Computer Organization

(CS303/EE-504B/EEE-504B/IC-504B)

Second Edition

WBUT-2015

About the Author

Tarun Kumar Ghosh is currently Assistant Professor and Head, Department of Computer Science and Engineering, Haldia Institute of Technology, Haldia, West Bengal. He previously served as Lecturer in the Department of Computer Science and Engineering, Asansol Engineering College, Asansol, West Bengal. He received his ME degree in Computer Science and Technology from Bengal Engineering College (Deemed University), Shibpur, Howrah (currently known as Bengal Engineering and Science University), and BTech in CSE from the University of Calcutta.

Prof. Ghosh is a member of ACM (Association for Computing Machinery), CSI (Computer Society of India) and Indian Society of Technical Education. He played an instrumental role in forming the CSI Student Branch at Haldia Institute of Technology and has been working there since as the coordinator. He has published several research papers in various conference proceedings and journals and conducted a number of workshops and seminars. A coordinator at IGNOU, Haldia Study Centre, Prof. Ghosh is also the recipient of a UGC Scholarship at the postgraduate level. His areas of interest include Computer Architecture, Grid Computing, Interconnection Networks, Operating Systems, and Computer Graphics.

Computer Organization

(CS303/EE-504B/EEE-504B/IC-504B)

Second Edition

WBUT-2015

Tarun Kumar Ghosh
Assistant Professor and Head
Department of CSE
Haldia Institute of Technology
Haldia, West Bengal

McGraw Hill Education (India) Private Limited
NEW DELHI

McGraw Hill Education Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

McGraw Hill Education (India) Private Limited

Published by McGraw Hill Education (India) Private Limited
P-24, Green Park Extension, New Delhi 110 016

Computer Organization, 2e (WBUT-2015)

ISBN 13: 978-93-392-2209-3
ISBN 10: 93-392-2209-1

Managing Director: *Kaushik Bellani*
Head—Products (Higher Education and Professional): *Vibha Mahajan*

Associate Sponsoring Editor: *Koyel Ghosh*
Editorial Executive: *Piyali Chatterjee*

Manager—Production Systems: *Satinder S Baveja*
Assistant Manager—Editorial Services: *Sohini Mukherjee*
Assistant Manager—Production: *Anjali Razdan*
Senior Graphic Designer—Cover: *Meenu Raghav*

Senior Publishing Manager (SEM & Tech. Ed.): *Shalini Jha*
Assistant Product Manager: *Tina Jajoriya*

General Manager—Production: *Rajender P Ghansela*
Manager—Production: *Reji Kumar*

Typeset at Text-o-Graphics, B-1/56, Aravali Apartment, Sector-34, Noida 201 301, and printed at

Cover Printer:

Visit us at: www.mheducation.co.in

Dedication

This book is dedicated to all my beloved students who are the stirring force behind this work, and to my wife, Mamata, and my daughter, Twarita, who are still bearing with my odd working hours.

Contents

Preface

OVERVIEW

The emergence of computers has revolutionized the design of electronic systems. Nowadays, computers are used almost everywhere: education, business, entertainment, games, sports, security, conference, and many more avenues. Designing a computer is a highly sophisticated and complex process. Over the years, different methods have been applied to design computers.

A brief explanation of the title *Computer Organization* is necessary here. Computer organization is a lower level, more concrete description of the system that involves how the constituent parts of the system are interconnected and how they inter-operate in order to realize the architectural specifications. The organizational issues include the hardware details transparent to the programmer like memory technology, interfacing between the CPU and peripherals, and system interconnect components like computer buses and switches. Computer architecture deals with the conceptual design and basic overview of a computer system. It refers to the parameters of a computer system that are visible to the user. It includes the instruction addressing modes, instruction formats, instruction sets, I/O mechanism, and related concepts.

About the Book

This book is the outgrowth of a series of lectures based on the course *Computer Organization and Architecture* and *Advanced Computer Architecture*, delivered over the last several years at different colleges under West Bengal University of Technology, West Bengal. The basic aim is to simplify the subject for every student.

This book is intended to serve as a first-level text for the revised curricula of *Computer Organization* (CS303), taught in 3rd semester of Computer Science and Information Technology branches and EE-504B, EEE-504B and IC-504B, taught in 5th semester of Electrical Engineering, Electrical and Electronics Engineering and Instrumentation and Communication Engineering branches of West Bengal University of Technology (WBUT), West Bengal.

Covering major parts of the 1st semester paper (MCA-101) of MCA of WBUT, it will also be useful for students of MCA, BCA and any other diploma course in Computer Science.

This book is mostly self-contained, assuming that the reader has a basic knowledge of computer programming, number systems, and digital logic. For the benefit of those who have no prior knowledge of digital logic and assembly language, the book includes an overview of the essential topics of digital electronics and assembly-language programming in the Appendix.

Salient Features

- Complete coverage of the WBUT syllabus (2010 Regulation)
- Presented in a lucid and concise manner using ample diagrams and flow charts
- Solved WBUT Question Papers from 2007 to 2014
- Comprehensive Pedagogy:

Illustrations:	350	
Solved Problems:	135	
Review Questions:	*Multiple Choice Questions*	165
	Short and Long Answer Questions	195

Chapter Organization

The book is divided into seven chapters:

- *Chapter 1* presents basics of digital computers, different layers in a computer system, overview of the operating systems, types of computers, brief history of computers, details of von-Neumann computers, and introduction to the instruction cycle.
- *Chapter 2* discusses different data representation methods in computer registers and their relative advantages and disadvantages, different arithmetic operations, and error detection and correction methods.
- *Chapter 3* introduces different types of registers, register transfer language (RTL) and shows how RTL is used to express micro-operations in symbolic form. This chapter also explains how a common bus can be constructed. Some important arithmetic units are also designed. A hypothetical arithmetic logic unit (ALU) is developed to show the hardware design of the most common micro-operations, and lastly a brief idea about bit-slice processors is introduced.
- *Chapter 4* gives the parameters of a memory, the concept of memory hierarchy, and different concepts and working principles of main memory, secondary memory and cache memory are covered. Several techniques to improve cache memory performance are described. The concept and operation of associative memory is explained in detail. The overview of virtual memory is also explained.
- *Chapter 5* contains basics of instruction sets, and the instruction formats. Different CPU organizations are discussed in detail with the help of examples. The basics of an instruction cycle and a variety of addressing modes are explained. CISC and RISC concepts and their characteristics and their relative advantages and disadvantages are described. As a case study of RISC machines, Sun SPARC and PowerPC are briefly discussed. The basic concepts about the pipelining technique are also introduced.
- *Chapter 6* presents control unit design, using both hardwired and microprogramming approaches. Both the approaches are illustrated through examples. Also, the microprogram sequencer is briefly discussed. At last, the basic concept of nanoprogramming is introduced.

- *Chapter 7* explains the techniques that computers use to communicate with peripheral devices. Three modes of data transfer between computers and peripherals are discussed in detail: programmed I/O, interrupt I/O and direct memory access (DMA). In addition, the basics of bus arbitration and input-output processor (IOP) are introduced. Lastly, both parallel and serial data transfer techniques are described with some standard interfaces.

Feedback and Comments

I look forward to the comments and suggestions from the readers for further improving the book. I can be reached at *tarun_ghosh_2000@rediffmail.com*

T K Ghosh

Publisher's Note

Remember to write to us. We look forward to receiving your feedback, comments and ideas to enhance the quality of this book. You can reach us at *info.india@mheducation.com*. Please mention the title and the author's name as the subject. In case you spot piracy of this book, please do let us know.

Acknowledgements

I sincerely thank all my colleagues, especially Mr. Anjan Mishra, Registrar; Mr. Susmit Maity; Mr S K Sahnawaj; and Mr Subhankar Joardar. I am grateful to all my teachers, especially Prof. Sudarshan Das, Prof. Subhrangshu Bandyopadhyay and Prof. Uma Bhattacharya. I must acknowledge all my students, including Sri Dipankar Dutta and Sri Prithayan Barua, without whom it was impossible to come up with the idea of writing this book. I sincerely appreciate my friend, Mr Tushar Kanti Das for his encouragement and helpful comments during the writing this book.

I am thankful to the following reviewers who have assessed various chapters of the script and provided valuable suggestions for improvement:

Mihir Kumar Mahato *MCKV Institute of Technology, Howrah*
Debojyoti Bagchi *Calcutta Institute of Engineering and Management, Kolkata*

I am also thankful to the team at McGraw Hill Education (India) Pvt Ltd., especially Ms Vibha Mahajan, Ms Koyel Ghosh, Ms Piyali Chatterjee, Ms Shalini Jha, Ms Tina Jajoriya, Ms Sohini Mukherjee and Ms Anjali Razdan for their cooperation and support in bringing out this book on time.

Last but not least, special thanks goes to my family members for their constant support throughout the period of preparation of the manuscript.

T K Ghosh

ROADMAP TO THE SYLLABUS

This text is suitable for the following subject codes:

CS303: Computer Organization (CSE & IT)

EE-504B: Computer Organization (EE)

EEE-504B: Computer Organization (EEE)

IC-504B: Computer Organization (ICE)

MCA 101: Computer Organization and Architecture (MCA)

CS303: COMPUTER ORGANIZATION (CSE & IT)
EE-504B: COMPUTER ORGANIZATION (EE)
EEE-504B: COMPUTER ORGANIZATION (EEE)
IC-504B: COMPUTER ORGANIZATION (ICE)

Module – 1:
Basic organization of the stored program computer and Operation sequence for execution of a program. Role of operating systems and Compiler/Assembler. Fetch, decode and execute cycle, Concept of operator, Operand, Registers and Storage, Instruction format. Instruction sets and Addressing modes. Commonly used number systems. Fixed and floating point representation of numbers.

Go to
CHAPTER 1: **FUNDAMENTALS OF COMPUTERS**
CHAPTER 2: **DATA REPRESENTATION AND COMPUTER ARITHMETIC**
CHAPTER 5: **COMPUTER INSTRUCTION SET**

Module – 2:
Overflow and underflow. Design of adders - Ripple carry and Carry look ahead principles. Design of ALU. Fixed point multiplication - Booth's algorithm. Fixed point division - Restoring and Non-restoring algorithms. Floating point - IEEE 754 standard.

Go to
CHAPTER 2: **DATA REPRESENTATION AND COMPUTER ARITHMETIC**
CHAPTER 3: **DATAPATH AND DESIGN OF ARITHMETIC LOGIC UNIT**

Module – 3:
Memory unit design with special emphasis on implementation of CPU-memory interfacing. Memory organization, Static and Dynamic memory, Memory hierarchy, Associative memory, Cache memory, Virtual memory, Data path design for read/write access.

Go to *CHAPTER 3*: **DATAPATH AND DESIGN OF ARITHMETIC LOGIC UNIT**
CHAPTER 4: **MEMORY ORGANIZATION**

Module – 4: Design of control unit - Hardwired and Microprogrammed control.
Introduction to instruction pipelining. Introduction to RISC architectures. RISC vs CISC architectures. I/O operations - Concept of handshaking, Polled I/O, interrupt and DMA.

Go to *CHAPTER 6*: **DESIGN OF CONTROL UNIT**
CHAPTER 7: **INPUT–OUTPUT ORGANIZATION**

MCA 101: COMPUTER ORGANIZATION & ARCHITECTURE (MCA)

Data and number representation-binary-complement representation, BCD-ASCII, Conversion of numbers from one number system to the other, (r-1)'s complement representation, Binary arithmetic

Go to *CHAPTER 2*: **DATA REPRESENTATION AND COMPUTER ARITHMETIC**

Structure of a digital machine (von-Neumann architecture), Logic gates, Basic logic operations, Truth tables, Boolean expression, Simplification, Combination circuits, Adders, Multiplexer, Sequential circuits, Registers.

Go to *CHAPTER 3*: **DATAPATH AND DESIGN OF ARITHMETIC LOGIC UNIT**

ROM, PROM, EPROM and dynamic RAM, Digital components, Bus structure: - Address bus, Data bus and DMA controller.

Go to *CHAPTER 4*: **MEMORY ORGANIZATION**

Karnaugh Map, Coder, Decoder, Counter – Asynchronous and Synchronous. Flip Flops – RS, JK, and D and T.

Go to *CHAPTER 5*: **COMPUTER INSTRUCTION SET**

Basic computer organisation and design, Micro-programmed control, Data representation, Register transfer and micro-operations, Central processing unit, Pipeline and vector processing, Computer arithmetic, Input-output organisation, Memory organisation, Microprocessors (8085), Personal computing, CPU architecture, Instruction format, Addressing mode, Stacks and handling of interrupts, Assembly language – Elementary probloems.

Go to *CHAPTER 6*: **DESIGN OF CONTROL UNIT**
CHAPTER 7: **INPUT–OUTPUT ORGANIZATION**

CHAPTER

1
Fundamentals of Computers

1.1 INTRODUCTION

Nowadays computers are used in almost all steps of life: education, business, entertainment, games, sports, security, conference, etc. The design of a computer is highly sophisticated and complex. Over the years, different methods have been applied in designing it. Parameters such as performance, cost, storage capacity, types of use determine the choice of different concepts and methods used in designing a computer. The study of these concepts and techniques will be the goal of the book.

A computer is an automatic machine made up of electronic and electro-mechanical devices, which processes the data.

Characteristics of Computers

(a) Speed—It has fast speed operation of several million operations per second.
(b) Accuracy—It is capable of performing calculations to the extent of 64 decimal accuracy.
(c) Storage—It is capable of storing large volumes of information.
(d) Decision making—It is capable of decision making according to the supplied information.

1.2 DIGITAL COMPUTERS

Most of the modern day computers are digital computers though some are also analog computers. An analog computer senses input signals whose values keep changing continuously. It allows physical processes such as pressure, acceleration, power, force etc. to be represented by electrical current or voltage signals. Digital computers (or simply computers) perform the calculations on numerical or digital values. Today's most of the computers fall into this class. Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit. Information is represented in digital computers in groups of bits. By using different coding techniques, groups of bits can be made to represent discrete symbols, such as decimal digits or letters of the alphabet.

