BASIC COMPUTATION AND PROGRAMMING WITH C

Subrata Saha
Subhodip Mukherjee

CAMBRIDGE

Basic Computation and Programming with C

Undergraduate students of engineering and computer science will come across detailed coverage of the fundamentals of computation and programming in C language, in this textbook. Essential concepts including operators and expressions, input and output statements, loop statements, arrays, pointers, functions, strings and pre-processor are described in a lucid manner. For better comprehension of the concepts, the book is divided into three parts: Fundamentals of Computer, Programming with 'C' and Technical Questions.

A unique feature of the book is 'Learn by Quiz – Questions/Answers', which has questions designed through confidence-based-learning methodology. This helps readers to identify right answers with adequate explanation and reasoning as to why the other options are wrong. Plenty of computer programs and review questions are interspersed throughout the text.

The book can also be used as a self-learning book by beginners in Computer Programming.

Subrata Saha is Head of the Department of Computer Applications, Techno India Hooghly, West Bengal. His areas of interest include object oriented languages, image processing and cryptography.

Subhodip Mukherjee is Head of the Department of Computer Applications, Techno India College of Technology, Kolkata. His areas of interest include object oriented languages, software engineering, computer architecture and database management systems.

Basic Computation and Programming with C

Subrata Saha
Subhodip Mukherjee

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781316601853

First published 2016

Printed in India

A catalogue record for this publication is available from the British Library

Library of Congress Cataloging-in-Publication Data
Names: Saha, Subrata, 1973- author. | Mukherjee, Subhodip, author.
Title: Basic computation and programming with C / Subrata Saha, Subhodip Mukherjee.
Description: New York : Cambridge University Press, 2016. | Includes bibliographical references and index.
Identifiers: LCCN 2016018907 | ISBN 9781316601853 (pbk.)
Subjects: LCSH: C (Computer program language) | Mathematics--Data processing.
Classification: LCC QA76.73.C15 S237 2016 | DDC 004.01/51--dc23 LC record available at https://lccn.loc.gov/2016018907

ISBN 978-1-316-60185-3 Paperback

Additional resources for this publication at www.cambridge.org/9781316601853

To my father Late Kamal Krishna Saha and to my mother Geetasree Saha for what I am today.

— Subrata Saha

To my mother who sacrificed all sorts of entertainment for my education during my school life and to my father Professor S. G. Mukherjee whose addiction to books and simple living inspired me to be a teacher.

— Subhodip Mukherjee

Contents

Figures

Tables

Preface

The C programming language is one of the most, academically as well as industrially, important programming languages in the world. It was unveiled in 1972 and since then, with gradual enhancements and enrichments, it has successfully established itself as a powerful language for programming microcontrollers, operating systems and various commercially significant software packages. Having unique features like block structure, stand alone functions and a rich set of keywords with very few restrictions, it is aptly regarded as the 'Mother Language' among programming languages. Hence it is necessary for every programmer to learn and, often, use C.

About the Book

Today, in any corner of the world, there is a necessity to learn computer programming for people to survive in any industry. All students do not possess technical background and therefore find learning C programming difficult. Keeping a strong focus on industrial requirements and the limitations of students from non-technical backgrounds, we have written this new book on C programming to enable students of non-technical as well as technical backgrounds learn this marvelous programming language in a completely new way.

This book is completely different from other popular C programming books available in the market. It teaches programming to someone who wants to learn how to program in C for the first time in his/her life with no programming knowledge at all. In other words, this book on C programming is for ***absolute beginners*** of programming but at the same time it outclasses several other C programming books in the market with its coverage. Although targeted at complete beginners, this book covers many advanced concepts in C programming and enables a non-programmer to develop into a competitive C programmer ready to face job interviews on C.

Structure of the Book

➢ This book is written presuming that the reader is not strong in English. All the explanations in each chapter are offered in simplest English to enable any reader to correctly understand a concept.

- ➢ Each chapter contains appropriate examples. It ends with a '*Learn by Quiz – Questions/Answers*', having multiple choice questions with their respective answers (with clarifications) to ensure understandability.
- ➢ In order to make the presentation visually interactive for students, neat labeled diagrams are provided wherever necessary. For each topic, the explanations are clear and concise, avoiding verbosity as much as possible.
- ➢ Each programming topic is presented with an exercise set consisting of a large number of solved C programs. Each such solved program is presented with WHYs and HOWs for each new or next statement of a program.
- ➢ This book also includes a rich collection of commonly asked as well as most probable, new C programming *interview solved questions* and programs with step-by-step explanations to enable a student to be successful in an interview.

Students, we have prepared this book for you using a unique teaching approach based on our academic learnings so that it looks least intimidating and most interesting. We are sure that after studying this new book on C programming, those of you for whom C programming has been a source of sorrow will now find in it joy and fun. Though every attempt has been made to avoid errors, we will be grateful to our readers if they bring to our notice any oversight they find. If the content design and organization of this book meets all the expectations and requirements of our students, then only can this book be considered a worthwhile accomplishment.

Acknowledgments

I thank my, beloved, students for their constant motivation which made me write this book despite the availability of many best-selling C programming books in the market. My students are extremely fond of and fascinated by my 'teaching from the ground up' style and their demand inspired me to author this amazingly simple C book which is aimed at non-programmers or absolute beginners. My heartiest thanks to them.

I would like to thank T. K. Ghosh, Chief Executive Director of Techno India Group for providing me the platform to teach so many young minds.

I want to thank M. K. Chakraborty for his constant support and inspiration.

I would also like to thank Subir Hazra, my friend and ex-colleague, who helped us in preparing the 'Learn by Quiz - Questions/Answers' section.

Most important, heartiest thanks to my family specially my wife Sriparna Saha and my beloved daughter Shreya for their support, encouragement, quite patient and unconditional love which helped, conclude the book.

Finally, I would like to thank all the reviewers of this book for their critical comments and suggestions. I convey my gratitude to Rachna Sehgal and the entire editing team at Cambridge University Press, India for their great work.

Subrata Saha

Thanks to T. K. Ghosh (Chief Executive Director) who found in me suitability to be the head of three departments of a Techno India Group college. He gave me the opportunity to prove myself, to be a good teacher to thousand of students.

Thanks to my teacher Professor Ranjan Dasgupta (Head, Computer Science, NITTTR-Kolkata) who gave me the confidence to write a book.

Thanks to my ex-colleague Professor Sumana Chakraborty who participated actively in discussions to freeze the table of contents of this book.

Subhodip Mukherjee

Part A

Fundamentals of Computer

1 Computer-History, Classification and Basic Anatomy

1.1 GENERATIONS OF COMPUTER

The first electronic digital computer, called the Atanasoff–Berry Computer (ABC), was built by Dr John V. Atanasoff and Clifford Berry in 1937. An electronic computer called the Colossus was built in 1943 for the US army. Around the same time, many others were also trying to develop computers. The first general-purpose digital computer, the Electronic Numerical Integrator and Computer (ENIAC), was built in 1946.

Computers since 1946 are categorized in five generations:

- First Generation: Vacuum Tubes
- Second Generation: Transistors
- Third Generation: Integrated Circuits
- Fourth Generation: Microprocessors
- Fifth Generation: Artificial Intelligence

Follows a brief description of each generation:

1.1.1 First Generation (1946–1956) Vacuum Tubes

Vacuum tubes were used to make circuits of first generation computers. For building memory, magnetic drums were used that were huge in size and weight. First generation computers were so large in size that they often took an entire room. They were also very prone to error. They were too expensive to operate and in addition consumed huge electricity. It is worth mentioning the amount of heat they generated. Despite using liquid based cooling system, they often got damaged due to heat.

Programs for first generation computers were written in machine language, the lowest-level programming language understood by computers, to perform operations. They were designed to solve only one problem at a time. Punched cards and paper tapes were used to feed input, and output was displayed on printouts.

The ENIAC (**Electronic Numerical Integrator and Computer)** and UNIVAC (**Electronic Discrete Variable Automatic Computer)** computers are examples of first-generation computing devices.

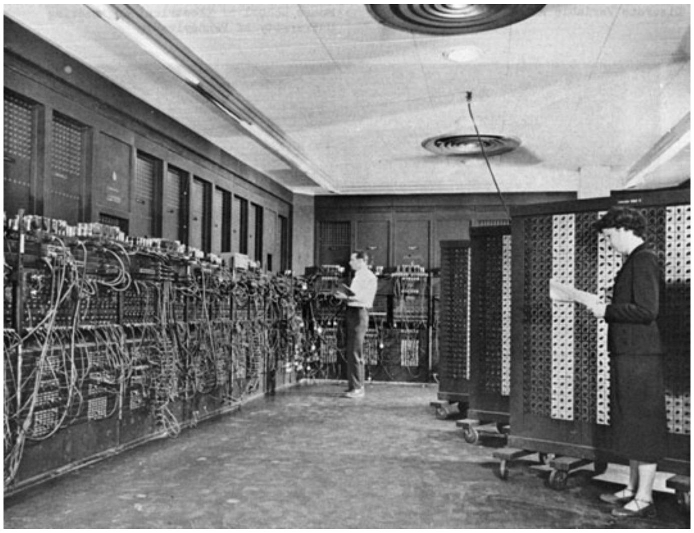


Fig. 1.1 ENIAC

ENIAC was the first electronic general-purpose computer. It was capable of being reprogrammed to solve various numerical problems. ENIAC was primarily designed to calculate artillery firing tables. It was mainly used in the United States Army's Ballistic Research Laboratory. ENIAC was introduced to the public at the University of Pennsylvania in 1946 as "Giant Brain." ENIAC' was funded by the United States Army.

ENIAC had a modular design. It had individual panels to perform separate functions. Twenty modules among them were accumulators that could add, subtract and hold a ten-digit decimal number in memory. Numbers were passed between these modules through several general-purpose buses. The modules were able to send and receive numbers, compute, save the answer and trigger the next operation without any moving component. That is why it could achieve high speed. Key to its versatility was the ability for branching. It could switch to different operations, depending on the sign of a computed result.

ENIAC contained 17,468 vacuum tubes, 1500 relays, 70,000 resistors, 7200 crystal diodes, 10,000 capacitors. It had a whopping 5,000,000 hand-soldered joints. It weighed more than 27 tons, was roughly 8 × 3 × 100 feet in size, occupied 1800 ft^2 and consumed 150 kW of electricity.

After ENIAC, a much improved computer named EDVAC (Electronic Discrete Variable Automatic Computer) was designed. EDVAC was a stored program computer. EDVAC was the first computer to work in binary number system. This is a major difference with ENIAC that used decimal number system.

1.1.2 Second Generation (1956–1963) Transistors

Transistors replaced vacuum tubes in the second generation of computers. The transistor was far superior to the vacuum tube in terms of size, generated heat and energy consumption. So computers made using transistors became smaller, faster, cheaper, more energy-efficient and more reliable.

