other levels of language.

Fifth Generation: 5 GL

Fifth-generation languages are a future concept. They are just like a conceptual view of what might be the future of programming languages. These languages will be able to process natural languages. The computers would be able to accept, interpret, and execute instructions in the nature or natural language of the end-users. The user will be freed from learning any programming language to communicate with the computers. The programmers may simply type the instruction or simply tell the computer via microphones what it needs to do. Since these languages are still in their infancy, only a few are currently commercially available. They are closely linked to artificial intelligence and expert systems.

Summary

You have reached the end of our interaction in this course.
In this unit, you learnt:

the classification and generations of computer programming languages;
the advantages and disadvantages of each generation; and
examples of languages in each generation

Self-Assessment Questions

1. Mention three (3) different programming languages according to their generations.
2. Differentiate between the generations of languages.
3. Explain the categories of 4GL.

Tutor Marked Assessment

- Differentiate between object code and source code
- Differentiate between compiler and assembler

Further Reading

- http://learnprogramming1.weebly.com/c/difference-between-source-code-and-object-code
- https://en.wikibooks.org/wiki/A-level_Computing/AQA/Computer_Components,_The_Stored_Progra

m_Concept_and_the_Internet/Fundamentals_of_Computer_Systems/Generations_of_programming_language

- https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/mjbn/article2.html