The computer system consists of three main components.

- Hardware
- Software
- Human resources

(a) Hardware Hardware refers to the physical devices attached with the computer such as central processing unit (CPU), keyboard, monitor, disk drive, printer and other peripherals. A block diagram of a simple computer is shown in the Fig. 1.1.

(i) *Memory:* The memory unit stores programs as well as data. The memory unit consists of different types of memories, each of which has been used for some specific purpose. There are basically three types of memories used in a system:

- *Secondary memory*: The slow-speed and low-cost devices that provide backup storage are called secondary memory. The most commonly used secondary memories are magnetic disks, such as hard disk, floppy disk and magnetic tapes. This type of memory is used for storing all programs and data, as this is used in bulk size.
- *Main Memory*: This is the memory that communicates directly with CPU. Only programs and data currently needed by the CPU for execution reside in the main memory.
- *Cache memory*: This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate.

(ii) *Arithmetic and Logic Unit (ALU):* It is the main processing unit which performs arithmetic and other data processing tasks as specified by the control unit. The ALU and control unit are the main constituent parts of the Central Processing Unit (CPU). Another component of the CPU is register unit—collection of different registers, used to hold the data or instruction temporarily (for register, see Chapter 3).

(iii) *Control Unit:* This is the unit that supervises the flow of information between various units. The control unit retrieves the instructions using registers one by one from the program, which is stored in the memory. The instructions are interpreted (or decoded) by the control unit itself and then the decoded instructions are sent to the ALU for processing.

(iv) *Input Unit:* This unit transfers the information as provided by the users into memory. Examples include keyboard, mouse, scanner, etc.

(v) *Output Unit:* The output units receive the result of the computation and displayed to the monitor or the user gets the printed results by means of a printer.

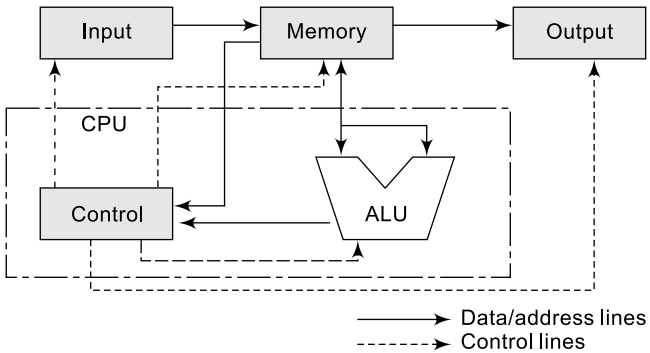


Figure 1.1 *Block diagram of a computer*

(vi) *Bus*: A bus is a subsystem that transfers data or address or a special signal (like read, write, etc) between various units of a computer or between two computers. A bus means a group of common communication lines, where each line is used to transfer one bit of data or address at a time. A bus can be used as a data bus or address bus. A data bus is used to transfer data between two units of a computer. An address bus is used to transfer address information. Sometimes data buses and address buses are used separately, and sometimes a single bus is used for both data and address transfer.

(b) Software The information processed by the hardware devices. Software consists of the instructions and data that the computers manipulate to perform various tasks. A sequence of instructions is called a program. Generally, software can be either application software or system software. Application software is a program or collection of programs used to solve a particular application-oriented problem. Examples include editor program, real player, and railway reservation program. System software is a program used to manage the entire system and to help in executing various application programs. Operating systems, compilers and device drivers are some of the system software's examples. System-programs are generally machine-dependent and are not concerned with specific application programs.

(c) Human resources It is the manpower and skilled personnel (programmers) available to perform operations on computer systems.

This book provides the knowledge necessary to understand the subject in a concise manner. This subject is sometimes considered from two different points of view as computer architecture and computer organization.

Computer architecture is the branch of study that deals with the conceptual design and basic overview of a computer system. It refers to the parameters of a computer system those are visible to the user. In other words, it deals with the attributes that have direct impact on the execution of a program. It includes the instruction addressing modes, instruction formats, instruction sets, I/O mechanism, etc.

Computer organization is a lower level, more concrete description of the system that involves how the constituent parts of the system are interconnected and how they inter-operate in order to realize the architectural specifications. The organizational issues include the hardware details transparent to the programmer, like the memory technology, interfacing between the CPU and peripherals, system interconnect components like, computer buses and switches.

For example, an issue like whether a computer will include multiply instruction in its instruction set is an architectural issue. Whether that multiply instruction will be implemented by a special hardware unit or by a technique of repeated add-shift operations, is an organizational issue.

1.3 LAYERS IN A COMPUTER SYSTEM

A computer system can be viewed as a collection of different layers, as shown in Fig. 1.2. The innermost layer is the hardware part that consists of central processing unit (CPU), main memory, input/output (I/O) devices, secondary storage, etc. The hardware provides the basic computing resources. The *Basic Input-Output System* (*BIOS*) is a program consisting of I/O drivers, which are

different programs to perform various I/O operations on behalf of various peripheral devices attached to the computer. When a program needs an I/O operation, it calls this program. BIOS programs are basically programs embedded on a chip, called *firmware*.

When first power is on, this program runs. The main task of the BIOS is to identify and initiate component hardware like hard drives, keyboard, and mouse. This prepares the computer to work by loading the operating system into the main memory from the hard disk. This process is known as *booting* (short form of bootstrapping). In summary, BIOS can be said to be a coded program embedded on a chip (firmware) that recognizes and controls various peripheral devices that build up a computer.

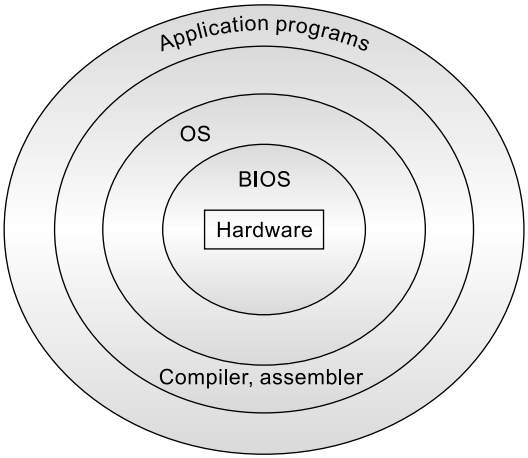


Figure 1.2 *Abstract layers in computer sysem*

A programmer writes an application program in a high-level language or assembly language. The earliest computers had their instructions written in a binary code known as *machine language* that could be executed directly. An instruction in machine language meaning "add the contents of two memory locations" might take the form

00 1110 1100000000 100 1100 100000 111

Machine-language programs are extremely difficult for humans to write and so are very error-prone. A substantial improvement is obtained by allowing operations and operand addresses to be expressed in an easily understood symbolic form such as

ADD X1, X2

This symbolic format, which is referred to as *assembly language,* came into use in the 1950s, as computer programs were growing in size and complexity. An assembly-language program requires a special system program called an *assembler* to translate it into machine language before it can be executed. Details about assembly language and assembler are discussed in the Appendix.

Because assembly language is specific to a given machine, programs written in assembly language are not transferable from one machine to another. To circumvent this limitation, general-purpose languages as BASIC, FORTRAN, PASCAL, and C/C++ have been devised; a program written in these languages can be machine-independent. These languages are called *high-level languages*.

A *compiler* is a system program, which converts the high-level language program into equivalent machine language program consisting of instructions of binary numbers (see Fig. 1.3). This translation in the machine language is called the *object code*. Each machine (processor) needs its own compiler or an interpreter for each high-level language. The main difference between a compiler and an interpreter lies in the process of generating machine code. The compiler reads the entire problem first and then generates the object code. On the other hand, the interpreter reads one instruction at a time, produces its object code, and executes the instruction before

Figure 1.3 *Function of a compiler*

reading the next instruction. BASIC language is a common example of an interpreter. Compilers are generally used in languages such as FORTRAN, COBOL, C/C++ and PASCAL.

An *operating system* (OS) is a set of programs and utilities, which acts as the interface between user programs and computer hardware. The purpose of an operating system is to provide an environment in which a user may execute the programs. An operating system is similar to a government. Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work. An operating system can be viewed as a resource manager. The operating system provides the means for the proper use of the resources in the operation of the computer system. The main resources are the computer hardware in the form of processor (i.e. CPU), memory, input/output devices, communication devices, etc., software and data. The following are the main functions of an operating system:

- User's program management
- Memory management
- Secondary storage management
- I/O management
- File management
- Protection
- Networking management
- Command interpretation

Types of Operating Systems

The operating systems (OS) can be classified into the following types:

Batch processing OS During 1960s this OS was used. Program, data and appropriate system commands to be submitted together form a job. Same type jobs are batched together and are executed at a time. This OS usually allows little or no interaction between users and executing programs. Thus, if any error is encountered, the program has to be debugged offline, which makes the OS very inconvenient in developing the program. Batch processing OS too has the drawback: large CPU idle time. MS-DOS is a popular batch processing OS.

Multiprogramming OS In this OS, multiple numbers of programs are executed concurrently. Several programs are stored in the main memory at a time. The OS takes one program from the memory and assigns it to the CPU for execution. Whenever, an I/O operation is encountered in the program, the CPU switches from this program to the next waiting program in the memory. Thus, during this time, the I/O operation for the first program is taken care by the I/O processor or DMA (direct memory access) controller and the second program is executed by the CPU. So, the CPU is not idle at any time and thus the throughput (i.e. no. of programs completed per unit time) of the system increases. Windows 98 is an example of multiprogramming OS.

Multi-tasking or Time-sharing OS This OS is basically a logical extension of multiprogramming OS. Here, the total CPU execution time is divided into equal number of slots. Multiple programs are kept in main memory. The CPU takes one program from the memory and executes the program for the defined time slot. When time slot is expired, the CPU switches from this program to the next in waiting. Thus, the CPU time is shared by several programs, which is why it is called time-shared OS. The main advantage of this OS is good CPU response time. Very popular example of this OS is UNIX.

Multithreading OS This is an operating system that allows different parts, called threads, of a program to execute concurrently, using one CPU. However, the program has to be designed well so that the different threads do not interfere with each other. Examples of the OS are Linux, UNIX and Windows 2000.

Real-time OS These systems are used in the applications with very rigid requirement of completion of task in the pre-specified time. The real-time operating systems are used in the environment, where a large number of events mostly external to the computer are taken as inputs and processed in a highly time-bound manner. These operating systems are application-oriented and are used in air defence system, nuclear plant, petro-chemical plant, etc.

Distributed OS In 1990s, the decrease in hardware costs gives rise to the development of distributed systems, where several CPUs are used to share their program-codes in order to execute one single task through high-speed network. Here, each CPU has attached main memory and as a result the CPUs do not share their main memory for code sharing. This system has advantages of good resource sharing and high computation speed. Amoeba is an example of distributed OS.

Multiprocessing OS The OS is capable of supporting and utilizing more than one CPU for one single task computation. In this case, the CPUs interact with each other through a large shared main memory. Examples of multiprocessing OS are UNIX, Linux and Windows 2000.

1.4 TYPES OF COMPUTERS

Digital computers can be categorized into four different types, based on their performance, size and cost. They are: mainframe computers, minicomputers, microcomputers and supercomputers.

Mainframe computer It is a large computer system consisting of thousands of ICs, which is physically distributed in more than one place. This computer is designed for intensive computational tasks and used by large organizations like banks, railways and hospitals. Mainframe computer is often shared by multiple users connected to the computer through several terminals. This computer is very expensive. Examples include IBM system/360, Burroughs B 5000 and UNIVAC 1100/2200 series.

Minicomputer This class of computers is smaller and slower version of mainframe computer. Thus, its cost is very less compared to the mainframe computer. This machine is designed to serve multiple users simultaneously and used by smaller organizations and research centres. Computers like DEC's PDP, HP 3000 series and CDC 1700 are minicomputers.

Microcomputer Invention of microprocessor (i.e. CPU on a chip) gives rise to the microcomputer that is small, low-cost and single user machine. It is also called personal computer (PC). This inexpensive computer is designed to use on a small desk or even to carry. This class of computers is very popular, due to its high performance per cost ratio and size. The more powerful microcomputer designed to perform scientific applications is called *workstation*. IBM PC series based on Intel's 80x86 family, Apple's Macintosh and Motorola's 680x0 family are examples of microcomputers.

Supercomputer This class of computers is the most powerful and expensive computer available today. This computer is design to perform fast using multiprocessing and parallel processing techniques. This machine is specially used for complex scientific applications, like weather forecasting, satellite launching, climate research and nuclear research. Popular supercomputers are Cray-1, Power-PC and Fujitsu's VP 200. An important point to be noted that today's supercomputer tends to become tomorrow's normal computer.

1.5 HISTORY OF COMPUTERS

The modern day computers are the results of combined efforts of several scientists over last century. The history of computers is divided into two eras: Mechanical Era and Electronic Era.

1.5.1 Mechanical Era

Abacus It is a manual device combining two fundamental concepts.

- Numerical information represented in physical form.
- Information can be manipulated in the physical form.

The mathematical operations such as addition, subtraction, division and multiplication can be performed on abacus (Fig. 1.4).

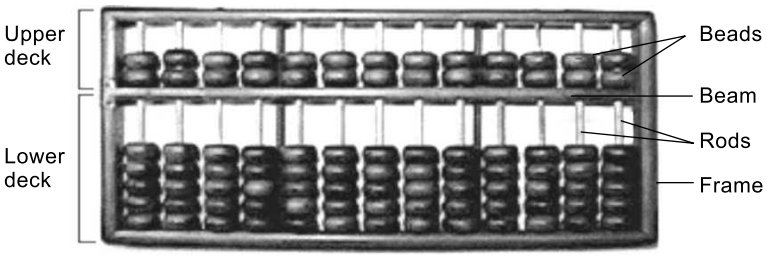


Figure 1.4 *Abacus*

The abacus is nearly 2000 years old. It is very useful for teaching simple mathematics to children. The abacus has a wooden frame with vertical rods on which wooden beads slide. Arithmetic problems are solved when these beads are moved around. Beads are considered counted, when moved towards the beam that separates the two decks: upper deck and lower deck. Each bead in the upper deck has a value of five and each bead in the lower deck has a value of one.