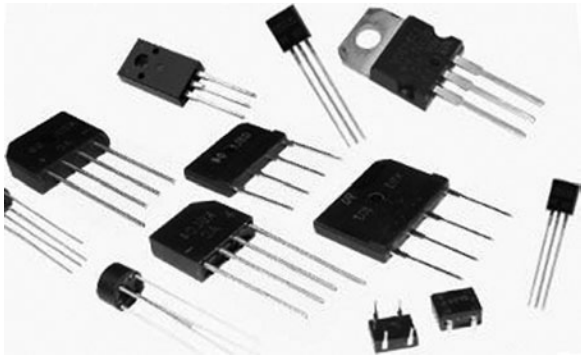


Fig. 1.2 Transistor

The transistors also generated a lot of heat that subjected the computer to damage. But it was much better than the vacuum tubes in terms of size and heat.

Second-generation computers used symbolic or assembly languages instead of binary machine language that allowed programmers to specify instructions in words instead of machine code or binary.

Second-generation computers were still using punched cards for input and print-outs for output. At this time, high-level programming languages, like early versions of COBOL and FORTRAN were being developed. The first computers that were developed around this time were used in the atomic energy industry.

1.1.3 Third Generation (1964–1971) Integrated Circuits

Third generation computers were based on integrated circuits (IC circuits). Transistors were much smaller. They were placed on silicon chips, called semiconductors. This invention dramatically increased the speed and efficiency of computers.

In these computers, users interacted through keyboards and monitors and interfaced with an operating system. Many different applications were possible to run at one time. A central program usually resided in memory to monitor others. Computers for the first time became accessible to a mass audience because they became cheaper and smaller.

1.1.4 Fourth Generation (1971–Present) Microprocessors

Brain of the fourth generation of computers is **microprocessors.** Thousands of integrated circuits were built on a single silicon chip. Computers of the size of an entire room in first generation could now fit in palm. The Intel 4004 chip, developed in 1971, contained all the components of the computer like the central processing unit, memory and input/output controls on a single chip.

In 1981 IBM introduced its first computer for the home user, and in 1984 Apple introduced the Macintosh. Microprocessors also moved out of the realm of desktop computers and into many areas of life as more and more everyday products began to use microprocessors.

As these small computers became more powerful, they could now be linked together to form networks, which eventually led to the development of the Internet. Fourth generation computers were also equipped with the mouse and other handheld devices. Graphical user interface (GUI) was also designed for these computers.

1.1.5 Fifth generation (Present and Beyond) Artificial Intelligence

Fifth generation computing devices are based on artificial intelligence and are still being developed, while some applications, like voice and handwriting recognisers, are in use today. Fifth-generation computing aims to develop devices that will be responsive to natural language input and will also be able to learn and self-organize.

1.1.6 Evolution of Intel Processors

As Intel processors or compatible processors are most popular for desktop systems, it is worth mentioning the various microprocessors introduced by Intel. The first processor available to public was 8085 having 40 pin, 2 MHz clock frequency, 6500 transistors, 8 bit data bus and 16 bit address bus. But first personal computer made by IBM with Intel microprocessor was based on 8086 having 1 MB addressable memory and 30000 transistors. Then came 80286 followed by 386SX, 386DX, 486 SX and 486 DX.

The 486 processors contained up to 1.4 million transistors, reached 100 MHz frequency and can address up to 4 Gigabyte of memory (32 bit address bus so 2^{32} = 4 GB). After 486, came the decade of Pentium, Pentium with MMX (multimedia extended) and Pentium II processors. Then the Pentium III processor touched the 1 GHz clock frequency mark. From mid of year 2000 the market leader was various versions of Pentium IV for steady six years, achieved clock frequency up to 3.8 GHz and packed up to whopping 180 million transistors in it. Presently core-i3, core-i5 and core-i7 are ruling the market though Intel dual core processors are also available. (In this span, Intel marketed few other processors but they did not get good response.)

1.2 CLASSIFICATION OF COMPUTERS

Computers can be classified in many ways depending on various features and criteria. Many of them will be mentioned in this section. Though readers of this are being introduced to computer and do not have an in-depth knowledge of computer architecture, some classifications based on various architectural difference are also being listed. This will build interest of the readers in the subjects such as 'MICROPROCESSOR', 'ARCHITECTURE', 'COMPUTER ORGANIZATION', 'PARALLEL PROCESSING', 'ASSEMBLY LANGUAGE PROGRAMMING', and 'PIPELINING'.

Note A computer's **architecture** is its abstract model and is the programmer's view in terms of instructions, addressing modes and registers. A computer's **organization** expresses the realization of the architecture. Architecture describes **what** the computer does and organization describes **how** it does it.

Here in this topic, following types of classifications will be discussed:

i. Based on computation power.
ii. Based on number of operands.
iii. Scalar VS Vector processor.

iv. Flynn's taxonomy
v. von Neumann vs. Harvard Architecture
vi. Big-endian vs. Little-endian

(i) Based on Size, Computing Power and Price

Based on their size, computing power and price the computers are broadly classified into four categories:

1. Microcomputers
2. Minicomputers
3. Mainframe computers
4. Supercomputer.

As the list descends, things will become large, expensive, complex and fast. Now follows a small introduction to all of them:

(1) Microcomputers are the most common kind of computers used by people today, whether in a workplace, at school or on the desk at home. These are microprocessor based systems. Microcomputers are small, low-cost and single-user digital computer. They consist of CPU, input unit, output unit, storage unit and the software. Microcomputers are stand-alone machines but they can be connected together to create a network of computers that can serve more than one user. Microcomputers include desktop computers, notebook computers or laptop, tablet computer, handheld computer, smart phones, etc. The most popular microprocessors in world are made by Intel and Apple for their Pentium based PCs and Apple Macintosh.

Here the evolution of Intel microprocessors will be worth mentioning. Intel first introduced the 8085 microprocessor having 8 bit data bus and 16 bit address bus but no personal computer was made based on it. The journey started with 8086 when IBM made IBM-AT based on it. Gradually came 80286, 386, 486 SX and 486 DX. Next processor was named Pentium with clock frequency 60 MHz to 300 MHz and introduced in the period 1993–1997. It had a 32 bit data bus and 32 bit address bus. The speed and internal cache memory size increased with time. Pentium II, Pentium III and Pentium IV were the successors. Then came Pentium DUAL CORE, Core2-DUO, and Quad-core processors having frequency in GHz level. Now the latest processors from Intel's stable are core-I3, core-I5 and core-I7.

(2) Minicomputer is a class of multi-user computers that lies in the middle range of the computing spectrum, in between the top end single-user systems (microcomputers or personal computers) and low end mainframe computers. They have high processing speed and high storage capacity than the microcomputers. Minicomputers can support up to 200 users simultaneously. The users can access the minicomputer through their PCs or terminal. They are used for various processor hungry applications in industries and research centers.

(3) Mainframe computers are multi-user, multi-programming and high performance computers. They operate at a very high speed, have huge storage capacity and can handle the workload of many users. The user accesses the mainframe computer via a terminal that may be a dumb terminal, an intelligent terminal or a PC. The terminals and PCs utilize the processing power and the storage facility of the mainframe computer. Mainframe computers are used primarily by corporate and governmental organizations for critical applications, bulk data processing such as census, industry and consumer statistics, enterprise resource planning and transaction processing.

(4) Supercomputers are the fastest and the most expensive machines. They have high processing speed compared to other computers. The speed of a supercomputer is generally measured in FLOPS (Floating point Operations Per Second). Some of the faster supercomputers can perform trillions of calculations per second. Supercomputers are built by interconnecting thousands of processors that can work in parallel. Supercomputers are used for highly calculation-intensive tasks, such as, weather forecasting, climate research, molecular research, biological research, nuclear research and aircraft design. Some examples of supercomputers are IBM Roadrunner, IBM Blue gene. The supercomputer assembled in India by C-DAC (Center for Development of Advanced Computing) is PARAM. PARAM Padma is the latest machine in this series. The peak computing power of PARAM Padma is One Tera FLOP.

(ii) Based on Number of Operands

The instruction-set of a computer is designed first, and then accordingly the architecture is designed. Just like a KEY is designed first then corresponding LOCK is assembled. A computer instruction should contain one operation code and zero or more operands on which the operation will be performed or where the result will be stored. Depending on maximum number of operands allowed in an instruction, computers can be classified:

1. Zero address machines (stack machine)
2. One address machine (accumulator machine)
3. Two address machine (Intel processors are of this type)
4. Three address machine

(iii) Scalar Processor vs Vector Processor

Each instruction executed by a scalar processor generally manipulates one or two data items at a time. On the contrary, instructions of a vector processor operate simultaneously on many data items.

(iv) By Flynn's Taxonomy

Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966. There are four categories defined by Flynn. His classification is based upon the number of concurrent instruction (or control) and data streams available in the architecture:

1. Single Instruction, Single Data stream (SISD) A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single Instruction Stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element to operate on single Data Stream.

2. Single Instruction, Multiple Data streams (SIMD) A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

3. Multiple Instruction, Single Data stream (MISD) Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance.

4. Multiple Instruction, Multiple Data streams (MIMD) Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

(v) von Neumann vs. Harvard Architecture

While talking about computers, there should be a topic on John von Neumann. This Hungary-born American mathematician gave the concept of stored-program architecture first but unfortunately in his life span no fully functional computer was made. His ideas will be discussed later on this chapter.

von Neumann architecture	Harvard architecture
1. Same memory for data & program	1. Separate data and code memory
2. Instructions are executed one by one. Execution requires at least two clock cycles. One for fetch and others for execution and pipelining is not possible.	2. Pipelining is possible.

(vi) Big-endian vs Little-endian

This is based on "How a large binary number is stored in 8 bit (byte) wide memory" (remember – each memory address can store 1 byte of data). Suppose the following 32 bit number have to be stored:

11111111 00000000 10101010 11110000

In case of **big-endian**, the most significant byte is stored in the smallest address. Here's how it would look:

Address	Data value
400	11111111
401	00000000
402	10101010
403	11110000

In case of **little-endian**, the least significant byte is stored in the smallest address. Here's how it would look:

Address	Data value
400	11110000
401	10101010
402	00000000
403	11111111

It can be easily understood by a little thought that "performing arithmetic operations are easier in little-endian computers where as comparing strings chronologically is easier in big-endian computers".

1.3 BASIC ANATOMY OF A COMPUTER

A digital computer is an electronic device that receives data, performs arithmetic and logical operations and produces results according to a predetermined program. It receives data from input devices and gives results to output devices. The central processing unit, also known as processor processes the data. Memory (primary and secondary) is used to store data and instructions. Follows, block diagram of a digital computer identifying the key components and their interconnection.

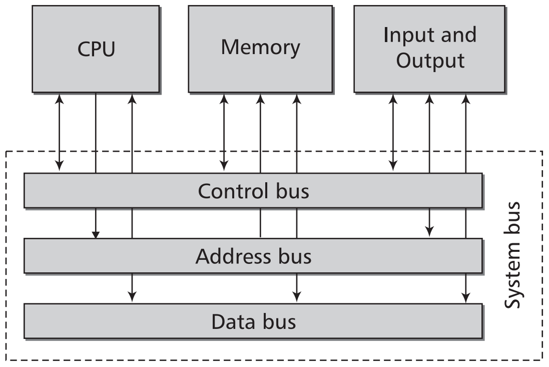


Fig. 1.3 Components of a computer

The Central Processing Unit (CPU) is like the brain of the computer. It is responsible for executing instructions. It controls the sequence of execution of instructions. It comprises a Control Unit (CU), an Arithmetic & Logic Unit (ALU) and huge number of registers. The CU controls the execution of instructions. First it decodes the instruction and then generates micro-operations in a particular order with the help of control memory. The ALU is responsible for performing arithmetic and logic operations.

The interconnections are referred as BUS**.** Buses are nothing but bunch of wires used to carry digital signals. There are three kinds of bus:

1. Address bus
2. Data bus
3. Control bus

Address bus carries address of memory from where to read/to where to write data. Size of address bus of a processor defines the amount of memory addressable by it. For example a processor with 16 bit address bus can access 2^{16} = 64 KB memory (2^6 = 64 and 2^{10} = 1024 = 1 K) and a processor with 32 bit address bus can access 2^{32}=4 GB memory (2^2 = 4 and 2^{30} = 1024 × 1024 × 1024 = 1024 × 1024 × 1 K = 1024 × 1 Mega = 1 Giga.). Address bus is unidirectional, i.e., it carries signal from CPU to other components (only CPU is intelligent enough to generate address).

Data bus is bidirectional. It carries data read from/to be written to a device. Its size and width of registers signify the size of data that can be crunched by the processor in one go.

Control bus carries control signals that activate/deactivate various circuits.

Block diagram of the common bus architecture follows:

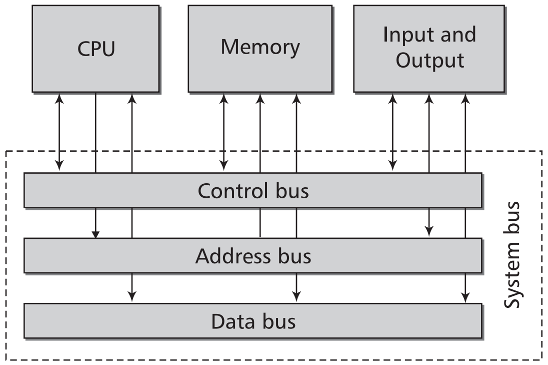


Fig. 1.4 Common bus architecture

1.4 VON NEUMANN ARCHITECTURE

While studying computer architecture, the name of Von Neumann comes first. Though he could not see any working model based on his proposal during his life span, design of all modern computers is based on the stored program model proposed by him. In 1945, the mathematician and physicist John von Neumann, along with others, had described computer architecture in the First Draft Report on the EDVAC. The Von Neumann architecture, which is also known as the Von Neumann model and Princeton architecture, is based on that. In this report a digital computer has been proposed that will contain the following parts:

- A processing unit containing an arithmetic logic unit and processor registers.
- A control unit containing an instruction register and program counter.
- Memory to store both data and instructions.
- External mass storage.
- Input and output mechanisms.

Stored-program computers were much advanced as compared to the program-controlled computers of the 1940s, like the Colossus and the ENIAC. These were programmed by switches and patches, which led to route data and could control the signals between the various functional units.

Some key points to remember about this architecture:

- A program should totally reside in the main memory prior to execution.[*]
- Data and code will reside in the same memory and will be indistinguishable.[†]

[*] As the programs became larger and larger, it was not possible to put the total program in the relatively smaller main memory. So **virtual memory** was introduced in operating systems. It is a technique that shows the free part of secondary storage as main memory. It keeps the total program in the secondary storage in blocks and fetches the required blocks to primary when necessary.

[†] Here **Von Neumann** design differs from **HAVARD ARCHITECTURE** that uses separate code and data memory on separate bus.

- Instructions will be fetched from memory and executed one at a time in a linear fashion.[‡] $$

1.5 MEMORY CLASSIFICATION AND HIERARCHY

Broadly classified, a computer system has two types of memory – Primary and Secondary.

RAM (Random Access Memory), ROM (Read Only Memory) and Cache memory (one kind of very fast random access memory) falls in the primary category where as Magnetic tape, CD-ROM, DVD, Hard disk, pen drive (flash memory) falls in the secondary mass storage category.

Computer system uses memory hierarchy to optimize hardware cost. For example, in a system where 4 megabyte of cache memory is being used, the size of RAM can be 2 GB or 4 GB. If a system is build with total 2 GB cache instead, the system cost will go beyond imagination but 'speed boost' will be only 10% of the system mentioned earlier.

If memory is classified in terms of access strategy then there are three categories:

1. Sequential access.
2. Random access.
3. Direct access.

In **sequential access memor**y, access time is directly proportionate to address. Magnetic tapes falls in this category.

In **random access memory**, access time is constant, i.e., to access content at 1st location or at millionth location, same time will be required.

All sorts of circular and rotating memories like hard disk, CD, DVD fall in **direct access** category where access time can be expressed as a function $T = Ax + By$ where A, B are constants and x, y are variables. These sorts of memory are divided in tracks and sectors. Time required by the read/write head to reach a track is known as **seek time**. Time required by a sector to reach under read/write head is known as **latency time.**

1.6 INPUT AND OUTPUT DEVICES

In this section various input and output devices commonly used in computer systems will be discussed.

1.6.1 Input Devices

1.6.1.1 Keyboard

Text information is entered in the computer by typing on the computer keyboard. Most keyboards, for example – the 101-key US traditional keyboard or the 104-key Windows keyboard, have alphabetic and numeric characters, punctuation marks and function keys. Keyboards are generally connected to the computer by a PS/2 connector or USB port.

[‡] To speed up execution, multiple instructions are fetched in a queue in a pipelined computer.

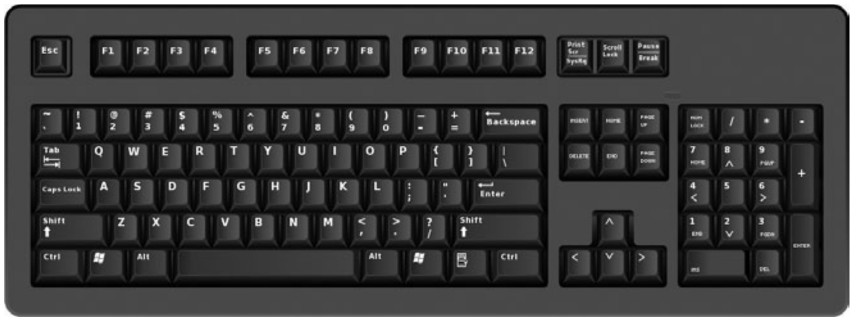


Fig. 1.5 Keyboard

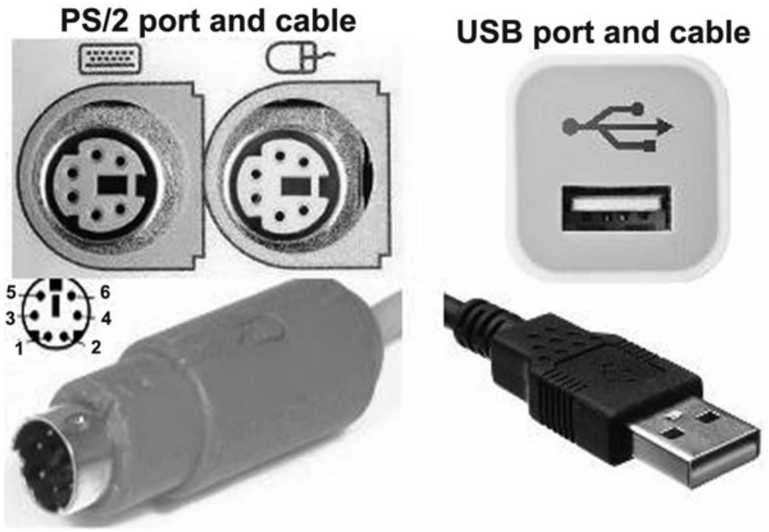


Fig. 1.6 USB and PS/2 ports

1.6.1.2 Mouse

Mouse is a pointing device. The **cursor** on the screen is moved by moving the mouse. A mouse have mainly two buttons – left and right. Newer mouses may contain other buttons and a roller for scrolling. In older models, a ball at the bottom of the mouse rolls on the surface as the mouse moves, and internal rollers sense the movement of the ball and transmit the information to the computer via the mouse cable. The newer optical mouse does not use a rolling ball, instead it uses a light and a small optical sensor to detect the motion of the mouse by tracking a tiny image of the desk surface. Even cordless mouse are very much affordable now a days, but they need regular battery change that reduces their popularity. Mouse sends two information to the computer – one is the X and Y coordinate of the pointer merged in a single number, another is the code of key pressed. Any activity on the mouse generates an interrupt that ultimately communicates an event such as mouse move, left click, double click, right click, and drag to the operating system. What incidents will happen with the mouse events, are completely programmable.

1.6.1.3 Scanners

A scanner is a device that converts a printed page or graphic to an image file that can be stored in a computer by digitizing it. It produces an image made of tiny pixels of different brightness and color values which are represented numerically and sent to the computer.

Fig. 1.7 Scanner

Pages containg text are also scanned and saved as images. To convert such images containing text to editable text files OCR (Optical Character Recognition) software is used.

Scanners are available in various sizes with various scan resolution capabilities. A4 size scanners (largest page size it can scan is A4-8.27″ × 11.69″) are most popular and affordable. Scanners with automatic document feeder are also available.

1.6.1.4 Microphone

A microphone can be attached to a computer through a sound card input or circuitry built into the motherboard to record sound. The sound is digitized and stored in the computer.

Fig. 1.8 Microphone

Our ear can hear analog sound of frequency range 20 to 20 MHz. But for storing sound in computer, the analog signal is converted to digital signal. Again when the sound is played back through speakers, the stored digital signal is converted to analog. (The process of digitization is beyond the scope of this book.)

1.6.2 Output Devices

1.6.2.1 Monitor

Monitor is the maximum used output device of a personal computers. At the begining, CRT- (Cathode Ray Tube) display in monochrome with low resolution was available. Gradually high resolution colour monitors were introduced. Presently bulky CRT monitors have been mostly replaced by LCD or LED monitors as LCD or LED displays consume less electricity, occupy less space and are capable of displaying more clear and flicker free images.