Mechanical Computer/Calculator Mechanical computer was invented by French philosopher Pascal in 1643. This could add and subtract 8 columns of numbers. Decimal numbers were engraved on counter wheels much like those in a car's odometer.

Babbage's Difference Engine It was the first computer to perform multi-step operations automatically, i.e. without human intervention in every step, and was designed by mathematician Charles Babbage in the 19th century. A difference engine is a special-purpose mechanical calculator-cum-

computer designed to tabulate polynomial functions. Since logarithmic and trigonometric functions can be approximated by polynomials, such a machine is more general than it appears at first.

Babbage's Analytical Engine Analytical Engine was the improved version of the Difference Engine. This machine is considered to be the first general-purpose programmable computer ever designed. It was a decimal computer with word length of 50 decimal digits and a storage capacity of 1000 digits. An interesting feature on this machine was conditional branch instruction handling. Two major components of this machine are an ALU called the *mill* and a main memory called the *store.* A program for the Analytical Engine (Fig. 1.5) was composed of two sequences of punched cards: Operation cards and Variable cards.

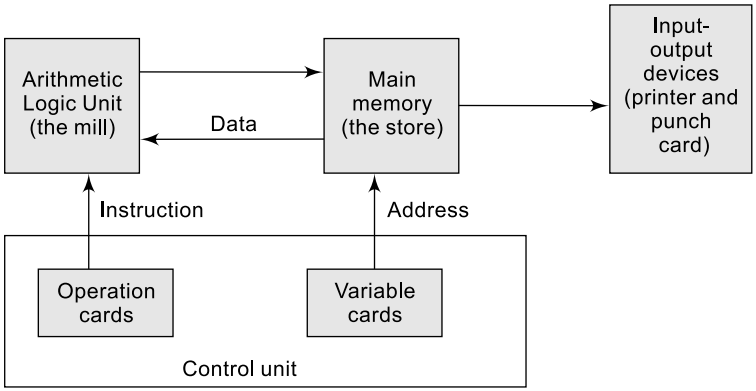


Figure 1.5 *Structure of Babbage's analytical engine*

Punched card is a magnetic plat, on which the holes are punched and they are sensed by a machine. Operation cards are used to select the operation to be performed by the mill and variable cards are used to specify the locations in the store from which inputs were to be taken or results sent.

1.5.2 Electronic Era

Five generations of electronic computers have been distinguished. Their major characteristics are summarized in the Table 1.1.

Table 1.1 *Generations of computers*

Generation no	*Technologies*	*Hardware features*	*Software features*	*Representative Computers*
1st (1946-1954)	Vacuum tubes, CRT memories	Fixed-point arithmetic	Machine language, assembly language	Institute for Advanced Studies (IAS), UNIVAC (Universal Automatic Computer), ENIAC (Electronic Numerical Integrator & Calculator)

(*Contd.*)

(*Contd.*)

2nd(1955-1964)	Discrete transistors, ferrite cores, magnetic disks	Floating-point arithmetic	High-level languages, subroutines	IBM (International Business Machine) 7094,
3rd (1965-1974)	Integrated circuits (SSI and MSI)	Microprogra-mming, Pipelining, Cache memory	Multi-programming operating systems, Virtual memory	IBM 360, DEC's (Digital Equipment Corporation) PDP-8
4th (1975-1978)	LSI/VLSI circuits, Semiconductor memories	Microprocessors, Micro-computers	Real-time OS, parallel languages, RISC	Motorola's 68020, Intel's 80x86 family.
5th (1979-onwards)	ULSI circuits, optical disk,	Embedded system, Massive parallelism	Multimedia, Artificial Intelligence, Internet	Intel's Xeon, Duo-core.

IAS Computer/ Von-Neumann Computer In 1946, Von Neumann and his colleagues began the design of a new stored-program computer, now referred to as the IAS computer, at the Institute for Advanced Studies, Princeton. Nearly, all modern computers still use this stored-program concept. This concept has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

This machine employed a random-access Cathode-Ray-Tube (CRT) main memory, which permitted an entire word to be accessed in one operation. Parallel binary circuits were employed. Each instruction contained only one memory address and had the format:

OPCODE ADDRESS

The central processing unit (CPU) contained several high-speed (vacuum-tube) registers used as implicit storage locations for operands and results. Its input-output facilities were limited. It can be considered as the prototype of all subsequent general-purpose computers.

Instruction Format The basic unit of information i.e. the amount of information that can be transferred between the main memory and CPU in one step is a 40-bit word. The memory has a capacity of $2^{12} = 4096$ words. A word stored in the memory can represent either instruction or data.

Data The basic data item is a binary number having the format shown in Fig. 1.6. Leftmost bit represents the sign of number (0 for positive and 1 for negative) while the remaining 39 bits indicate the number's size. The numbers are represented as fixed-point numbers.

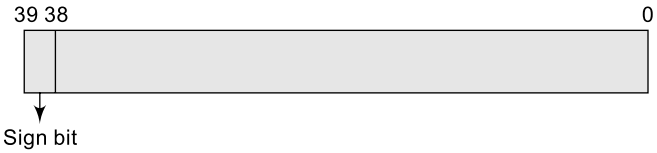


Figure 1.6 *Number word*

Instruction IAS instructions are 20 bits long, so that two instructions can be stored in each 40-bit memory location. An instruction consists of two parts, as shown in Fig. 1.7: an 8-bit op-code (operation code), which defines the operation to be performed (add, subtract, etc.) and a 12-bit address part, which can identify any of 2^{12} memory locations that may be used to store an operand of the instruction.

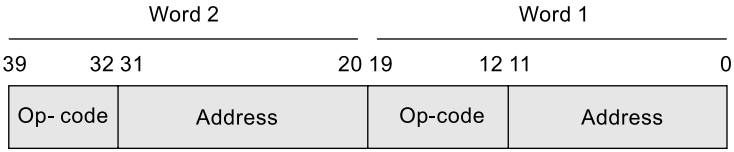


Figure 1.7 *Instruction word*

Reduced Word Length IAS instruction allows only one memory address. This results in a substantial reduction in word length. Two aspects of the IAS organization make this possible.

1. Fixed registers in the CPU are used to store operands and results. The IAS instructions automatically make use of these registers as required. In other words, CPU register addresses are implicitly specified by the op-code.
2. The instructions of a program are stored in the main memory in approximately the sequence in which they are to be executed. Hence the address of the next instruction pair is usually the address of the current instruction pair plus one. The need for a next instruction address in the instruction format is eliminated. Special branch instructions are included to permit the instruction execution sequence to be varied.

Structure of an IAS Computer The CPU of the IAS computer, as shown in Fig. 1.8, consists of a data-processing unit (also known as datapath) and a program control unit. It contains various processing and control circuits along with a set of high-speed registers (AC, MQ, DR, IBR, PC, IR, and MAR) intended for temporary storage of instructions or data or memory addresses. The arithmetic-logic circuits of the data-processing unit perform the main actions specified by instructions. The control circuits in the program-control unit (simply control unit) are responsible for fetching instructions, decoding instructions, fetching data (operands) from the memory and providing proper control signals for all CPU actions. An electronic clock circuit is used to generate the basic timing signals needed to synchronize the operation of the different parts of the system.

The main memory M is used for storing programs and data. A word transfer can take place between the 40-bit data register (DR) of the CPU and any location M(X) with address X in M. The address X to be used is stored in a 12-bit address register (MAR). The DR may be used to store an operand during the execution of an instruction. Two additional registers for the temporary storage of operands and results are included: the accumulator (AC) and the multiplier quotient register (MQ). Two instructions are fetched simultaneously from M and transferred to the program control unit. The instruction that is not to be executed immediately is placed in an instruction buffer register (IBR). The op-code of the other instruction is placed in the instruction register (IR) where it is decoded. The address field of the current instruction is transferred to the memory address register (MAR). Another address register called the instruction address register or the program counter (PC) is used to store the address of the next instruction to be executed.

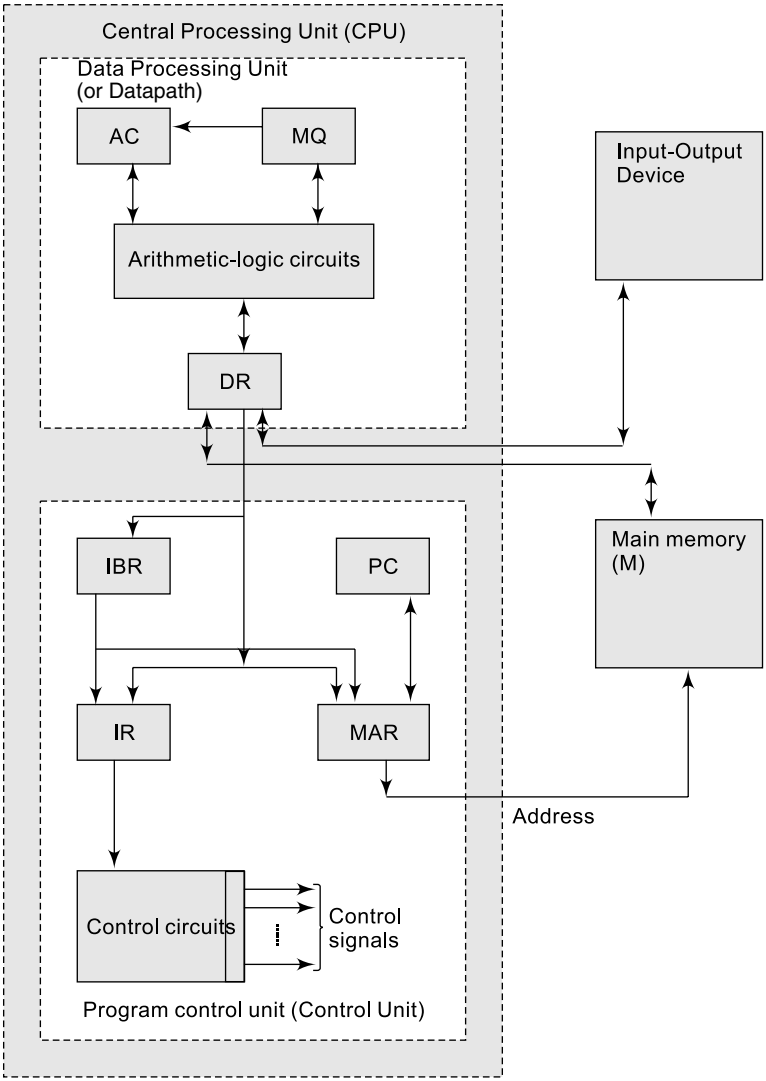


Figure 1.8 *Structure of an IAS computer*

Von-Neumann Bottleneck One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck. This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required data-word to come. Another way to reduce the problem is by using special type computers known as *Reduced Instruction Set Computers (RISC).* This class of computers generally uses a large number of registers, through which the most of the instructions are executed. This computer usually limits access

to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

1.6 VARIOUS SUBSYSTEMS OF A COMPUTER

Nearly all modern-day computers are following the Von Neumann's concept (i.e. stored program concept). So, we generally use the term "computer" throughout the book to mean the Von Neumann computer, unless it is stated otherwise.

In a computer, the component types recognized fall into four main groups: CPU, memory unit, I/O unit, and system bus; as shown in Figure 1.9. Here, we give a brief summary of the characteristics of these major components.

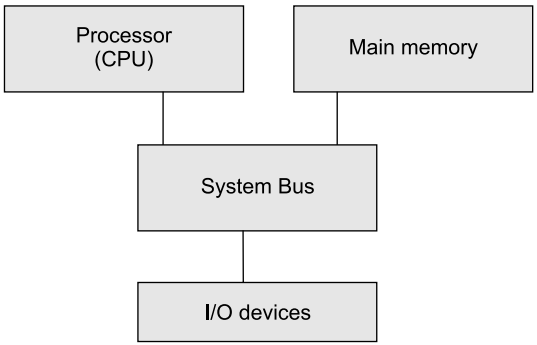


Figure 1.9 *Major components of a computer system*

1.6.1 Central Processing Unit (CPU)

The CPU (simply processor) is the major subsystem of a computer. It has two major functions:

1. Program Execution This involves a variety of data operations such as data processing, data storage and data movement.

2. Interfacing with other Subsystems For program execution, the processor has to interface with other subsystems like main memory, cache memory and I/O devices. The processor has the responsibility of overall coordination of intercommunication between these subsystems. This is usually done via a set of buses.

Most contemporary CPUs are microprocessors, implying that their physical implementation is a single VLSI chip. Figure 1.10 shows the essential internal organization of a CPU at the register-level. The CPU contains the logic needed to execute its particular instruction set and is divided into datapath (i.e. data-processing unit) and control units. The control unit generates addresses of instructions and data stored in external memory. In this particular system a cache memory is placed between the main memory M and the CPU. The cache is a fast buffer memory designed to hold an active portion of the system's address space; it is often placed, totally or in part, on the same IC as the CPU. Each memory request generated by the CPU is first directed to the cache. If the required information is not currently available to the cache, the request is redirected to M and the cache is automatically updated from M. The control unit fetches instructions from the cache or M and decodes them to derive the control signals needed for their execution. The CPU's datapath has the arithmetic-logic circuits that execute most instructions; it also has a set of registers for temporary data storage. The CPU manages a system bus, which is the main communication link among the CPU-cache subsystem, main memory, and the I/O devices.