1.6.2.2 Printer

When some output on paper is needed, a printer is a must. The three most common types of printers are Dot matrix, Inkjet and Laser. Dot matrix printers provide poorest quality output at lowest cost. It is mainly used for text based outputs on pre-printed stationary such as cash-memo and ticket.

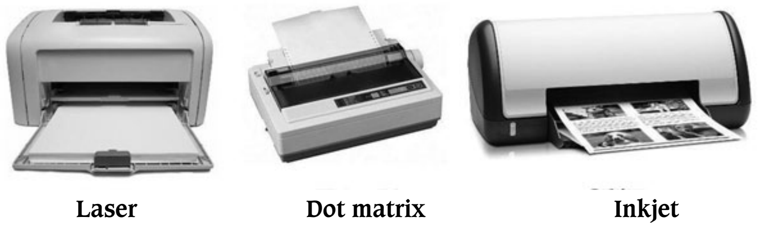


Fig. 1.9 Printers

Laser printers produce best quality printouts using powder formed inks filled in toner cartiges. Cost of laser printers are high but printing cost is less than that of a deskjet printer that uses liquid inks with magnetic particles filled in ink cartiges.

Exercises

1. Write short notes on Von Neumann architecture.
2. Differentiate between Von Neumann and Havard architecture.
3. With a neat block diagram, describe various components on a computer.
4. Classify memories in terms of access strategy.
5. “All ROMs are RAM but all RAMs are not ROMs” – Justify.
6. For a medicine or grocery store what kind of printer you will suggest and why?
7. Write down various ports availble in a computer system.
8. What is a BUS? How many types of BUS is available?
9. Write full form of ENIAC.
10. What do you mean by Virtual Memory. What is the largest possible size of it? Why it is called virtual?

2 Introduction to Number System and Logic Gates

PLAYING WITH NUMBERS

2.1 INTRODUCTION

'Computer' as the name suggests is meant for computation. So knowledge of computer is not possible without the knowledge of various number systems, number formats and their advantages and drawbacks. That is why mathematicians always played a major role in the advancement of computer science.

From pre-historic age when human beings were formed by evolution, numbers are being used to count, label and measure. Initially tally marks were used for counting. Bones, cave pictures from pre-historic age have been discovered with marks cut into them that are similar to tally marks. It is assumed that these tally marks were used for counting elapsed time, such as numbers of days or to keep count of animals. Tally mark was not a positional number system. From tally mark evolved the Roman number system. All these systems have limitations of representing large numbers. Historically it is said that number system with place value was first used in Mesopotamia, from 3400 BC onwards. In India, Aryans used the word SHUNYA in Sanskrit to represent void. This is used as ZERO in mathematics. Greeks were confused about using zero. But eventually this place holder developed today's mostly used *positional number system* where the value of the number depends on the position of the digits. For example $349= 3*10^2 + 4*10^1 + 9*10^0$ as 9 is at position 0, 4 is at position 1 and 3 is at position 2. And obviously the number 1000000 is larger than the number 999999. In fact the value of a number in a positional number system depends on two things, one position and the other is base of that number system.

2.2 BASE OF A NUMBER SYSTEM

Number of digits present in a number system is called **base** of that number system.

In general we use decimal number system, whose base is 10.

EXAMPLE

In **Decimal** base is ***10*** and the digits are: 0,1,2,3,4,5,6,7,8,9.

In **Binary** base is ***2*** and the digits are: 0,1.

In **Octal** base is ***8*** and the digits are: 0,1,2,3,4,5,6,7.

In **Hex**adecimal base is ***16*** and the digits are: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

A-10
B-11
C-12
D-13
E-14
F-15

So in a number system where base is 3, the digits will be 0, 1 and 2.Similarly in a number system where base is 13, the digits will be 0,1,2,3,4,5,6,7,8,9,A,B,C. Remember that the number 187 is not an octal number as the digit 8 cannot be presented in an octal number.

2.3 REASON BEHIND USING BINARY NUMBER SYSTEM

In computer we use binary number system and its variations (whatever may be the interpretation of a value of number, the digits present in a number used in computer are only 0 and 1). This is because today's computers are digital computers made with semiconductor switches that build the logic of the computer. The switches can be in either ON or OFF state (i.e., 0 or 1 state). In other simple word we have to deal with two voltages HIGH and LOW* to work in binary but if we have to design a computer that works in decimal internally we have to deal with 10 different, distinct voltages in the circuit and that is really complicated.

Now question may arise that 'if binary is used in computer then why learn Hex or octal?' the answer is simple: huge binary numbers can be represented in short Hex numbers that are easy to remember and understand by human being. For example the Hex number '3C4F' is equivalent to binary '0011110001001111'.

Again 'why Hex, why not decimal?' – "Converting a number from Binary to Hex or Binary to Octal and vice versa is easier and requires less computation than converting binary to Decimal".

Tip Always remember that all positional number systems sing the same tune. You have to catch the perfect tune to play with them. If the pattern is perfectly recognized then one can handle huge numbers without using pen and paper. For example:

There are 100 numbers from 00 to 99. If we divide them in two equal groups the numbers at the center are 49 and 50.

Now if asked the same question for the set of numbers starting from 00000000 to 99999999 everybody will answer 49999999 and 50000000 without any calculation and just by pattern matching.

*though other voltages than HIGH and LOW exists in digital circuits, better not to bother about them until you start studying VLSI design.

2.4 CONVERSION AMONG DIFFERENT BASES

$$\begin{tikzcd} \boxed{\text{Decimal}} \arrow[r] & \boxed{\text{Other base}} \end{tikzcd}$$

Rule

(i) Divide the number to be converted by target base
(ii) Store the remainder and divide the quotient by target base.
(iii) Continue step II until quotient is 0.
(iv) Write all stored remainders in reverse order to get equivalent result.

As example let us convert the decimal number 37 to different bases:

System	Octal	Hex	Binary
	8 | 37 8 | 4 ----5 0 ----4	16 | 37 16 | 2 ----5 0 ----2	2 | 37 2 | 18 ----1 2 | 9 ----0 2 | 4 ----1 2 | 2 ----0 2 | 1 ----0 0 ----1
Result	45	25	100101

$$\begin{tikzcd} \boxed{\text{Decimal}} & \boxed{\text{Other base}} \arrow[l] \end{tikzcd}$$

Rule Sum up digit multiplied by base to the power position. i.e., $\Sigma \text{digit} * \text{base}^{\text{position}}$

As example let us return back to 37:

System	Octal	Hex	Binary
	$4*8^1+5*8^0$	$2*16^1+5*16^0$	$1*2^5+1*2^2+1*2^0$
Result	37	37	37

Tip You do not have to follow the rule in order to convert from binary to decimal. Many easy shortcut methods are available. Before stating them, please mug up following tables thoroughly so that you can recite the values if asked randomly.

Table 2.1 Powers of 2

2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Table 2.2 Binary values of decimal numbers

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

*Remember that **"if we multiply a number by base a '0' is added at the end"** 35*10=350, 35*10^2=3500, 35*10^5=3500000 and so on.*

Similarly in binary, notice that

Binary of 3= 11

Binary of 6= 110

Binary of 12= 1100

Binary of 24= 11000

Similarly in Octal

Octal of 2 = 2

Octal of 16 = 20

Octal of 128 = 200

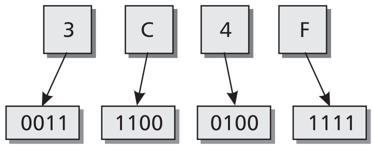
Now you can use any of the following tricks for conversion

- *Notice that if you remember binary of 13 then you can say binary of 26, 52, 104 and so on.*
- *Let us take a larger example – Decimal equivalent of the binary number 11001110 is 206. As the number 11001110 = 11000000 + 1110 = 3*2^6 + 14= 192 + 14 = 206.*
- *Adding up power of 2 at the positions where there is 1 in the binary number gives equivalent decimal easily.*
- *128 = 2^7 so its binary will be 10000000 (7 zeros) hence binary of 127 will be 1111111 (7 ones) and binary of 129 will be 10000001*

Just write equivalent binary of the digits of the Hex number side by side using 4 bits for each Hex digit.

EXAMPLE

$3C4F_H$ = 0011110001001111

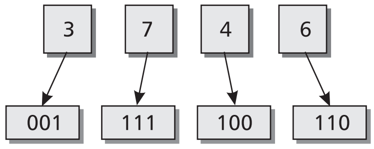


OCTAL
Binary

Just write equivalent binary of the digits of the octal number side by side using 3 bits for each octal digit.

EXAMPLE

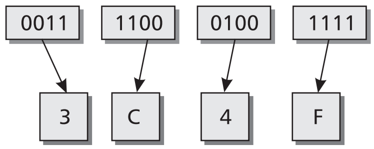
3746_8 = 011111100110



Group the digits of the binary number in four, taking from right. If required take extra 0 at left. Now write hex equivalent for each group.

EXAMPLE

11110001001111=$3C4F_H$

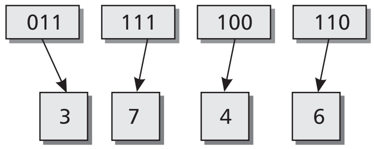


OCTAL
Binary

Group the digits of the binary number in three, taking from right. If required take extra 0 at left. Now write hex equivalent for each group.

EXAMPLE

$11111100110 = 3746_8$



2.5 SIMILARITIES BETWEEN NUMBER SYSTEMS OF DIFFERENT BASES

I would like to repeat again the sentence – *"all positional number systems sing the same tune"*. From the following table the similarities will be clear.

Table 2.3 Similarity of different bases

	Decimal	Octal	Hex	Binary
Highest number using 3 digits	$10^3-1=999$	$8^3-1=777$	$16^3-1=FFF$	$2^3-1=111$
Value of 1000−1=?	999	777	FFF	111
Value of base	10=10	8=10	16=10	2=10
Subtracting 1	30−1=29	30−1=27	30−1=2F	11110−1=11101
Multiply by power of base	$37*10^3=37000$	$37*8^3=37000$	$37*16^3=37000$	$100101*2^3=100101000$

2.6 ADDITION OF TWO NUMBERS

Before stating the addition process, let us know two terms MSB and LSB. The leftmost digit of a number is called MSB (Most Significant Bit) and the rightmost digit of a number is called LSB (Least Significant Bit).

Now let us recap the addition process of two number, we are doing since our nursery classes:

- We start addition from adding LSB of two numbers.
- We divide the result by base to obtain quotient and remainder.
- We write remainder as sum and quotient goes to carry to next digit addition.

Pictorially it can be shown as:

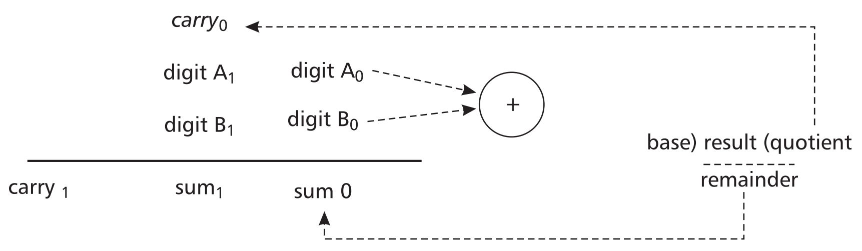


Fig. 2.1 Addition process

EXAMPLE

DECIMAL	STEP	RESULT (in decimal)	Quotient	Remainder
35	1(5+7)	12	1	2
67	2(3+6+1)	10	1	0
102				

In the above example the LSB of 35 is 5 and LSB of 67 is 7. So we first add them and get 12. We then divide 12 by base, i.e., 10 here to get 2 as remainder and 1 as quotient. So 2 is sum and 1 is carry. In the next step, we add 3, 6 and carry 1 from the previous addition to get 10. We repeat same process of division to get 0 as sum and 1 as carry. As there are no more digits left, the carry 1 will be written at the MSB.

Similarly addition of two numbers in different bases follows.

OCTAL(8)	STEP	RESULT (in decimal)	Quotient	Remainder
35	1(5+7)	12	1	4
67	2(3+6+1)	10	1	2
124				

HEX(16)	STEP	RESULT (in decimal)	Quotient	Remainder
3C	1(12+13)	25	1	9
6D	2(3+6+1)	10	0	A
A9				

BINARY(2)	STEP	RESULT (in decimal)	Quotient	Remainder
11	1	2	1	0
11	2	3	1	1
110				

Note that same procedure has been followed for all number systems. Only difference is that the division was done by corresponding base, e.g., 8 for Octal, 16 for Hexadecimal and so on.

We can add multiple binary numbers at a time in same process:

BINARY(2)	STEP	RESULT (in decimal)	Quotient	Remainder
110	1	4	2	0
101	2	5	2	1
111	3	7	3 (11 in binary)	1
011				
101				
100				
11110				

2.7 SIGNED BINARY NUMBERS

So far we were talking about unsigned numbers having only magnitude. In this section, we will see how to represent negative numbers in binary. The main hurdle is that we cannot store the symbol (–) minus in computers. We have only 0(zero) and 1(one) to use. There are many possible ways:

(A) **Signed magnitude form:** We can use the MSB as sign. For positive numbers we can use MSB = 0 and use MSB = 1 for negative numbers. Remaining bits used to store the magnitude. For example, in an 8 bit representation +5 = 00000101 and –5 = 10000101.

(B) **1′s complement method:** To get the **1′s (**one′s) complement of a binary number we just have to invert all the bits in the binary representation of the number (replace 0 by 1 and 1 by 0). The 1′s complement of the number then behaves like the negative of the original number. If 1′s complement of a number is calculated we get the original number. This is like –(–5) = +5. Also if we add 1′s complement of 2 with binary of 5 we get binary of 3 showing that 1′s complement of 2 is behaving like –2.

(C) **2′s complement method:** The two′s complement of an N-digit binary number is defined as the complement with respect to 2^N, i.e., the result obtained by subtracting the number from 2^N. **Taking the 1′s complement of the number and then adding 1 with the complement will give same result.** It can be easily verified that 2′s complement of a number is the original number as in the case of 1′s complement. (*Please note that any extra bit after MSB in 2′s complement system is simple discarded.*)

Though both 1′s complement and 2′s complement system behaves almost same still 2′s complement is mostly used for the two differences:

(i) In 1′s complement value of 0 and –0 is different (0000 vs. 1111) but in 2′s complement both zero and minus zero have same value, which is more realistic.

(ii) If we add P with –P in 1′s complement we get –0 but in 2′s complement we get 0. e.g. 5+(–5) → 0101 + 1010 = 1111 in 1′s complement but 5+(–5) → 0101 + 1011 = 0000 in 2′s complement.

While dealing with 2′s complement numbers do not forget the range of numbers that can be represented using N bits. An *N*-bit 2′s-complement number system can represent every integer in the range $-(2^{N-1})$ to $+(2^{N-1} - 1)$. Please refer to following table for the ranges.

Number of bits	Range in 2′s complement	Range in unsigned representation
3	−4 to 3	0 to 7
4	−8 to 7	0 to 15
5	−16 to 15	0 to 31
8	−128 to 127	0 to 255
10	−512 to 511	0 to 1023
N	$-(2^{N-1})$ to $(2^{N-1} - 1)$	0 to $(2^{N} - 1)$
Range of numbers using N bits		

Some examples of addition and subtraction using 2′s complement system are given for ease of understanding.

i)

23–8=		23 =	00010111		23 =	00010111		
23+(–8)	→	+8 =	00001000	→	–8 =	+11111000		

						=00001111	→	+15

ii)

8–23= 8+(-		8 =	00001000		8 =	00001000		
23)	→	+23 =	00010111	→	–23 =	+11101001		

						=11110001	→	–15

iii)

–8+23=		8 =	00001000	→	–8 =	11111000		
	→	+23 =	00010111		+23 =	+00010111		

						=00001111	→	+15

iv)

8+23=		8 =	00001000	→	8 =	00001000		
	→	+23 =	00010111		+23 =	+00010111		

						=00011111	→	+31

2.7.1 BCD (Binary Coded Decimal) Number

Binary-coded decimal (BCD) is a binary encoding of decimal numbers where each decimal digit is represented by a fixed number of binary bits (i.e., 0/1), usually four or eight.

For example BCD of the number 243 is 0010 0100 0011 where four bits are used for each digit.

Although BCD was used in many early decimal computers, now-a-days it is rarely used.

If 4 digits are used for every digit then it is called **packed BCD.**

If 8 digits are used for every digit then it is called **unpacked BCD.**

2.7.2 Advantage of BCD (Over Binary Number System)

- It gives more accurate representation and rounding of decimal quantities.
- Can be easily converted into human-readable representations.

2.7.3 Drawbacks of BCD

- Increase in the complexity of the circuits needed to implement basic arithmetic.
- More storage space required.

2.7.4 Addition of Two BCD Numbers

(i) First add two 4-bit BCD digits using normal binary addition.

(ii) If the 4-bit sum is less or equal to 9, the sum is in proper BCD and no correction is required but If the sum is greater than 9 or if a carry is generated from the sum then add 6 (0110) with the sum to get proper BCD result. If in this adjustment process a carry is generated then it will be added as carry to the next decimal place.

2.7.5 Excess-3 or XS-3 Code

The Excess-3 code of a decimal number is calculated by adding 3 (three) to each decimal digit of the given number and then replacing each digit of the newly generated decimal number by its four bit binary equivalent.

It is a biased BCD code. It was used on few older computers as well as in cash registers and old hand held portable electronic calculators of earlier days.

Number	0	1	2	3	4
BCD	0000	0001	0010	0011	0100
Excess-3	0011	0100	0101	0110	0111

Number	9	8	7	6	5
BCD	1001	1000	0111	0110	0101
Excess-3	1100	1011	1010	1001	1000

It is an example of **unweighted code** (as each position within the binary equivalent of the number is not assigned a fixed value).

Please note from the above table that excess-3 is self complimenting code or reflective code, as 1′s compliment of any number (0–9) is available within these 10 numbers. For example 1′s complement of 9 (1100) is 0011 which is excess-3 code of 0. Same phenomenon can be watched for 1–8, 2–7, 3–4 and 4–5.

The primary advantage of XS-3 coding over non-biased coding is that a decimal number can be nines' complemented (for subtraction) as easily as a binary number can be ones' complemented; just invert all bits.

2.8 ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

The American Standard Code for Information Interchange (ASCII) is a character-encoding scheme originally based on the English alphabet that encodes 128 specified characters into the 7-bit binary integers. They can be classified as:

- The numbers 0–9.
- The letters a–z.
- The letters A–Z.
- Some basic punctuation symbols.
- Some control codes that originated with teletype machines.
- A blank space.

ASCII includes definitions for 128 characters. Among them there are 33 non-printing control characters that affect how text and space are processed and 95 printable characters, including the space.

Part of the ASCII table is given below:

Table 2.4 ASCII values

Character	Value			
	Dec	Hex	Binary	Oct
Null character	0	00	000 0000	000
Bell	7	07	000 0111	007
Backspace	8	08	000 1000	010
Horizontal tab	9	09	000 1001	011
Line feed	10	0A	000 1010	012
Carriage return	13	0D	000 1101	015
Escape	27	1B	001 1011	033
Space	32	20	0100000	040
*	42	2A	0101010	052
+	43	2B	0101011	053
−	45	2D	0101101	055
/	47	2F	0101111	057
0	48	30	0110000	060
9	57	39	0111001	071
A	65	41	1000001	101
Z	90	5A	1011010	132
a	97	61	1100001	141
z	122	7A	1111010	172

2.9 LOGIC GATES AND BOOLEAN ALGEBRA

All digital circuits can be classified into two broad categories:

1. Combinational Circuits
2. Sequential Circuits.

2.9.1 Combinational Circuits (Combinatorial Logic Circuits)

The digital circuits where the output(s) is/are totally dependent on the present input(s) are called combinational circuits.

When input(s) is/are changed, the information about the previous input(s) is lost. So we say that combinational logic circuits have no memory. This is in contrast to sequential logic, in which the output depends not only on the present input but also on the previous output.

In this chapter, we will concentrate on combinational circuits.

2.9.2 Sequential Circuits

The digital circuits in which the output depends both on the present input and the history of the input (previous output) are called sequential circuits. In other words, sequential logic has memory while combinational logic does not.

Sequential circuit can be described by following block diagram:

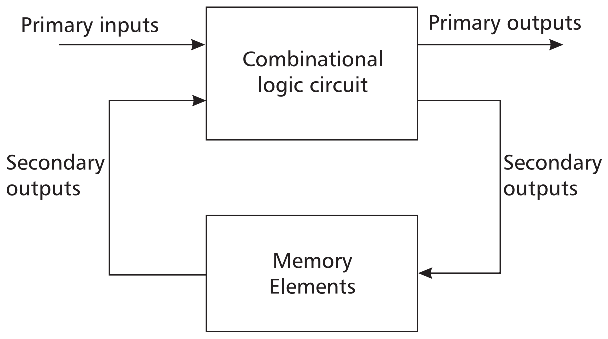


Fig. 2.2 Sequential circuit

2.10 GATES (LOGIC GATES)

Logic gates are combinational circuits having one or more input but only one output.

A logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic performed is Boolean logic. Logic gates are implemented electronically using diodes or transistors.

2.11 BASIC GATES

AND gate OR gate NOT gate – These three are basic gates and we can create any Boolean logic circuit using combination of these three gates.

Truth Table: A truth table is a table that describes the behavior of a logic gate. It lists the value of the output for every possible combination of the inputs.

Boolean algebra is different from ordinary algebra in one major place. Boolean constants and variables can have only two possible values, 0 or 1.