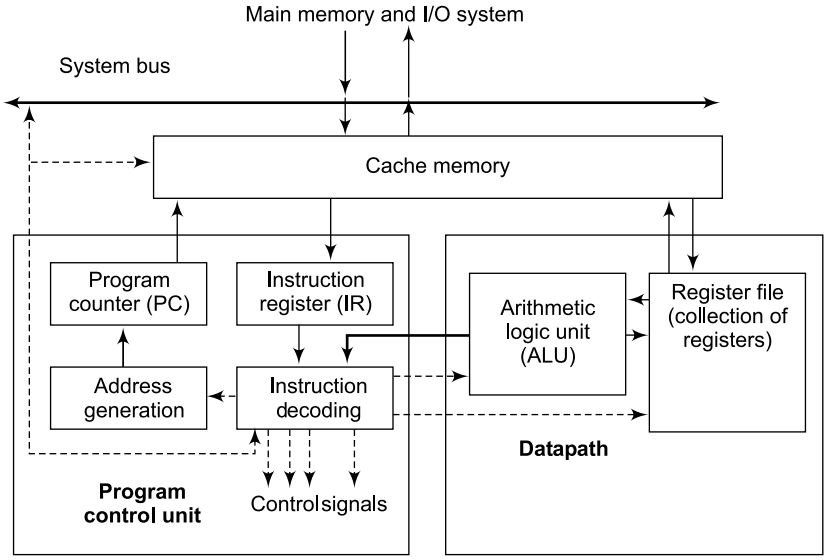


Figure 1.10 *Internal organization of a CPU*

The CPU is a synchronous sequential circuit whose clock period is the computer's basic unit of time. In one clock cycle the CPU can perform a register-transfer operation, such as fetching an instruction word from M via the system bus and loading it into the instruction register (IR). This operation can be expressed formally by IR ← M(PC); where PC is the program counter the CPU uses to hold the expected address of the next instruction word. Once in the control unit, an instruction is decoded to determine the actions needed for its execution; for example, perform an arithmetic operation on data words stored in CPU registers. The control unit then issues the sequence of control signals that enables execution of the instruction in question. The entire process of fetching, decoding, and executing an instruction constitutes the *instruction cycle* of CPU.

1.6.2 Memory Unit

A memory unit is an integral part of a computer system. Its primary function is to store all information needed by the system. Typically, a memory unit holds programs and data. A computer designer has to pay attention to the memory unit design because the memory system cost is a significant fraction of the cost of the total computer system. The system performance is largely dependent on the organization, storage capacity, and speed of operation of the memory system.

In a broad sense, a computer memory system can be logically divided into three categories:

- Main memory
- Secondary memory
- Cache memory

Main memory is the storage area in which all programs are executed. The CPU can directly access only those items that are stored in the main memory. Therefore, all programs and data must be within the main memory to speed up execution. Previously, the main memory was designed using magnetic cores. In modern computers, semiconductor technology is employed in main memory design. Usually

the size of the main memory is much larger than that of collection of processor registers (register file), and its operating speed is slower than the processor registers by a factor of almost 10.

Secondary memory refers to the storage medium comprising slow devices such as magnetic tapes and disks. These devices are used to hold data files and programs, such as compilers and database management systems, which are not frequently needed by the processor. Secondary memories are also referred to as auxiliary or backup store.

Studies have shown that typical programs spend most of their execution times in a few main routines or loops. In this situation, in a short period of time, the addresses generated by a CPU have the tendency to get clustered around small regions in the main memory. This phenomenon is known as the *locality of reference*. Suppose a small but a fast memory (faster than the main memory by 5 or 6 times) is installed to keep the most frequently needed information, and the CPU is instructed to access this fast memory (as opposed to the main memory). The efficiency of program execution can then be significantly improved. This newly added memory is known as the *cache* memory. This concept was first implemented in the IBM 360/85 computer. Since this technique proved to be very successful, it is included in modern-day computer systems.

1.6.3 I/O Unit

A user communicates with a computer system via the I/O devices interfaced to it. The user can enter programs and data using the keyboard on a terminal and execute the programs to obtain results. Therefore, the I/O devices connected to a computer system provide an efficient means of communication between the computer and the outside world. These I/O devices are commonly called *peripherals* and include keyboards, CRT displays, printers, disks, and teletypewriters.

The characteristics of the I/O devices are normally different from those of the computer. For example, the speed of operation of the peripherals is usually slower compared to the computer, and the word length of the computer may be different from the data format of the peripheral devices. To make the characteristics of the I/O devices compatible with those of the computer, interface hardware circuitry between the computer and I/O devices is necessary. Interfaces provide all input and output transfers between the computer and peripherals by using an I/O bus. An I/O bus carries three types of signals: device address, data, and command.

For large computers, a separate intelligent I/O processor (IOP) or data channel is provided to route all I/O transfers. To make small computer systems inexpensive, a separate interface rather than a smart I/O processor is provided with each I/O device. I/O processors control all major I/O functions and relieve the computer of these tasks.

The CPU uses the I/O bus when it executes an I/O instruction. A typical I/O instruction has three fields. When the computer executes an I/O instruction, the control unit decodes the op-code field and identifies it as an I/O instruction. The CPU then places the device address and command from respective fields of the I/O instruction on the I/O bus. The interfaces for various devices connected to the I/O bus decode this address, and the appropriate interface is selected. The identified interface decodes the command lines and determines the function to be performed. Typical functions include receiving data from an input device into the CPU or sending data to an output device from the CPU.

1.6.4 System Bus

A bus is a subsystem that transfers data or address or special signal (like read, write, etc) between

various units of a computer or between two computers. A bus means a group of common communication lines, where each line is used to transfer one bit of data or address at a time. A shared communication path consisting of one or more connection lines is known as a *bus*. Sometimes, we say *n*-bit bus or *n*-line bus, the meaning of which is that the bus consists of *n* parallel lines to transfer *n*-bits of data all at a time. The *n* is called width of the bus. The width of the bus has an impact of performance of computer. The wider the bus, the greater the number of bits transferred at a time. A bus can be used as a data bus or address bus. A data bus is used to transfer data between two units of a computer. An address bus is used to transfer address information.

All units of the computer communicate with each other by buses. In systems with many components, communication may be controlled by a subsystem called a *system bus* (*interconnection network*). The function of the system bus is to establish dynamic communication paths among the components via the buses under its control. For cost reasons, these buses are usually shared. Only two communicating devices can access and use a shared bus at any time, so a problem known as *bus contention* results when several system components request use of the bus. The system bus resolves such contention by selecting one of the requesting devices on some priority basis and connecting it to the bus. The system bus may place the other requesting devices in a queue.

Bus control is one of the functions of a processor such as a CPU or an IOP. An IOP controls a common I/O bus to which many IO devices are connected. The IOP is responsible for selecting a device to be connected to the I/O bus and from there to the main memory. It also acts as a buffer between the relatively slow I/O devices and the relatively fast main memory. Larger systems have special processors whose sole function is to supervise data transfers over shared buses.

1.7 INSTRUCTION CYCLE

The instruction cycle is one of the most important mental models of computation. This embodies the basic principle of how all modern processors work. This functional model has remained more or less the same over decades no matter how and when the development of processors has taken place ever since the days of Von Neumann architecture to today's supercomputers. The principles are fairly simple and can be easily generalized to any processor or operating system.

Once a computer has been powered on, it performs a continuous cycle called *instruction cycle* that consists of the following subcycles:

1. Fetch cycle
2. Decode cycle
3. Execute cycle
4. Interrupt cycle

Figure 1.11 shows the sequence of steps to be performed by any computer for each instruction execution.

Fetch Cycle The fetch cycle begins with retrieving the address stored in the *Program Counter* (PC) register. The address stored in the PC is some valid address in the memory holding the instruction to be executed (in case this address does not exist we would end up causing an interrupt or exception). The Central Processing Unit (CPU) completes this step by fetching the instruction stored at this address from the memory and transferring this instruction to a special register—*Instruction*

Register (*IR*)—to hold the instruction to be executed. The PC is incremented to point to the next address from which the new instruction is to be fetched.

Decode Cycle The decode cycle is used for interpreting the instruction that was fetched in the fetch cycle. Decoding determines three things:

- Type of operation to be performed by ALU.
- If data operands are needed for the instruction execution, the memory or register addresses of those data are calculated.
- The address of the result to be sent.

Execute Cycle This cycle, as the name suggests, simply executes the instruction that was fetched and decoded in ALU.

Typical instructions include:

- Performing logic operations on data (such as OR, AND, comparison, etc.)
- Performing arithmetic operations on the data (such as addition, subtraction, etc.)

Interrupt Cycle An interrupt can occur any time during the program execution. Whenever it is caused, a series of events takes place so that the instruction fetch-execute cycle can again resume after the OS calls the routine to handle the interrupt. Therefore, when an interrupt occurs, the following steps are performed by the OS:

- Suspend the execution of current instruction
- Push the address of current instruction on the memory stack
- Loading the PC with the first address of the interrupt handler
- This starts the instruction fetch execute cycle again for the instructions in the interrupt handler.
- Set the mode of operation as a privileged one, often termed as the s*upervisor mode*, so that the OS can execute the handler.
- Once the OS completes the execution of the interrupt handler, the address of the next instruction to be executed is obtained from popping the value of the address in the stack. The suspended instruction can now continue with its execution.

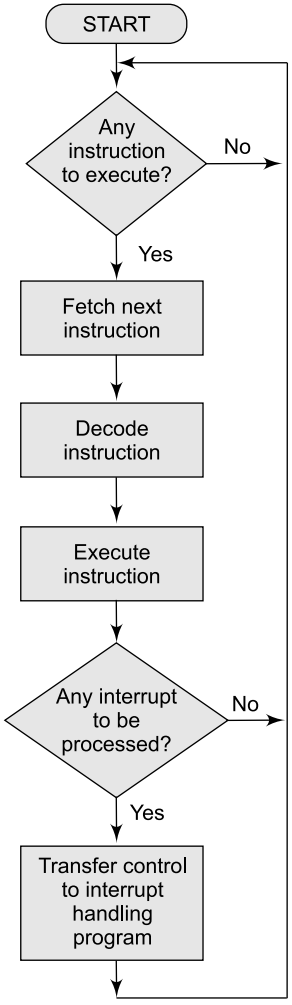


Figure 1.11 *A simple instruction cycle*

This cycle of fetching a new instruction, decoding it and finally executing it continues until the computer is turned off.

1.8 HARVARD ARCHITECTURE

In 1944, Howard Aiken of Harvard University developed a computer (named *Automatic Controlled Calculator* and later *Harvard Mark* I) which used two separate memories, one for program storage (on punched tape) and other for data storage (on relay latches).

The Harvard architecture uses physically separate memories for their instructions and data, requiring dedicated buses for each of them (see Fig. 1.12). Thus instructions and data can be fetched simultaneously.

Program memory and data memory can be of different widths, type etc. Both memories can be accessed at the same time using separate buses. Thus Harvard computers are faster than Von Neumann computers for a given circuit complexity. This architecture has been followed in modern day systems like, Digital Signal Processors (DSP) and Microcontrollers.

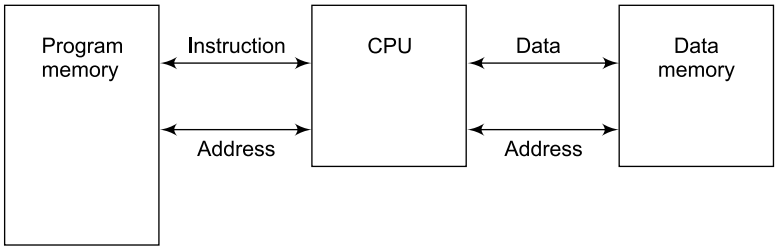


Figure 1.12 *Harvard architecture*

The instruction format of the Harvard Mark I machine was:

$$\text{ADDRESS}_1\ \text{ADDRESS}_2\ \text{OPCODE}$$

where ADDRESS_1 and ADDRESS_2 specified the registers storing the operands while ADDRESS_2 also specified the destination register where the result could be stored. OPCODE specified the operation (add, subtract or multiplication etc) to be performed. The storage had the capacity to store seventy two 23-digit decimal numbers.

SOLVED PROBLEMS

1. *What are the differences between low-level language and high-level language?*

Answer

(a) Low-level languages are closer to the computers, that is low-level languages are generally written using binary codes; whereas the high-level languages are closer to the human, that is these are written using English-like instructions.

(b) Low-level language programs are machine dependent, that is, one program written for a particular machine using low-level language cannot run on another machine. But, high-level language programs are machine independent.

(c) As far as debugging is concerned, high-level programs can be done easily than low-level programs.

(d) It is more convenient to develop application programs in high-level languages compared to the low-level languages.

2. *What are the differences between machine language and assembly language?*

Answer

(a) Machine language instructions are composed of bits (0s and 1s). This is the only language the computer understands. Each computer program can be written in different languages, but ultimately it is converted into machine language because this is the only language the computer understands. Assembly language instructions are composed of text-type mnemonic codes.
(b) Machine language instructions are difficult to understand and debug, since each instruction is only combination of 0s and 1s. However, since assembly language instructions are closer to the human language (i.e. English), it is easy to debug.
(c) In terms of execution, machine language is faster than assembly language. Because for assembly language program, one converter called assembler is needed to convert it into equivalent machine language program; whereas no converter is needed for machine language program.

3. *Differentiate between compilers and interpreters.*

Answer

(a) Compiler is a system program that converts the source program written in a high-level language into corresponding target code in low-level language. This conversion is done by compiler at a time for all instructions. However, the interpreter is a system program that translates each high-level program instruction into the corresponding machine code. Here, in interpreter instead of the whole program, one instruction at a time is translated and executed immediately. Popular compilers are C, C++, FORTRAN, and PASCAL. The commonly used interpreters are BASIC and PERL.
(b) The compilers execute more efficiently and are faster compared to interpreters. Though, the interpreters can be designed easily.
(c) The compilers use large memory space compared to interpreters.

4. *Describe the advantages and disadvantages of assembly language.*

Answer

Advantages of Assembly Language:

The advantage of assembly language over high-level languages is that the computation time of an assembly-language program is less. An assembly-language program runs faster to produce the desired result.

Disadvantages of Assembly Language:

(i) Programming is difficult and time-consuming.
(ii) The assembly language is machine-oriented. The programmer must have a detailed knowledge of the structure of the computer s/he is using. S/He must have the knowledge of registers and instruction set of the computer, connections of ports to the peripherals, etc.
(iii) The program written in assembly language for one computer cannot be used on any other computer, i.e. the assembly-language program is not portable. Each processor has its own instruction set and hence its own assembly language.
(iv) An assembly-language program contains more instructions compared to a high-level language program. Each statement of a program in a high-level language (such as C, FORTRAN, PASCAL, etc.) corresponds to many instructions in an assembly-language program.

5. *Discuss briefly about Princeton architecture and Harvard architecture.*

Answer

Princeton computers are computers with a single memory for program and data storage. The Von-Neumann architecture is also known as Princeton architecture.

Harvard computers are computers with separate program and data memories. Data memory and program memory can be different widths, type etc. Program and data can be fetched in one cycle, by using separate control signals: 'program memory read' and 'data memory read'. Example includes Harvard Mark 1 computer.

6. *What is an Operating System (OS)? Briefly describe the major functions of an OS.*

Answer

An operating system is a collection of programs and utilities, which acts as the interface between the user and computer. The operating system is a system program that tells the computer to do tasks under a variety of conditions. The main objective of an operating system is to create a user-friendly environment.

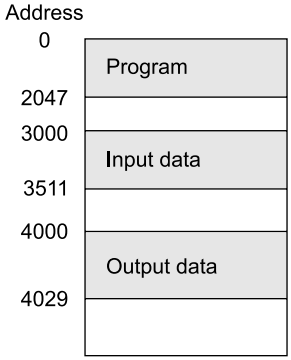
The following are the main functions of operating systems:

1. Managing the user's programs.
2. Managing the memories of computer.
3. Managing the I/O operations.
4. Controlling the security of computer.

7. *Show the addressing for program and data, assuming Von-Neumann architecture for storing the following program:*
   (a) *Assume that a program has a length of 2048 bytes and the program starts from an address 0.*
   (b) *The input data size is 512 bytes and stores from 3000.*
   (c) *The results of 30 bytes generated after program execution are stored at address 4000.*

Answer

The figure below shows the addressing of program and data.



Addressing of stored program in von Neumann architecture

8. *What is Von Neumann bottleneck? How can this be reduced?*

Answer

Since the CPU has much higher speed than the main memory (RAM), the CPU has to wait longer to obtain a data-word from the memory. This CPU-memory speed disparity is referred to as *Von-Neumann bottleneck*.

This performance problem is reduced by using a special type fast memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required data-word to come. Another way to reduce the problem is by using special type computers known as *Reduced Instruction Set Computers* (*RISC*). The intension of the RISC computer is to reduce the total number of the memory references made by the CPU; instead it uses large number of registers for same purpose.

9. *Why does increasing the amount of data that can be stored in a processor's register file (i.e. collection of registers) generally increase the performance of the processor?*

Answer

The registers are very fast and are logically placed inside the processors. Thus, accessing data in registers are faster than accessing data in the memory. Hence, by providing more data in the register file allows more data to be accessed at this faster speed, improving performance.

10. *What is the merit and demerit in using a single I/O bus for all the devices connected to a given system?*

Answer

Merit: The use of single bus means less complexity in system design. A single bus allows many devices to interface to it without requiring that the system designers provide separate interfaces for each device.

Demerit: The use of single bus reduces the bandwidth (i.e. speed of I/O operation). The several devices attached to the single bus have to share the total possible bandwidth of the bus, limiting the performance.

11. *Assume that an LSI IC at semiconductor memory stores 1024 bits. Assume a main memory unit of 1024 ICs. How many bytes do these ICs store?*

Answer

Since an LSI IC at semiconductor memory stores 1024 bits, the main memory unit of 1024 ICs stores 1024×1024 bits = $(1024 \times 1024)/8$ bytes = 128 KB.

12. *How does a multiprogramming system give the illusion that multiple programs are running on the machine simultaneously? What factor can cause this illusion to void?*

Answer

In multiprogramming system, the processor switches among multiple programs executing them very frequently—50 or more times per second. If the number of programs executing in the system is relatively small, each program will get a chance to execute often enough that the system looks like processor is executing all of the programs at the same time. However, if the number of programs executing on the system gets too large, the processor will be busy in context switching (i.e. transferring control among multiple programs) and thus execution of programs will get reduced, making the illusion void.

13. *Suppose a 600 MHz machine does the number of context switching 60 times per second. How many cycles are there in each time-slice?*

Answer

600 MHz = 600×10^6 cycles per second.
Therefore, the number of cycles per time-slice = $(600 \times 10^6)/60 = 10^7$.

14. *To achieve a speed-up of 4 on a program that originally took 80 ns to execute, what must be the execution time of the program be reduced to?*

Answer

$$\text{Speed-up} = \frac{\text{Execution time}_{\text{before}}}{\text{Execution time}_{\text{after}}}$$

Now, we have speed-up = 4 and execution $\text{time}_{\text{before}}$ = 80 ns.
Therefore, from the speed-up formulae, we get execution $\text{time}_{\text{after}}$ = 20 ns.
So, to achieve the speed-up of 4, execution time must be reduced to 20 ns.

REVIEW QUESTIONS

Group-A

1. Choose the most appropriate option for the following questions:
   (i) CPU consists of
      (a) main memory and ALU
      (b) main memory, ALU and control unit
      (c) cache memory, ALU and control unit
      (d) ALU, control unit and registers.
   (ii) Control unit is
      (a) a logic unit to provide control signals for different units
      (b) used to control input-output from a processor
      (c) used to control the CPU
      (d) used to fetch the instruction from memory.
   (iii) A compiler is
      (a) an application program
      (b) a system program used to convert a high-level program to low-level program
      (c) a part of operating system
      (d) a system program used to convert an assembly language program to low-level program.
   (iv) An example of popular batch processing operating system is
      (a) Windows 98
      (b) MS-DOS
      (c) UNIX
      (d) Windows 2000.
   (v) Which of the operating systems supports multiple CPUs through shared main memory?
      (a) multiprogramming OS
      (b) real-time OS
      (c) distributed OS
      (d) multiprocessing OS.

(vi) BIOS is
  (a) a collection of I/O driver programs
  (b) part of OS to perform I/O operations
  (c) firmware consisting of I/O driver programs
  (d) a program to control one of the I/O peripherals.

(vii) Babbage's difference engine is a computer
  (a) for subtraction
  (b) for both addition and subtraction
  (c) for performing multi-step operations automatically
  (c) for arithmetic and logical operations.

(viii) Stored program computers
  (a) store the program and data in the same memory
  (b) store the program and data in two separate memories
  (c) store programs in memory
  (d) store the program and data in the same address of memory.

(ix) Stored program computers
  (a) require a permanent memory
  (b) need small memory
  (c) allow self modifying programs
  (d) do not treat themselves as data.

(x) IAS computer introduced the concept of
  (a) ALU, main memory, PC, AC and multiplier quotient (MQ) registers for executing instructions as well as stored program
  (b) ALU, AC and MQ for executing an instruction as well as stored program
  (c) decimal, fixed-point and floating point ALUs with stored program
  (d) ALU, main memory, PC and subroutine calls.

(xi) The Von-Neumann bottleneck is a problem, which occurs due to
  (a) small size main memory
  (b) high-speed CPU
  (c) speed disparity between CPU and main memory
  (d) malfunctioning of any unit in CPU.

(xii) Where does the control unit look in order to find the address of the next instruction to be fetched?
  (a) Memory address register (WAR)
  (b) Instruction register (IR)
  (c) Memory buffer register (MBR)
  (d) Accumulator (AC)

(xiii) Any computer must at least consist of
  (a) data bus
  (b) address bus
  (c) control bus
  (d) all of the above.

(xiv) Virtually all computer designs are based on the Von Neumann architecture. A high level view of this architecture has the following three components:
  (a) Buses, memory, input/output controllers

(b) Hard disks, floppy disks, and the CPU
(c) Memory, the CPU, and printers
(d) Memory, input/output modules, and the CPU.

(xv) Which of the following programming languages has an instruction set closest to the machine language of a computer?
(a) BASIC
(b) Fortran
(c) Assembly language
(d) C++

(xvi) An operating system that allows several processors to perform computation as the same time is called
(a) single program
(b) multitasking
(c) multiprocessing
(d) real-time processing.

(xvii) The fundamental conceptual unit in a computer is
(a) CPU
(b) hard drive
(c) operating system
(d) transistor.

(xviii) A computer that is advertised as having a 96K byte DRAM memory and a 2.1 Gigabyte hard drive has
(a) 96 K bytes of primary memory and 2.1 gigabytes of secondary memory
(b) 2.1 gigabytes of primary memory and 96 K bytes of secondary memory
(c) 96 bytes of cache, 2.1 gigabytes of primary memory
(d) 96 K bytes of cache, 96 K bytes of primary memory, and 2.1 gigabytes of secondary memory.

(xix) Machine language and assembly language programming are
(a) preferred because machine codes can be run fast by the processor
(b) not preferred because larger memory is needed for a program
(c) preferred over high-level languages for correct development of a program
(d) not preferred over high-level languages for fast development of a program.

(xx) The unit which performs the tasks of fetching, decoding and managing the execution and then storing the results is
(a) ALU (b) CU (c) Register file (d) Memory unit.

Group-B

2. What do you mean by a computer system? Describe the functions of different hardware components.
3. Classify the types of software in general. Discuss each of them in brief.
4. Define: operating system and compiler. Briefly discuss about different operating systems with examples.
5. What is BIOS? What is bootstrap loader?
6. Distinguish between compiler and interpreter with examples.
7. Describe the functions of OS?
8. Describe different types of external memories.
9. Discuss about different types computers: mainframe, mini, micro and supercomputer.

10. What is the stored program computer? Discuss about its instruction and data formats.
11. Describe the structure of IAS computer.
12. Write short notes on: Von Neumann (IAS) computer and its bottleneck.
13. What are the basic stages of the fetch-execute cycle?
14. List two registers that are critical to the fetch-execute cycle.
15. Which computer components are important in the fetch-execute cycle? For each, state the reasons why.
16. Can any computer component become involved in the fetch-execute cycle?
17. Describe what happens when an instruction is fetched.
18. Describe what happens while an instruction is decoded.
19. Describe what happens during the execution of an instruction.
20. Briefly describe Harvard architecture with a diagram.

CHAPTER

2

Data Representation and Computer Arithmetic

2.1 INTRODUCTION

The computers store the binary information needed to perform some operation in memory or processor registers. The binary information can be instruction or data. Instruction is a bit or group of bits used to instruct the computer to execute the program. Data (sometimes called operands) are numbers and other binary-coded information that are operated on by an instruction to achieve required results. In this chapter, we show how different data types are represented in binary coded form in processor registers and the arithmetic operations which are performed on stored data.

2.2 DATA TYPES

Different user (application) programs use different types of data based on the problem. A program can operate either on numeric data or non-numeric data. The different types of non-numeric data are as follows:

- Characters
- Addresses
- Logical data

All non-binary data are represented in computer's memory or registers in the binary coded form.

Character Data A character may be a digit, an alphabet or a special symbol, etc. A character is represented by a group of bits. It includes upper-case and lower-case alphabets (26), decimal numbers (10) and special characters, such as +, @, *, etc. A set of multiple characters usually form a meaningful data. The standard code to represent characters is American Standard Code for Information Interchange (ASCII). This standard uses an 8-bit pattern in which 7 bits specify the character. The 8^{th} bit is generally used as a parity bit for error detection or sometimes it is permanently 1 or 0. Another popular code is Extended Binary Coded Decimal Interchange Code (EBCDIC) used for large computers. The computer systems such as IBM System/360 use this code. It is an 8-bit code without parity.

Addresses The data or operand is often used as an address for some instructions. An address may be a processor register or a memory location from where the required operand value is retrieved for instruction execution. An address is represented by a group of bits. In some instructions, multiple operand addresses are specified for data to be retrieved or stored. The details of this will be discussed in Chapter 5.

2.3 NUMBER SYSTEMS

In our day-to-day life, we are using decimal numbers, which are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. In other words, humans are most accustomed with decimal system. But, a computer can only understand the information composed of 0s and 1s. That means the binary number system is followed by the computers in most natural way. However, sometimes it is necessary to use other number systems, like hexadecimal or octal or decimal systems.

A number in the number system of *base or radix* (r) is represented by a set of symbols from r distinct symbols. The decimal number system uses 10 digits from 0 to 9, thus its base is 10. The binary system uses two distinct digits 0 and 1, thus its base is 2. For octal system (base r = 8), a number is represented by 8 distinct digits 0 to 7. The 16 symbols used in hexadecimal number system (base 16) are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Here, the symbols A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15 respectively.

The value of a number is calculated by summing up all multiplied value of each digit with an integer power of r. For example, the decimal number 8752.4

$$= 8 \times 10^3 + 7 \times 10^2 + 5 \times 10^1 + 2 \times 10^0 + 4 \times 10^{-1}$$

Similarly, the binary number 101101

$$= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Decimal Codes As we know, humans understand decimal system easily and computers process every thing in binary, therefore there must be a conversion system from decimal-to-binary representation in computer's input process. Similarly, binary-to-decimal conversion system must be a part of the computer's output process. These conversions should be performed very rapidly. To facilitate rapid conversions, several number codes exist that encode each decimal separately by a group of bits. These types of codes are called *decimal codes*. Two widely used decimal codes are: *BCD* (binary coded decimal) and *Excess-3 code*.

BCD (Binary Coded Decimal) The BCD is the simplest binary code to represent a decimal number. In BCD code, four bits are used to represent a decimal number. For example, decimal 5 is represented by 0101. When a decimal number consists of more than one decimal digit, each digit is independently represented by its 4-bit binary equivalent. For example, 39 is represented by 0011 1001.