Boolean 0 and 1 are not actual numbers, on the contrary they represent the state of a voltage variable, or what is called its logic level.

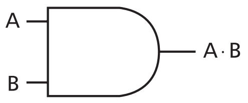
Common representation of 0 and 1 is:

- Logic 0: False, Off, Low, No, Open Switch
- Logic 1: True, On, High, Yes, Close Switch

2.11.1 AND Gate

AND gate is a logic gates having two or more inputs whose output is 1 if all of its inputs are 1 otherwise the output is 0.

AND gate is represented by the following block diagram:



If the inputs are a, b then the output is written as **a.b** or simply **ab (so '.' is the symbol of AND in Boolean expression).**

Truth table of a two-input and three-input AND gate follows

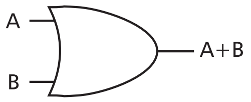
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

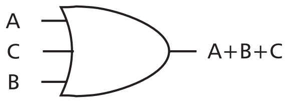
A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

2.11.2 OR Gate

OR gate is a logic gates having two or more inputs whose output is 0 if all of its inputs are 0 otherwise the output is 1.

OR gate is represented by the following block diagram:





If the inputs are a, b then the output is written as **a+b (so '+' is the symbol of OR in Boolean expression).**

Truth table of a two input and three input OR gate follows:

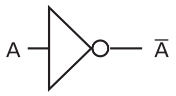
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

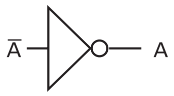
A	B	C	A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

2.11.3 NOT Gate

NOT gate can have only one input. The output is just reverse of the input. i.e., output is 1 if input is 0 & vice-versa.

NOT gate is represented by the following block diagram (also in composite circuit a bubble implies NOT).

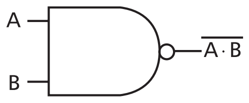




The output is denoted by a bar over the input. If the input is A then output will be $\overline{A}$, if the input is $\overline{A}$ then the output will be A (double bar cancels).

If the input is A+B+C then the output will be $\overline{A+B+C}$ not $\overline{A} + \overline{B} + \overline{C}$, they are different. NOT gate is also known as **Astable Multivibrator**.

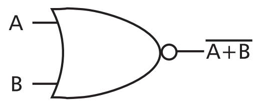
NAND gate Can be considered as NOT of AND. For inputs A, B the output will be $\overline{AB}$. Block diagram follows:



REMEMBER THE FOLLOWING IDENTITIES:

$A+A=A$
$A.A=A$
$A+1=1$
$A.1=A$
$A+0=A$
$A.0=0$
$A+\overline{A}=1$
$A.\ \overline{A}=0$

NOR gate Can be considered as NOT of OR. For inputs A, B the output will be $\overline{A+B}$. Block diagram follows:



2.12 UNIVERSAL GATES

NAND and NOR are known as Universal Gates as by combining multiple NAND or NOR gates we can construct any Boolean logic without using any other gate.

Construction of three basic gates using NAND gate follows that sufficiently proves that NAND is a Universal Gate. (NOR is a Universal Gate-proof left to reader).

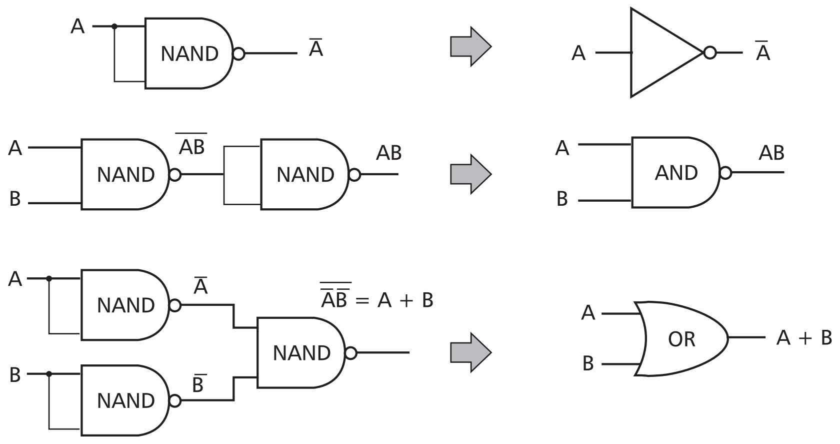


Fig. 2.3 NAND as universal gate

De Morgan's Laws:

1. $\overline{A+B} = \overline{A} \cdot \overline{B}$ (Proof by truth table is left for the reader)
2. $\overline{A} + \overline{B} = \overline{A \cdot B}$

Distributive Law

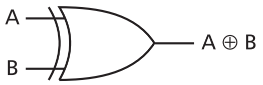
A+B.C = (A+B).(A+C)

Proof

$$\begin{aligned}(A+B).(A+C) &= A\cdot A + A\cdot B + A.C + B\cdot C \\ &= A + A\cdot B + A\cdot C + B\cdot C \\ &= A\cdot(1+B+C) + B\cdot C \\ &= A\cdot 1 + B\cdot C \\ &= A + B\cdot C\end{aligned}$$

XOR (Exclusive OR gate) It may have two or more inputs. The output is 1 if **odd number of input is 1** otherwise 0.

Denoted by $A \oplus B$ and the block diagram follows:

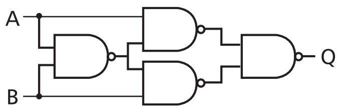


Truth table of a two input and three input XOR gate follows:

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	C	A⊕B⊕C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The XOR gate with inputs *A* and *B* implements the logical expression $\overline{A}B + A\,\overline{B}$.



XOR gate constructed using only NAND gates

EXERCISES

1. Convert the Hexadecimal number 3F1D.3AB to Octal.
2. What is the advantage of 2's complement number over 1's complement.
3. Add the following numbers using 2's complement:
   - **(a)** 5, 3
   - **(b)** 5, 3
   - **(c)** 5, 3
   - **(d)** 16, 3
4. Which gates are called UNIVERSAL gates and why?
5. Draw circuit for $AB(\overline{A} + B)(A+\overline{B})$
6. Add the binary numbers: 100, 100, 110,011,111, 101.
7. Convert decimal 379 to Octal and Hex.
8. How to get output in POS form from Karnaugh map?
9. Write De Morgan's law.
10. Write distributive law.

3 Introduction to System Software and Operating Systems

3.1 INTRODUCTION TO ASSEMBLER

Initially when computer was invented, people used to communicate with them by on and off switches denoting primitive instructions. Then machine language was introduced, where programmer have to mention instructions as well as operands and addresses in binary format, i.e., using 0 and 1. Both writing and understanding a machine language program is difficult. The functionality of this program is difficult to understand, and a person going through it may not be sure of what can be achieved through the program.
Assembly language program is one in which symbols such as letters, digits and special characters, are used for operation part, address part and other parts of instruction code. Both machine language and assembly language are referred as low level languages as the coding a problem is at the individual instruction level, i.e., for each line of a program written in these languages one and only one machine instruction is executed. Assembly language is considered as a second generation language. Computers based on different processors have got their own assembly languages which depend on the architecture of the processor. An assembler translates a program written in assembly language to machine language code.

Output of an assembler is an object file with extension '.obj'. It is the role of linker (another system-software) to make executable .exe file from it. Another system-software called loader, present in operating systems, loads the executable on memory when the program is required to run. The process is illustrated in Fig. 3.1. Process for producing executable file.

Typically, assemblers make two passes over the assembly file, i.e., reads the assembly program twice. (One pass assemblers with more complicated design are also available.)

In the first pass, it reads each line and records labels in a symbol table and in second pass, use info in symbol table to produce actual machine code for each line.

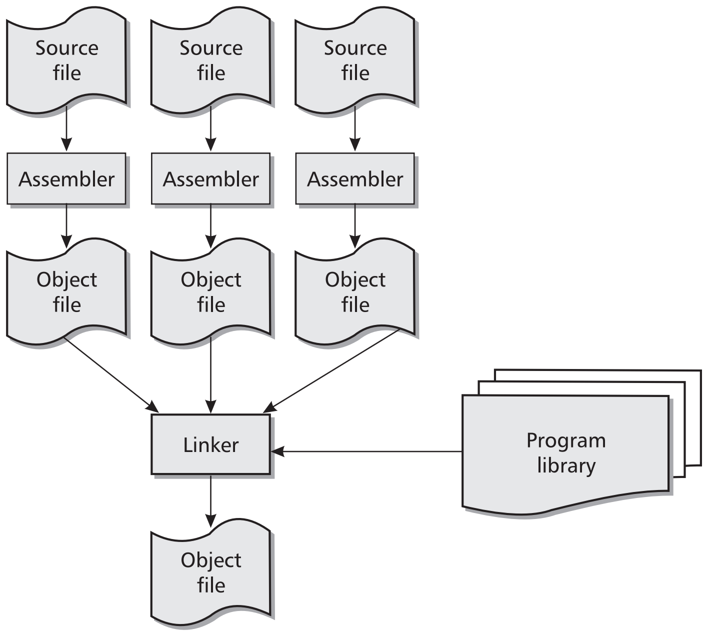
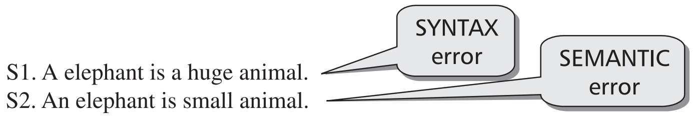


Fig. 3.1 Process for producing executable file

3.2 INTRODUCTION TO COMPILER


S1. A elephant is a huge animal.
S2. An elephant is small animal.
SYNTAX error
SEMANTIC error

Carefully read the above two sentences. The sentence S1 has grammatical mistake as before 'elephant' we have to use 'An' not 'A'. This type of error is called syntax error. The second sentence S2 is grammatically correct but still sounds wrong as an elephant is not at all a small animal. This sort of error is known as semantic error or logical error.

A compiler checks a file containing a program or part of a program (written in high level language) only for syntax errors. However it is the programmers' responsibility to keep the program free from semantic errors. Semantic errors appearing in a program are removed by the programmers by means of debugging and various testing methodologies that are not our matter of interest here.

We have to remember that compiler always compiles file wise, i.e., if a program is spanned over 10 separate files, we have to compile 10 files separately and we will get 10 object files. Error in one file will not affect successful compilation of another file. After successful compilation of all the files, system software called **linker** creates a single executable file of the program from 10 object files. So it is the role of a **linker** to create executable file from object file.

Before questioning that 'how a compiler detects grammatical errors?' we in a nutshell will learn 'what is a grammar?'

3.2.1 Formal Definition of Grammar

A grammar G consists of the following components:

- A finite set N of non terminal symbols, none of which appear in strings formed from G.
- A finite set (Sigma) of terminal symbols that is disjoint from N.
- A finite set P of production rules.
- A distinguished symbol S as start symbol ($S \in N$).