BCD code is weighted (positional) code and weights of four bits which represent an individual digit are 8, 4, 2 and 1. For this reason, the BCD code is sometimes called 8-4-2-1 code. In weighted codes, the bits are multiplied by weights mentioned and addition of the weighted bits gives the decimal digit. For example, the number 1001 in BCD code (8-4-2-1 code) gives the decimal equivalent = $8 \times 1 + 4 \times 0 + 2 \times 0 + 1 \times 1 = 9$.

The code has the advantage of rapid conversion, though it has the disadvantage in forming complements. For example, the 1's complement of 0011 (decimal 3) is 1100 (decimal 12), which is invalid BCD code. To solve this problem, another decimal code called excess-3 code is used.

Excess-3 Code In this code, 0011 (decimal 3) is added to each BCD (decimal digit) of a number. For example, 0110 (decimal 6) = 1001 (decimal 9) in excess-3 code. The excess-3 code for 435 is 0111 0110 1000. This code has been used in some older computers. The disadvantage of this code is that it is not a weighted code that means the sum of weights of bits is not equal to the corresponding decimal digit.

2.4 COMPLEMENTS OF NUMBERS

Before going to discuss the numerical data representation, we have to know about the complements of a number, since complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. Complements of numbers in base/radix r system are of two types: the r's complement and (r-1)'s complement. Thus for binary system, there are 2's and 1's complement. For decimal system, complements are 10's and 9's complements.

(r-1)'s Complement For a number N having m digits in base r, the (r – 1)'s complement of N is defined as $(r^m - 1) - N$. In case of decimal numbers, r = 10 and r – 1 = 9, so the 9's complement of N is $(10^m - 1) - N$. Now, we know that 10^m = 1000…0 (m 0s) in decimal. Therefore, $10^m - 1$ is equivalent to 99…9 (m 9s) in decimal. For example, m = 5, we have 10^5 = 100000 and $10^5 - 1$ = 99999. It infers that the 9's complement of a decimal number is obtained by subtracting each digit from 9. For example, the 9's complement of 35367 is 99999 – 35367 = 64632.

In case of binary number system, r = 2 and r–1 = 1, so the 1's complement of N is $(2^m - 1) - N$. Again, we have, 2^m = 1000…0 (m 0s) in binary. Thus, $2^m - 1$ equals to 111…1 (m 1s) in binary. For example, m = 5, we have $2^5 = (100000)_2$ and $2^5 - 1 = (11111)_2$. Thus the 1's complement of a binary number is obtained by subtracting each bit from 1. However, the subtraction of a bit from 1 means the bit to change from 1 to 0 and 0 to 1. Hence, the 1's complement of a binary number is obtained by replacing 1s into 0s and 0s into 1s. For example, the 1's complement of a binary number 101101 is 010010.

Similarly, the (r–1)'s complement of numbers in other systems like octal and hexadecimal can be formulated.

r's Complement The r's complement of a number N having m digits in base r is defined as $r^m - N$; if N ≠ 0 and 0; if N = 0. Note that, $r^m - N = [(r^m - 1) - N] + 1$. Thus, the r's complement of a number is obtained by adding 1 to the (*r*–1)'s complement of the number. For example, the 10's complement of a decimal number 43678 is 56321 + 1 = 56322 and the 2's complement of a binary number 101101 is 010010 + 1 = 010011. An m-bit 2's-complement number system can represent every integer in the range -2^{m-1} to $+2^{m-1} - 1$. Also, note that, the complement of a complement of a number is the original number. The r's complement of N is $r^m - N$. The complement of the complement is $r^m - (r^m - N) = N$, which is the original number. Now, we will see the application of using r's complement in subtraction of unsigned numbers.

Subtraction of Unsigned Numbers

For subtraction, the borrow method is used in real life. In this method, when the minuend digit is smaller than the corresponding subtrahend digit, we borrow a 1 from the next higher significant position. This method is popularly used in school level mathematics. This method is found to be

complex than the method that uses complements, when subtraction is to be performed in digital computers. So, computers usually use the method of complements to implement subtraction.

The subtraction of two m-digit unsigned numbers A – B (B ≠ 0) in base r can be performed using the rule: The minuend A is added with the r's complement of the subtrahend B, which gives A + (r^m – B) = (A – B) + r^m.

Now, following two cases may arise.

Case-1: When A ≥ B

The addition gives an end carry r^m from leading bit position (most significant position), which is to be discarded and the rest is the result A – B.

Case-2: When A < B

The addition does not give any end carry and the addition is equal to r^m – (B – A), which is the r's complement of (B – A). Now, to get the result in familiar form, the r's complement of the result is taken and a negative sign is placed in front of the result.

Let's consider the example of subtraction, 45328 – 20342 = 24986. The 10's complement of 20342 is 79658.

A = 45328
10's complement of B = 79658
Sum of these two = 124986
Discard end carry 10^5 = –100000
Thus, answer is = 24986.

Now consider, an example where A < B. The subtraction 20342 – 45328, which will give –24986 as answer, is to be performed.

We have,

A = 20342
10's complement of B = 54672
Sum = 75014

There is no end carry.

The result is – 24986, after taking 10's complement of 75014.

The same way, the subtraction of two unsigned binary numbers can be done.

2.5 BINARY DATA REPRESENTATION

A number can be either unsigned or signed. Unsigned numbers are positive numbers, including zero. So, the unsigned numbers can be represented by its magnitude, there is no need to represent the sign of the numbers. Positive and negative numbers, including zero are treated as signed numbers. In order to differentiate between positive and negative numbers, we need a notation for sign. In real life, a negative number is indicated by a minus sign in leftmost position followed by its magnitude and a positive number by a plus sign in leftmost position followed by its magnitude. But, everything in computers must be represented with 1s and 0s, including the sign of numbers. As a result, it is the common practice to represent the sign of numbers with a bit placed in the leftmost (most significant) position of the number. The convention is that a bit 0 is used as sign bit for positive numbers and 1 for negative numbers.

Moreover, a number may have a radix (binary) point. A number may be a fraction or integer or mixed integer-fraction number, depending on the position of the radix point. Then the natural question comes in mind: where is this radix (binary) point stored in the registers? There are two ways of specifying the position of the binary point in a register: (1) by giving it a fixed position. So, this method used to represent numbers is referred to as fixed-point representation method. (2) by using a floating-point representation. The fixed-point numbers are known as integers whereas floating-point numbers are known as real numbers.

2.5.1 Fixed-Point Number Representation

In a fixed-point representation, all numbers are represented as integers or fractions. The fixed-point method assumes that the radix (binary) point is always fixed in one position. The two widely used positions in register are (1) a binary point in the extreme left of the register to make the stored number a fraction, and (2) a binary point in the extreme right of the register to make the stored number an integer. As we have said everything is represented by 1s and 0s in computers, so in either case, the binary point can not be stored in register; its existence is assumed from the fact of the number's type, viz. whether the number stored in the register is a fraction or an integer. Most of the computers follow the first method (i.e. binary point is in the extreme left of the register).

The positive fixed-point (integer) number is represented by 0 in sign bit position and the magnitude by a positive binary number. For example, +12 is to be stored in an 8-bit register. +12 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 12, i.e. 0 0001100. There is only one way to represent a positive number. However, there are three representations for a negative integer number. The negative number is represented by 1 in the sign bit position and the magnitude of the number is represented in one of three possible ways:

(a) signed-magnitude representation
(b) signed-1's complement representation
(c) signed-2's complement representation

In signed-magnitude representation of a negative number, the number is represented by a 1 in the sign bit position and the magnitude by positive binary number. For other two representations, the negative number is represented by a 1 in the sign bit position and the magnitude by either the 1's complement or 2's complement of its positive value. There are three different methods to represent -12 with 8-bit registers.

In signed-magnitude representation:	1 0001100
In signed-1's complement representation:	1 1110011
In signed-2's complement representation:	1 1110100

In signed-magnitude representation, the range for numbers using n-bit register is: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$. Note that there are two representations of 0 (+0 and −0). +0 has a value of 0 in the magnitude field and sign bit as 0, while − 0 has a value 0 in the magnitude field and sign bit as 1. Also, this method is not suitable for arithmetic operations in computer, as it creates hardware complexity in computers.

The signed-1's complement method has difficulties because it has two different representations of 0, like signed-magnitude method. It is useful as a logic operation since the change of 1 to 0 and 0 to 1 is equivalent to a logical complement operation. Thus, this method is not usually used for arithmetic operation.

In signed-2's complement representation, the range for numbers using n bits is: $-(2^{n-1})$ to $+(2^{n-1}-1)$. This range comes from the fact that there is only one representation for 0, allowing an odd number of non-zero values to be represented. Thus, the negative numbers are represented using only signed-2's complement method.

Arithmetic Operations on Fixed-Point Numbers

Arithmetic Addition The addition of two numbers in the sign-magnitude system is performed by the rules of ordinary arithmetic. This method of addition is quite complex when it is implemented in computers. However, the addition using signed-2's complement method is very simple and can be stated as follows:

The two numbers, including their sign bits, are added and the carry out from the sign (leftmost) position is discarded, if any. Numerical examples are illustrated below, where numbers are stored in 8-bit registers.

+9	00001001		–9	11110111	
+15	00001111		+15	00001111	
+24	00011000		+6	00000110	

+9	00001001
–15	11110001
–6	11111010

Negative results are automatically in 2's complement form. For example in last case, the signed binary number 11111010 is a negative number because the leftmost bit is 1. So, its magnitude's (7-bits: 1111010) 2's complement is 0000110. That is the binary equivalent of +6. Therefore, the result is –6.

Arithmetic Subtraction Like the arithmetic addition, the subtraction of two numbers in the signed-magnitude system is performed by the rules of ordinary arithmetic and it is quite complex in implementing in computers.

But, subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated as follows:

The 2's complement of the subtrahend, including the sign bit is added with the minuend, including the sign bit. The carry out from the sign bit position is discarded, if any. The subtraction can be summarized with the relation:

$$A - B = A + \text{1's complement of } B + 1.$$

This method of subtraction is obtained from the fact that the subtraction can be converted into addition, if sign of subtrahend is reversed. In other words,

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

Consider the subtraction (–9) – (–15) = +6. For 8-bit registers, this can be written as 11110111 + 00001111 = 100000110. The correct answer is 00000110 (i.e., +6), after removing the end carry.

Note that the addition and subtraction of binary numbers in the signed-2's complement system are performed by the same basic addition and subtraction rules as unsigned numbers. Moreover, the addition and subtraction of binary numbers are performed by only the addition operation. Thus, computers need only one common additive hardware circuit to handle two arithmetic operations.

Overflow in Fixed-Point Representation An overflow is a problem in digital computers because the numbers are stored in registers, which are finite in length. If two numbers of n digits each are added and the sum occupies n + 1 digits, an overflow will occur. This holds good irrespective of the numbers' type. Since a register of n-bit can not accommodate the result of n + 1 bits, an overflow results. If it occurs, a corresponding flip-flop in CPU is set, which is then verified by the user or program.

If one number is positive and the other is negative, after an addition overflow cannot occur, since addition of a positive number to a negative number produces a number that is always smaller than the larger of the two original numbers. However, an overflow may occur if the two numbers added are of same sign i.e., both are positive or both are negative. Let's consider following examples.

Carries:	01		Carries:	10	
+69	0	1000101	–69	1	0111011
+78	0	1001110	–78	1	0110010
+147	1	0010011	–147	0	1101101

Observe that the 8-bit result that should have been positive (first example) has a negative sign bit and the 8-bit result that should have been negative (second example) has a positive sign bit. However, if the carry out from the sign bit position is treated as the sign of the result, the 9-bit answer thus obtained will be correct answer. Since the 9-bit answer cannot be accommodated with 8-bit register, we say that an overflow results.

To detect an overflow condition the carry into the sign bit position and the carry out from the sign bit position are examined. If these two carries are both 0s or both are 1s, there is no overflow. If these two carries are not equal (i.e., if one is 0 and other is 1), an overflow condition exists. This is illustrated in the examples where the two carries are explicitly shown. Using an XOR gate (For detail, see Appendix), whose two inputs are these carries, an overflow can be detected when the output of the gate is 1.

2.5.2 Floating-Point Representation

In scientific applications of computers, fractions are frequently used. So, a uniform system of representation is required which automatically keeps track of the position of the radix (binary) point. Such a system of representation is called floating-point representation of numbers.

In floating-point representation, a number has two parts. The first part is called *mantissa or fraction*, to represent a signed fixed-point number, which may be a fraction or an integer. The second part is called *exponent or characteristic*, to designate the position of the radix point. For example, in floating-point representation, the decimal number +786.231 is represented with a mantissa and an exponent as follows:

Mantissa	Exponent
+ 0.786231	+ 03

This representation is equivalent to the scientific notation $+0.786231 \times 10^{+03}$.

In floating-point representation, a number is always assumed to interpret a number in the following scientific notation:

$$\pm M \times r^{\pm E}$$

Out of three (mantissa M, radix r and exponent E), only the mantissa M and the exponent E are physically present in the register, including their sign bits. The radix r is not present, but its presence and the position of it in the mantissa are always assumed. Most computers use fractional system of representation for mantissa, but some computers use the integer system for mantissa.

If the integer system of representation for mantissa is used, the decimal number +786.231 is represented in floating-point with a mantissa and an exponent as follows:

Mantissa	Exponent
+ 786231	–03

The floating-point binary number is also represented in the same fashion. For example, the binary number + 10011.1101 can be represented with 12-bit mantissa and 6-bit exponent as below.

Mantissa	Exponent
0 10011110100	0 00101

The mantissa has a leading 0 to indicate positive sign and the exponent has representation of + 5.

When the most significant digit (left most) of the mantissa is nonzero, the floating-point number is said to be normalized. For example, 0.00386×10^4 is not a normalized number. The corresponding normalized number is 0.386×10^2. Similarly, the 8-bit binary number 00011100 is not normalized because of the three leading 0s. The number can be normalized by shifting it three positions to the left and leaving out the leading 0s to obtain 11100000. But, the left shifting three positions means multiplication of the original number with $2^3 = 8$. So, to retain the value of number same, the exponent must be subtracted by 3. The maximum possible precision for the floating-point number is achieved by normalization. Another advantage of using normalized floating point numbers is increased coverage of numbers. If a computer uses all numbers as normalized, then one bit position can be saved by omitting the most significant position, which is always 1. This 1 is called *hidden* 1. It should be noted that a zero cannot be normalized because it does not have a nonzero digit.

Floating-point representation is more useful than fixed-point representation, in dealing with very large or small numbers. Moreover, the floating-point representation is more accurate in arithmetic operation.

IEEE Standard for Floating-Point Numbers Initially, different computer manufacturers were using different formats for the floating-point representation. The most common ways of representing floating-point numbers in computers are the formats standardized as the IEEE (Institute of Electrical and Electronics Engineers) 754 standard, commonly called "IEEE floating-point". These formats can be manipulated efficiently by nearly all modern floating-point computer hardware. It has two similar formats as follows:

1. Single Precision Format It is 32-bit format, in which 8-bit is for exponent, 23-bit for mantissa, 1-bit for sign of the number, as shown in Fig. 2.1. Here, the implied base 2 and original signed exponent E are not stored in register. The value actually stored in the exponent field is an unsigned integer E′ called *biased exponent*, which is calculated by the relation $E' = E + 127$. This is referred to as the *excess*-127 format. Thus, E′ is in the range $0 \le E' \le 255$. The end values of this range, 0 and 255, are used to represent special values. Therefore, the range of E′ is $1 \le E' \le 254$, for normal values. This means that the actual exponent (E) is in the range $-126 \le E \le 127$.

The lower-order 23 bits represent the mantissa. Since binary normalization is used, the most significant bit of the mantissa is always set to 1. This bit is not explicitly represented; it is assumed to

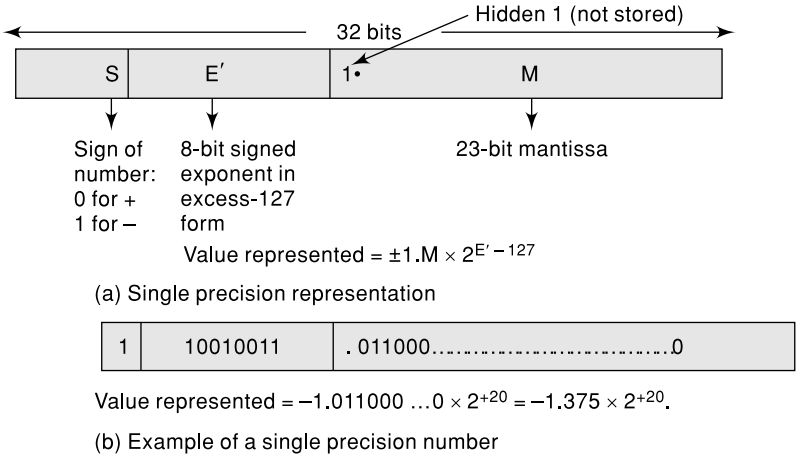


Figure 2.1 *IEEE Single precision format*

be to the immediate left of the binary point. This bit is called hidden 1. Thus, the 23 bits stored in the mantissa M field actually represent the fractional part of the mantissa, that is, the bits to the right of the binary point. The 1-bit sign field S contains 0 for positive and 1 for negative number.

2. Double Precision Format This is 64-bit format in which 11-bit is for biased exponent E′, 52-bit for mantissa M and 1-bit for sign of the number, as shown in Fig. 2.2. The representation is same as single precision format, except the size and thus other related parameters.

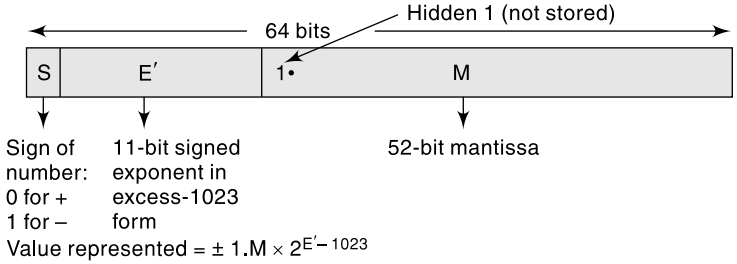


Figure 2.2 *IEEE double precision representation*

Special Values

Here we consider IEEE single precision (i.e., 32-bit) format. The end values 0 and 255 of the biased exponent E′ are used to represent special values.

Case –1: When $E' = 0$

If the mantissa $M = 0$, the exact 0 value is represented.

If the mantissa $M \neq 0$, *denormal* numbers are represented. Their value is $\pm 0.M \times 2^{-126}$. Thus, they are smaller than the smallest normal number. The mantissa part of the number $0.M$ has a leading 0 instead of the usual leading 1. That is, M is any nonzero 23-bit fraction. The purpose of introducing denormal numbers is to allow for *gradual underflow*, providing an extension of the range of normal representable numbers that is useful in dealing with very small numbers in certain situations.

Case- 2: When $E' = 255$

If the mantissa $M = 0$, the value *infinity* (∞) is represented. The infinity is the result of dividing a normal number by 0.

If the mantissa $M \neq 0$, the result represented is called *Not a Number* (NaN). A NaN is a result of performing an invalid operation such as dividing 0 by 0 or $\sqrt{-1}$.

Similarly, for double precision (i.e., 64-bit) format, the end values 0 and 2047 of the biased exponent E' are used to represent special values.

Example 2.1 Represent the binary positive number 1101011 in the IEEE single-precision format.
The binary positive number 1101011 = + 1.101011×2^6
The 23-bit mantissa M = 0.101011 000000 000000 00000
The biased exponent E′ = E + 127 = 6 + 127 = 133 = 1000 0101
The sign bit S = 0, since the number is positive.
Therefore, the IEEE single-precision (32-bit) representation is:

0	1000 0101	101011 000000 000000 00000

Example 2.2 Represent the decimal number – 0.75 in the IEEE single-precision format.
The decimal number – 0.75 = – 0.11 in binary = -1.1×2^{-1}
The 23-bit mantissa M = 0.100000 000000 000000 00000
The biased exponent E′ = E + 127 = –1 + 127 = 126 = 0111 1110
Since the number is negative, the sign bit S = 1
Therefore, the IEEE single-precision (32-bit) representation is:

1	0111 1110	100000 000000 000000 00000

Arithmetic Operations on Floating-point Numbers

Add/Subtract Operation The rule for the operation is summarized below:

Steps

1. Choose the number with the smaller exponent and shift its mantissa right a number of positions equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition/subtraction on the mantissas and determine the sign of the result.
4. Normalize the result, if necessary.

Multiplication Operation The rule for the operation is summarized below (based on IEEE Single-precision representation):

Steps

1. Add the exponents and subtract 127.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the result, if necessary.

Division Operation The rule for the operation is summarized below (based on IEEE Single-precision representation):

Steps

1. Subtract the exponents and add 127.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the result, if needed.

Overflow and Underflow in Floating-Point Representation If the result of an arithmetic operation on floating-point numbers is too large or too small to be stored in computer, an overflow or underflow may result. If two floating-point numbers of the same sign are added, a carry may be generated from the high-order bit position (most significant bit position) we call it as *mantissa overflow*, as this extra bit cannot be accommodated in the allotted mantissa field. Such overflow can be corrected easily by shifting the sum one position to the right and thus incrementing the exponent. When two floating-point numbers are multiplied, the exponents are added. Some times, the sum of the exponents may be very large and all bits of the sum result can not be stored in the allotted exponent field. This is called *exponent overflow*. This type overflow can not be corrected and hence an error signal is generated by the computer.

Similarly, when two floating-point numbers are subtracted, there may be at least one 0 in the most significant position in the mantissa of the result. Then the resultant mantissa is said to be in *underflow* condition. This condition can again be corrected by shifting the result to the left and decrementing the exponent until a non-zero bit appears in the left-most position in the mantissa. In case of division of two numbers, the exponent of the divisor is subtracted from the exponent of the dividend. The subtraction result may be too small to be represented. This is called *exponent underflow*. Like exponent overflow, this problem can not be solved and thus an error signal is generated by the computer.

2.6 GUARD BITS AND TRUNCATION

When the mantissa is shifted right, some bits at the right most position (least significant position) are lost. In order to obtain maximum accuracy of the final result; one or more extra bits known as *guard bits*, are included in the intermediate steps. These bits temporarily contain the recently shifted out bits from the right most side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are not stored. However, based on the guard bits, the value of the mantissa can be made more precise by the rounding (truncation) technique.

The truncation of a number involves ignoring of the guard bits. Suppose n = 3 bits are used in final representation of a number, n = 3 extra guard bits are kept during operation. By the end of the operation, the resulting 2n = 6 bits need to be truncated to n =3 bits by one of the following three methods. In all cases truncation error exists, which is E = actual value – truncated value.

Chopping In this method, simply all n = 3 guard bits are dropped. All fractions in the range 0. b_{-1} b_{-2} b_{-3} 000 to 0. b_{-1} b_{-2} b_{-3} 111 are truncated to 0. b_{-1} b_{-2} b_{-3}. The truncation error of chopping is $0 \le E \le 0.000111 < 0.001 = 2^{-n}$. Since E is always greater than 0, we say this truncation error is *biased*.

Von Neumann Rounding If at least one of the guard bits is 1, the least significant bit of the retained bits is set to 1, no matter whether it is originally 0 or 1; otherwise nothing is changed in retained bits and simply guard bits are dropped. Two worst cases may arise (for b_{-3} = 1 and 0) when at least one of the guard bits is 1.

Case-1: The number 0. b_{-1} b_{-2} 1111 is truncated to 0. b_{-1} b_{-2} 1. The truncation error is E = 0. b_{-1} b_{-2} 1111 – 0. b_{-1} b_{-2} 1 = 0.000111 < 0.001 = 2^{-n} .

Case-2: The number 0. b_{-1} b_{-2} 0001 is truncated to 0. b_{-1} b_{-2} 1. Then the truncation error is E = 0. b_{-1} b_{-2} 0001 – 0. b_{-1} b_{-2} 1 = – 0.000111 > – 0.001 = – 2^{-n}.

Both cases can be summarized as | E | < 2^{-n}. Thus the Von Neumann rounding error is *unbiased*, because the range of error is symmetrical about 0.

Rounding Here, the truncation of the number is done according to the following rules.

Rule-1: If the highest guard bit is 1 and the rest guard bits are not all 0s, a 1 is added to the lsb position of the bits retained. Thus, 0. b_{-1} b_{-2} b_{-3} 1xx is rounded to 0. b_{-1} b_{-2} b_{-3} + 0.001. The error in this case is E = 0. b_{-1} b_{-2} b_{-3} 1xx – (0. b_{-1} b_{-2} b_{-3} + 0.001) = 0.0001xx – 0.001 = –(0.001 – 0.0001xx) > – 0.0001 = – $2^{-(n+1)}$.

Rule-2: If the highest guard bit $b_{-(n+1)}$ is 0, drop all guard bits. Thus, the number 0. b_{-1} b_{-2} b_{-3} 0xx is rounded to 0. b_{-1} b_{-2} b_{-3}. The truncation error E is = 0. b_{-1} b_{-2} b_{-3} 0xx – 0. b_{-1} b_{-2} b_{-3} = 0.0000xx > – 0.0001 = – $2^{-(n+1)}$.

Rule-3: If the highest guard bit is 1 and the rest guard bits are all 0s, the rounding depends on the lsb, $b_{-n} = b_{-3}$.

If lsb = 0, the number 0. b_{-1} b_{-2} 0100 is truncated to 0. b_{-1} b_{-2} 0. The error in this case is E = 0. b_{-1} b_{-2} 0100 – 0. b_{-1} b_{-2} 0 = 0.0001 = $2^{-(n+1)}$.

If lsb = 1, the number 0. b_{-1} b_{-2} 1100 is truncated to 0. b_{-1} b_{-2} 1 + 0.001. The truncation error E is = 0. b_{-1} b_{-2} 1100 – (0. b_{-1} b_{-2} 1 + 0.001) = – 0.0001 = – $2^{-(n+1)}$.

The value represented by guard bits is 0.5 * 2^{-n}, it is randomly rounded either up or down with equal probability (50%). The rounding error of these cases can be summarized as | E | ≤ $2^{-(n+1)}$. Thus, the rounding error is unbiased.

Comparing the three rounding methods, we see that the rounding technique has the smallest unbiased rounding error, but it requires most complex and costly hardware.

2.7 MULTIPLICATION OF UNSIGNED AND SIGNED INTEGERS

Two unsigned integers can be multiplied the same way as two decimal numbers by manual method. Consider the multiplication of two unsigned integers, where the multiplier Q =12 = $(1100)_2$ and the multiplicand M = 5 = $(0101)_2$ as illustrated in Fig. 2.3.

```
M ⇒   0101
Q ⇒   1100
      0000
     0000
    0101
   0101          } Partial Products
-------
  0111100        Final Product
```

Figure 2.3 *Paper and pencil method (Manual method)*

2.7.1 Array Multiplier

In the paper and pencil (manual) method, shifted versions of multiplicands are added. This method can be implemented by using AND gates and full adders (for full adder, see Appendix). The hardware realization of this method for 4-bit multiplier Q and 4-bit multiplicand M is shown in Fig. 2.4. The basic combinational cell used in the array as the building block handles one bit of the partial product, as shown in Fig. 2.4(a). If the multiplier bit Q_i is 1, then this cell adds an incoming partial product bit to the corresponding multiplicand bit M_i. Each row of the array adds the multiplicand appropriately left shifted to the incoming

partial product PP_i to generate outgoing partial product PP_{i+1}. If the multiplier bit Q_i is 0, PP_i is passed down vertically as PP_{i+1} with a physical right shift. Initial partial product PP_0 is all 0s and PP_4 is the desired 8-bit product $M \times Q = Pr_7\ Pr_6\ Pr_5\ Pr_4\ Pr_3\ Pr_2\ Pr_1\ Pr_0$.