The definition can be explained by an example of English grammar where

- ∑(Sigma) is the set of all English words.
- N is the set of non terminals that correspond to the structural components in an English sentence, such as <SENTENCE>, <SUBJECT>, <PREDICATE>, <NOUN>, <VERB>, <ARTICLE>, and so on.
- The start symbol would be <SENTENCE>.
- And we have the following production rules:
  - <SENTENCE> → <SUBJECT><PREDICATE>
  - <SUBJECT> → <NOUN>
  - <PREDICATE> → <VERB><NOUN>
  - <NOUN> → Ram
  - <NOUN> → Sam
  - <NOUN> → Jadu
  - <NOUN> → flower
  - <VERB> → loves

Now consider following three strings to check whether they are sentence or not

(i) Ram loves flower.
(ii) Sam loves flower.
(iii) Jadu flower loves.

Using the grammar defined above we can reach start symbol (goal symbol) <SENTENCE> from the first string in the following way:

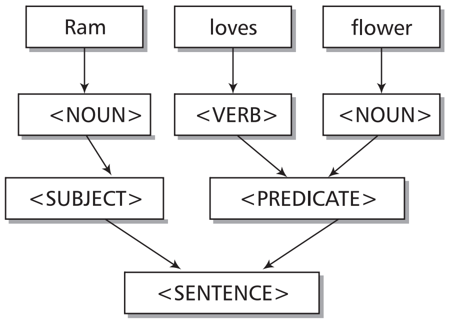


Fig. 3.2 Parse tree of a sentence

Similarly sentence can be formed from the second string but from the third string start symbol <SENTENCE> cannot be reached from the above defined grammar. (Readers having Bengali, Hindi or other Sanskrit based languages as mother tongue notice that in the grammar of their mother tongue a VERB usually comes after NOUN in the PREDICATE.)

Compiler does the same job as discussed above. It tries to reach the goal symbol of the high level language it is meant for, from the program written in the file it compiles using the grammar defined for the language. If it fails then it reports error. Though detail of a compiler is out of scope of this book, in the following section we will learn the different components of a compiler and their role.

3.2.2 Components of a Compiler

A compiler may have some or all of the following components:

(a) Lexical Analyzer or Lex It scans the source code as a stream of characters and breaks it into bunch of meaningful pieces called tokens. For identifying keywords or variables, it takes help of a symbol table.

(b) Parser Using the tokens supplied by the lex, parser tries to build a parse tree (Fig. 3.1) whose root will be the goal symbol of the language. If it succeeds then the program is free of syntax error and is passed to the next phase.

(c) Intermediate code generation The compiler generates an intermediate code of the source code for the target machine. The source program is converted to a program meant for an abstract machine. It is in between the high-level language and the machine language. The intermediate code is generated in such a way that it makes it easier to be translated into the target machine code.

(d) Code optimization In this phase, the intermediate code is optimized for faster execution. Some unnecessary lines are removed and code is sometimes rearranged for speedup and better memory management.

(e) Code generation The code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into re-locatable machine code.

It should be noted that programs written in some high level languages are translated and executed by another kind of software called **INTERPRETERS.**

The basic difference between Interpreters and Compilers follows:

- Compiler takes entire program as input whereas interpreter takes single instruction as input, i.e., translates the source program line by line.
- Program needs to be compiled once but in case of interpreter, it converts higher level program into lower level program for every execution.
- In case of compiler, errors are displayed after the total program is checked. On the contrary, the interpreter displays error (if any) for every instruction interpreted.

3.3 INTRODUCTION TO OPERATING SYSTEM

Operating system is a system-software that acts as an interface between the user and the computer. It is a supervisory program that manages all sorts of hardware resources such as CPU, memory, disk and I/O devices. It also provides common services to computer programs. From the time of starting the computer till shutting it down, operating system resides in memory and can take commands from the user.

The digital computer was in its nascent stage in the 1940s and there were no operating systems. Machines of the time were in preliminary stage and programs could generally be entered one bit at a time on rows of machine switches. No programming languages, including assembly languages, were known at the time. The General Motors Research Laboratories implemented the first operating systems in early 1950s for their IBM 701. Various computer specific operating systems were developed till 1980 but never been used by mass audience. After that two operating systems have dominated the personal computer scene: MS-DOS, developed by Microsoft, for the IBM PC and other machines using the Intel 8088/8086 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family. Both of them were character based. User has to give input through keyboard and output was on the monitor displaying characters only. Then Graphical User Interface based OS such as Windows, GUI based UNIX and MAC OS came to make the computer acceptable to common people.

In the following section, brief description of DOS, UNIX and Windows are given:

3.3.1 MS-DOS

Microsoft Corporation wrote the version of DOS entirely in 8086 assembly language that IBM chose for its first line of personal computers. DOS is an acronym for the computer's 'Disk Operating System' program. It is named so as at that time it is the only small operating system that fits in a single floppy disk. Some information worth remembering about DOS follows:

- It requires three files – io.sys, msdos.sys and command.com to boot a system.
- It accepts commands from user at the command prompt (a command prompt looks like c:\> with a blinking _ called cursor).
- It allows only one program to run at a time.
- It supports two types of commands internal and external. Internal commands are inbuilt in 'command.com' and require no other file.
- It supports tree like hierarchical directory structure with '8.3' file name format.

Some useful DOS internal commands and their usage follows:

Table 3.1 Some DOS commands

Command	Purpose
MD	Make directory
CD	Change directory
RD	Remove directory
CLS	Clears screen
DATE	Show/set system date
TIME	Show/set system time
COPY	Copy one file to another or to console
TYPE	Displays content of file on screen
DEL	Deletes a file
DIR	Shows the content of a directory
PATH	Show/set current path
VOL	Shows volume label of a disk

3.3.2 Microsoft Windows™

Presently Microsoft Windows refers to the set of operating systems developed by Microsoft Corporation. In 1985, Microsoft ships Windows 1.0 a GUI operating system shell that runs under DOS. Then, rather than typing MS–DOS commands, moving a mouse to point and click way through screens, or 'windows' were possible. Drop-down menus, scroll bars, icons, dialog boxes and many other visual items were introduced in WINDOWS that make programs easier to learn and use. Switching among several programs without having to quit and restart each one was possible. Microsoft shipped Windows 1.0 with many utility programs like MS-DOS file manager, Paint – a drawing utility, Notepad to create and edit text files, Calculator, and a calendar, card file, and clock to help manage day-to-day activities. There was even a game – Reversi. Since its launch, Windows dominated the personal computer market. Windows 2.0, 2.1 3.0 and 3.11 were released and used till 1994. All of them were 16 bit and was a program that runs under MS-DOS as a shell. In 1995, Windows 95 was introduced, that was 32 bit version and was independent OS not a DOS shell. Seven million copies of Windows 95 were sold in the first month.

After Windows 95 there were several versions such as 98, XP, NT, Server 2000 and 2003, Windows CE, Vista, Windows 7 and 8 then presently Windows 10.

Beside support to old file allocation table of disk 'FAT' used in DOS, WINDOWS comes with two other file allocation tables named FAT32 and NTFS to support larger disks. NTFS also provides user access control to files and directory that increases security to disk contents.

Now a days, internet has become an integral part of daily life. To browse the content of a web page from internet, a software called browser is required. WINDOWS comes with a browser Internet Explorer built-in.

A program runs in a rectangular area called window in the WINDOWS operating system. Elements that may be present in a window are shown in the Fig 3.3 a sample window.

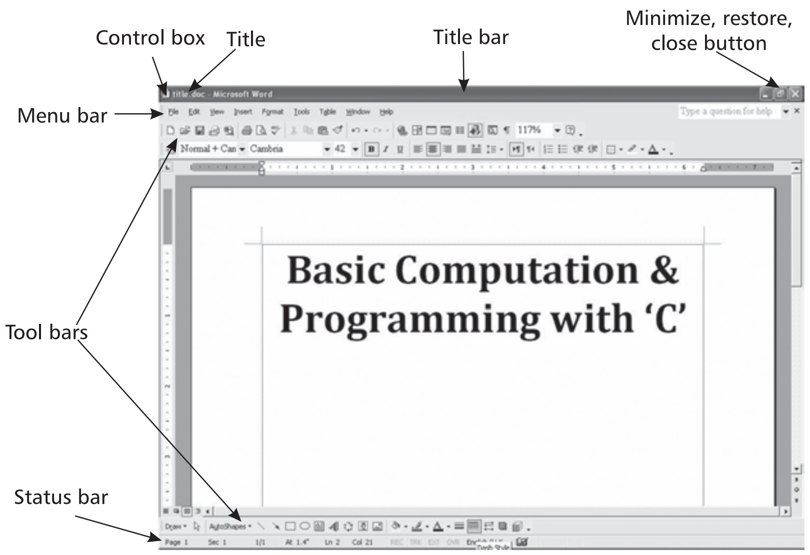

Control box
Title
Title bar
Minimize, restore, close button
Menu bar
Tool bars
Status bar
title.doc - Microsoft Word
Basic Computation & Programming with 'C'

Fig. 3.3 A sample window

3.3.3 UNIX

In the year 1970, Ken Thompson, Dennis Ritchie and others developed UNIX which is a family of multitasking, multiuser operating system. It was designed to be used in the Bell labs where it was developed. Later American Telephone and Telegraph Company (AT&T) licensed it for outside parties. In a command-line operating system a text command is entered and the computer responds according to that command. With a graphical user interface (GUI) operating system, user interacts with the computer through a graphical interface with pictures and buttons by using the mouse and keyboard. Presently with UNIX both command-line (more control and flexibility) and GUI (easier) are available.

Follows some useful UNIX commands: (Please note that UNIX commands are case sensitive but DOS commands are not.)

Table 3.2 Some UNIX commands

Command	Purpose
cat	Display File Contents
cd	Changes Directory
chgrp	change file group
Chown	Change owner
compress	Compress files
Cp	Copy source file into destination
File	Determine file type
find	Find files
ftp	File transfer program
ftp	File transfer program
info	Displays command information pages online
kill	Send a signal to a process
logout	log off UNIX
ls	Display information about file type.
man	Displays manual pages online
mkdir	Create a new directory
mv	Move (Rename) an old name to new name
ps	Display the status of current processes
pwd	Print current working directory
rcp	Remote file copy
rm	Remove (Delete) filename
rmdir	Delete an existing directory provided it is empty
sort	Sort file data
uncompress	Uncompress files
uniq	Report repeated lines in a file

Command	Purpose
uniq	Report repeated lines in a file
vi	Opens vi text editor
wc	Count words, lines, and characters
who	List logged in users

Introduction to UNIX will be incomplete without knowledge of **UNIX shell**. To access the services of a kernel, users of an operating system have to use an interface. The software that provides that interface is called a shell. The UNIX shell, similarly, is a shell or command line interpreter that provides user interface for the UNIX or UNIX-like operating systems. Operation of the computer is directed by the user by entering commands as text, which is then executed by a command line interpreter. UNIX shells are of four types:

- Bourne shell (sh): Developed by Stephen Bourne at Bell Labs in 1974; Default prompt is $.
- C shell (csh): Developed by Bill Joy at the University of California at Berkeley in 1978; Default prompt is %.
- Korn shell(ksh): Developed by David Korn at Bell Labs in 1983;
- Bourne – again shell (bash): Default shell on Linux operating system; Runs on UNIX-like operating system and also available for Microsoft Windows systems.

In this section, a brief idea about the well known **vi editor** is given. Unix offers an editor named vi, which is a screen oriented text editor written by Bill Joy in the year 1976. vi editor is most suitable way to create a file because some operations like line change, caps lock, underline can be performed through this which could not be performed by the cat command. vi has three types of mode, Input mode (for entering text), Command mode (every key pressed is interpreted as a command to run on text) and the Last line mode or ex mode (commands can be entered in the last line of the screen to act on text).

Idea of the **UNIX file system** is provided in the following section. UNIX supports two basic objects, namely file and processes. A file is basically a store house of information that is represented as a sequence of bytes in the system. UNIX filenames can be up to 255 characters. A process is such a file that is being executed by the system, so it is a popular concept that UNIX treats everything as a file. File s in UNIX can be categorized into three sectors:

- Ordinary file or Regular file (Ex: text file)
- Directory file (Ex: any directory)
- Device file (Ex: Printer, Floppy drive, CD-ROMs)

Every file is associated with a table, which contain all information about the file except its name and content, this table is known as inode (index node) and is accessed by the inode number. A file has several attributes that are modifiable by some well-defined rules. "ls – l" command is used to display the file attributes.

UNIX files are integrated in a single directory structure. The file system is arranged in a structure like inverted tree (or hierarchical), i.e., Top of the tree is the root and is written as '/'(forward slash). root(/) is actually a directory file and it has all the sub-directories of the system under it.

Some features that are worth remembering about UNIX file system follows:

1. root directory(/) has no parent, every other has parent.
2. Directory contain the name of the file and the corresponding inode no.
3. root directory has an inode no. 2. inode no. 0 and 1 are not used.
4. The size of the directory is small because it contains only the name of the file and the inode no. It does not contain the data of the file.
5. Basically directory is the mount point consisting of files and file related information.
6. Mount means attached, so swap space has no mount point because it is a raw partition that is used by any file system for swapping.
7. User can make logical partition into a directory and mount it.
8. If user unmounts any directory means the pointer, i.e., the mount point is hidden.

In UNIX, a file is treated as a normal file if it resides in dormant state on the disk but when it is executed it is treated as a process. A process has several attributes among them two are important:

- PID: process id
- PPID: parent process id

UNIX operating system required three system calls for creation of process:

- fork(): Creates a new process
- exec(): For running the process
- Wait(): Executes by parent process to wait for the child process to complete.

In the process hierarchy, init is the first process whose PID is 1, sched is the parent of init whose PID is 0. UNIX is a popular operating system which is capable of handling activities from multiple users at the same time.

EXERCISES

1. After executing the command 'cd subhodip' at DOS prompt 'c:\book>' how the command prompt will look like?
2. What do you mean by 8.3 naming convention of files?
3. What you can do with the control box of a window?
4. What is the difference between one pass and two pass assembler?
5. What is full form of GUI?
6. What is a shell in UNIX?
7. Which commands are used to display contents of a directory in DOS and UNIX?
8. List different file types available in UNIX.
9. Where UNIX was developed?

4 Algorithms and Flow Chart

4.1 FLOW CHART

A flowchart is a type of diagram that represents an algorithm or process. The steps involved in the process are shown as boxes of various kinds. Arrows that connects the boxes depicts the correct order in which the steps should be performed. This diagrammatic representation illustrates a solution to a given problem. As a diagram is equivalent to 1000 words, flowcharts helps people working in a project to understand the problem very easily.

Flowcharts are broadly classified into two categories:

1. Program Flowchart
2. System Flowchart.

Program flowcharts are symbolic or graphical representation of computer programs in terms of flowcharting symbols. They contain the steps of solving a problem unit for a specific result; **System flowcharts**, on the other hand, contain solution of many problem units together that are closely related to each other and interact with each other to achieve a goal.

In this book, concentration will be on program flowcharts as this book is mainly on a programming language. A program flowchart is an extremely useful tool in program development activity in the following respects:

1. Any error, omission or commission can be easily detected from a program flowchart than it can be from a program.
2. A program flowchart can be followed easily and quickly.
3. It can be referred if program modifications are needed in future.

Follows a program flowchart to 'find maximum among three numbers' to give idea about the look of a program flowchart.

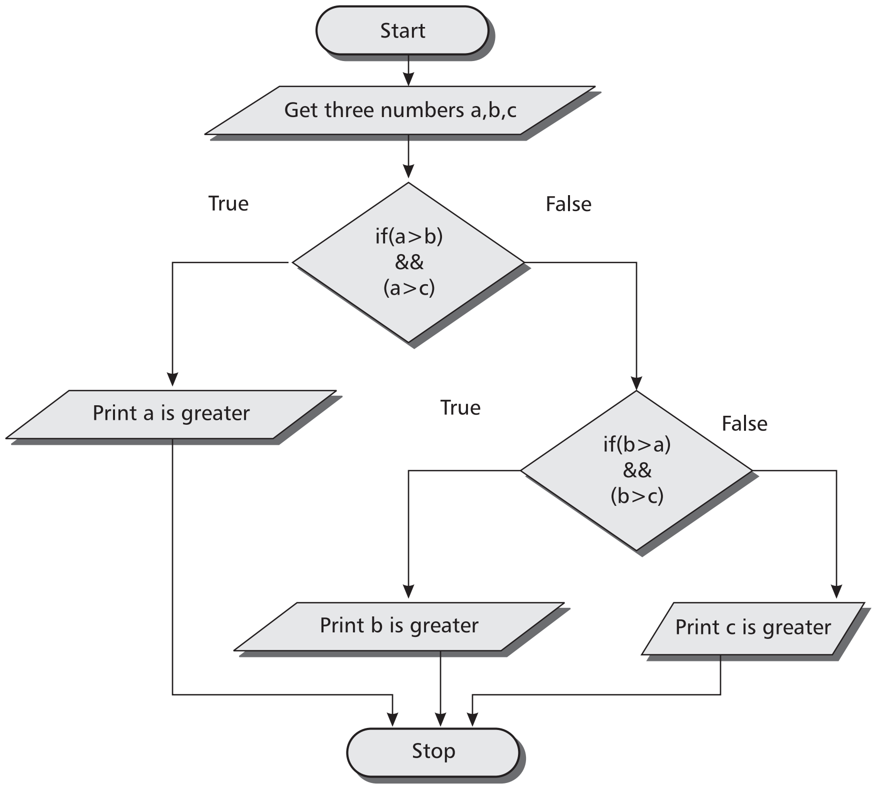


Fig 4.1 Flowchart of finding maximum of three numbers

Before drawing flowchart being familiar with the **standard symbols of flowchart** is required. Follows the list of symbols used in program flowchart along with their brief description:

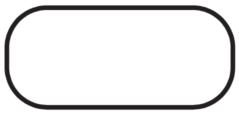
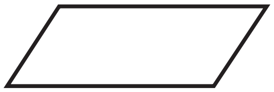
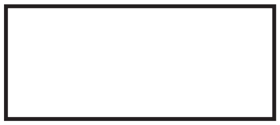
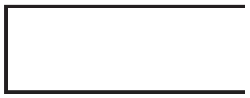
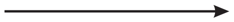
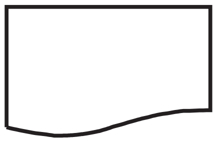
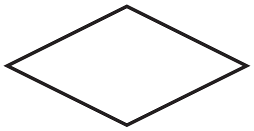
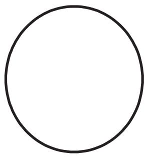
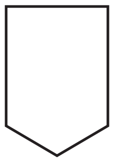
The following rules should be followed while drawing program flowcharts:

- Only the standard symbols should be used in program flowcharts.
- Program logic should depict the flow from top to bottom and from left to right.
- Each symbol used in a program flowchart should contain only one entry point and one exit point, with the exception of the decision symbol.
- The operations shown within a symbol of a program flowchart should be expressed independent of any particular programming language.
- All decision branches should be well labeled.

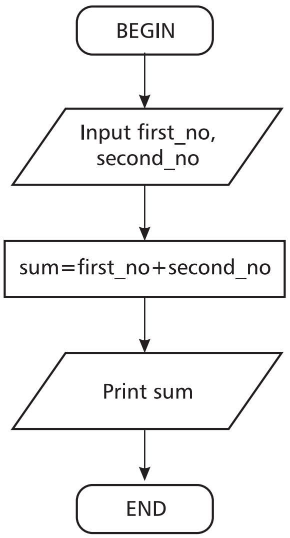
Now follows some preliminary flowchart:

(Please note that the corresponding ‘C’ code is given along with each flowchart. First go through the flowcharts and ignore the codes. Revisit the ‘C’ codes when you complete PART-B Chapter-4 of this book.)

Table 4.1 Flowchart symbols

Symbol	Name	Usage
	Terminal	Used to show the beginning and end of a set of computer-related process.
	Input/output	Used to show any input/output operation.
	Computer processing	Used to show any processing performed by a computer system.
	Comment	Used to write any explanatory statement required to clarify something.
	Flow line	Used to connect the symbols.
	Document input/output	Used when input comes from a document and output goes to a document.
	Decision	Used to show any point in the process where a decision must be made to determine further action.
	On-page connector	Used to connect parts of a flowchart continued on the same page.
	Off-page connector	Used to connect parts of a flowchart continued to separate pages.

PROGRAM 1 Flowchart to show how sum of two numbers can be obtained.

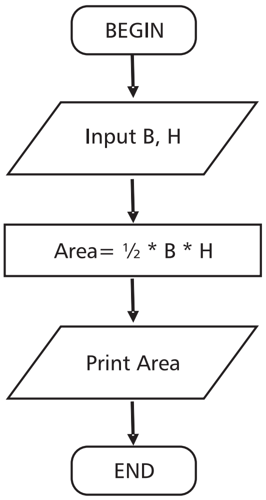


```
#include<stdio.h>
void main()
{
int first_no, second_no, sum;
scanf("%d", &first_no);
scanf("%d", &second_no);
sum=first_no + second_no;
printf("%d", sum);
}
```

PROGRAM 2 Flowchart to calculate area of triangle based on base and altitude.

Area of triangle = ½ × Base × Altitude

Accordingly the flow chart follows:



```
#include<stdio.h>
void main()
{
int B, H, Area;
scanf("%d", &B);
scanf("%d", &H);
Area=0.5 * B * H;
printf("%d", Area);
}
```