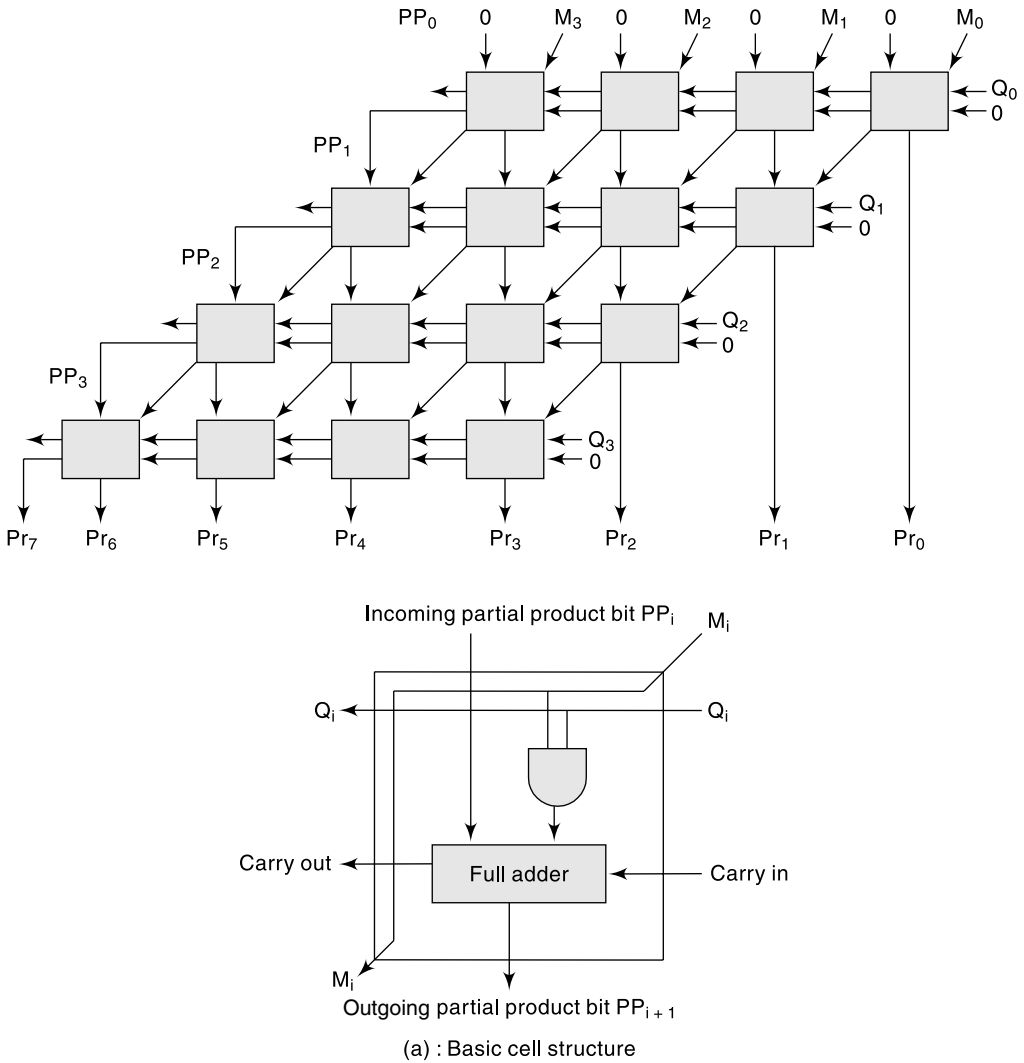


Figure 2.4 *4 × 4 array multiplier*

2.7.2 Sequential Multiplication Method for Unsigned Numbers

Here, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative position. When the corresponding bit of the multiplier is 0, there is no need to add all zeroes to the partial product since it will not alter its value. An n × n unsigned multiplier has three n-bit registers, A, M and Q. The

multiplication method is described in the flowchart shown in Fig. 2.5. The A register, called the accumulator, is initialized to 0. The Q register is initially set to the multiplier value.

When the algorithm is terminated, the A register holds the high-order n bits, and the Q register holds the low-order n bits of the product. The M register always holds the multiplicand. The F flip-flop holds the end carry generated in the addition. This flip-flop F is used as the serial input, when the register pair AQ is shifted right one position.

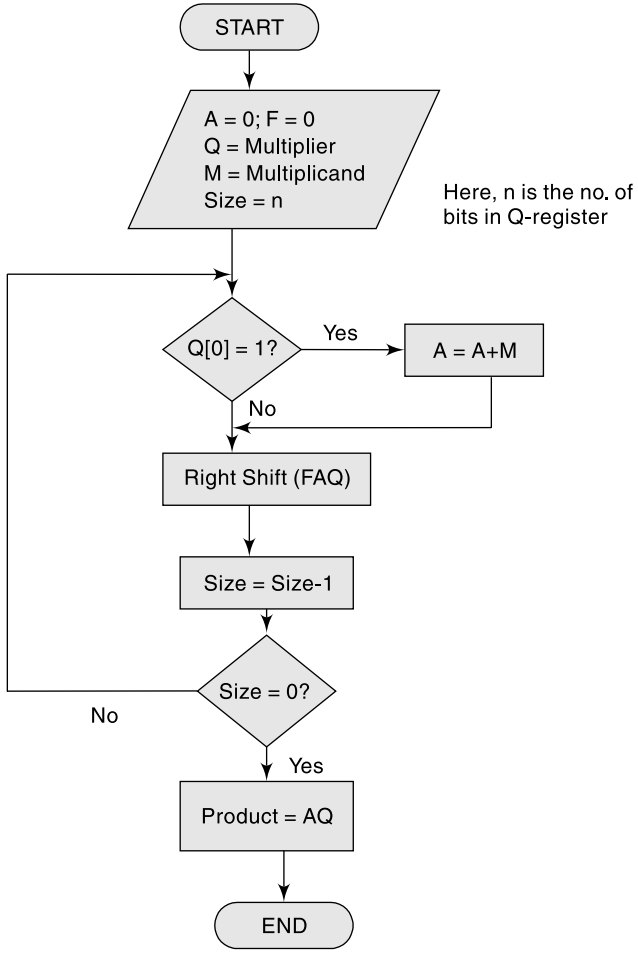


Figure 2.5 *Sequential multiplication method*

Example 2.3 To illustrate this method, consider, the multiplier Q = 14 = 1110_2 and the multiplicand M = 6 = 0110_2.

	M	F	A	Q	Size
Initial Configuration	0110	0	0000	1110	4

Step-1					
As Q[0]=0					
R.S.(FAQ)	0110	0	0000	0111	3
Size – –					
Step-2					
As Q[0]= 1					
A = A + M	0110	0	0110	0111	—
RS(FAQ)	0110	0	0011	0011	2
Size – –					
Step-3					
As Q[0]=1					
A = A + M	0110	0	1001	0011	—
RS(FAQ)	0110	0	0100	1001	1
Size – –					
Step-4					
As Q[0]=1					
A= A + M	0110	0	1010	1001	—
RS(FAQ)	0110	0	0101	0100	0
Size – –					

Since the size register is currently 0, the algorithm is terminated and the final product is = AQ = 0101 $0100_2 = 84_{10}$.

This method of multiplication is good for unsigned number multiplication. In case of signed number multiplication, the signs of the operands can be treated separately and the multiplication of magnitudes of the numbers can be processed using the above method. The sign of the product is determined as $M_n \oplus Q_n$, where M_n, Q_n are the signs of the multiplicand (M) and the multiplier (Q) respectively.

2.7.3 Booth's Multiplication Procedure (for Signed Numbers)

As we have seen lastly, the multiplication of signed numbers in sequential multiplication method requires extra processing steps besides the main multiplication for the magnitude. This is an overhead when operands are denoted in signed 2's complement form. The overhead can be eliminated by a specific mapping rule, called the *recoded multiplication technique*, in which the multiplier is mapped in accordance with the recoding technique. The basis of the recoding technique is the property, called *string property*. This states that "a block of consecutive k 1s in a binary sequence of multiplier may be replaced with a block of k – 1 consecutive 0s surrounded by the digits 1 and $\bar{1}$".

For example, consider the following multiplier:

0011110 (equivalent decimal is 30).

By the string property, it may be considered as the difference between 0100000 (decimal 32) and 0000010 (decimal 2). The multiplication by 0011110 can be achieved by summing up the following two products:

(a) 2^5 times the multiplicand.

(b) 2's complement of 2^1 times the multiplicand.

In sequential multiplication method, four additions are required due to the string of four 1s. This can be replaced by one addition and one subtraction. This is one significant advantage of Booth's multiplication method over sequential multiplication method. The recoding rule on multiplier can be summarized as below:

Step-1: Start searching 1 from lsb (right most bit). Skip all 0s and continue the search till first 1 encountered.

Step-2: Change the first 1 in multiplier as $\bar{1}$.

Step-3: Search for a 0 one by one without disturbing all succeeding 1s; just recode them (1s) as 0s. When a 0 is encountered, change this 0 as 1.

Step-4: Proceed to look for next 1 without disturbing 0s and continue using steps 2 and 3.

Table 2.1 *Booth's Recoding Rule*

A_i	A_{i-1}	*Recoding (change) for* A_i	*Remarks on multiplier*
0	0	$\bar{0}$	Sequence of 0s
1	0	$\bar{1}$	Start of sequence of 1s
1	1	0	Sequence of 1s
0	1	1	End of sequence of 1s

Example 2.4 Original number = 0011110 = 30
Recoded form = 01000 $\bar{1}$ 0 = (0 + 32 + 0 + 0 + 0 – 2 + 0) = 30.

Based on this recoding rule, the Booth's algorithm for multiplication can be developed easily. The algorithm inspects two lower-order multiplier bits at time to take the next step of action. The algorithm is described by the flowchart in Fig. 2.6. A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contain the product. Ignore the right end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier. The circuit block diagram of the Booth's multiplication algorithm is shown in Fig. 2.7.

Example 2.5 To see how this procedure works, the following example is considered. M = – 6 = 1010 and Q = 7 = 0111.

	M	A	Q	Size
Initial Configuration	1010	0000	0111 0	4
Step-1 As Q[0] = 1 and Q[–1]=0				
A=A – M	1010	0110	0111 0	—
And ARS(AQ)	1010	0011	0011 1	3

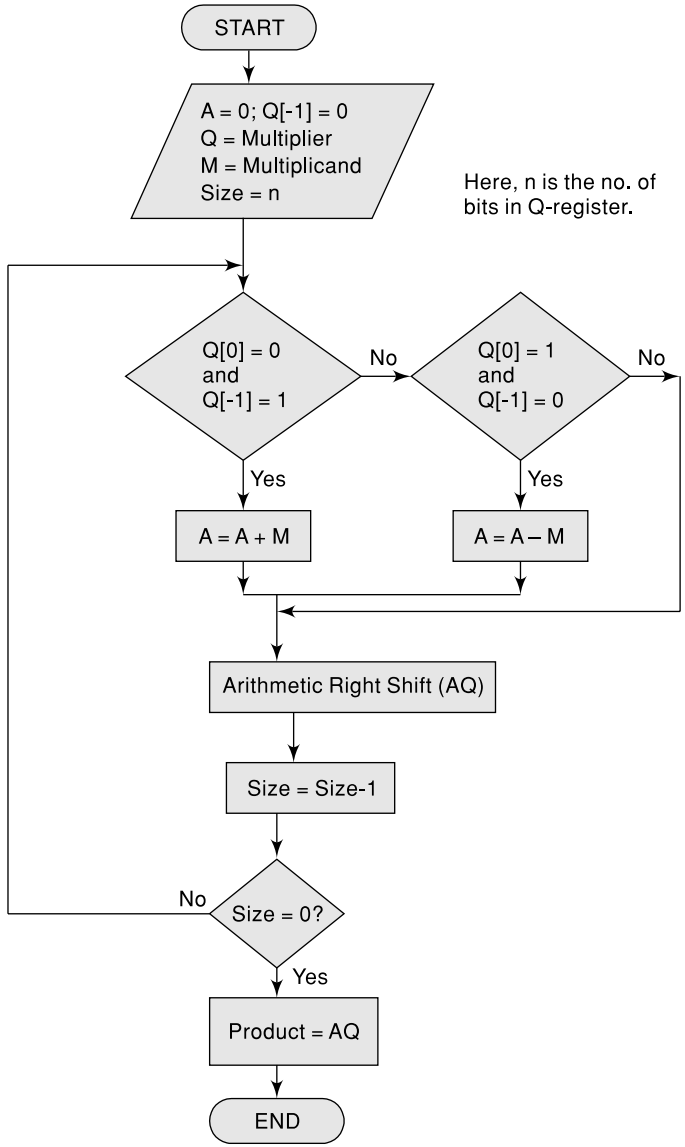


Figure 2.6 *Booth's multiplication algorithm*

Step-2

As Q[0]=1 and Q[–1]=1

ARS(AQ)	1010	0001	1001 1	2

Step-3

As Q[0]=1 and Q[–1]=1

ARS(AQ)	1010	0000	1100 1	1

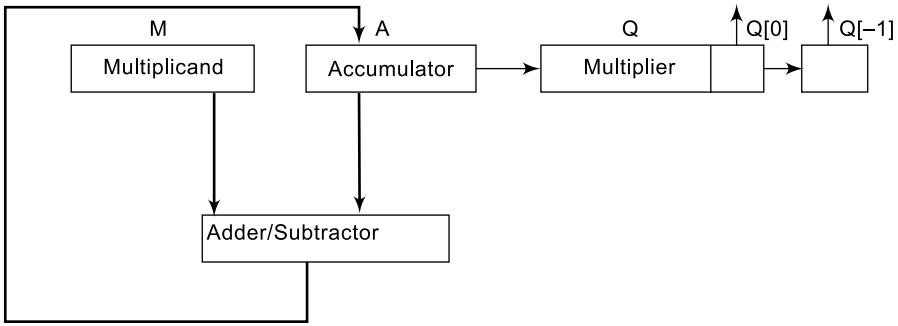


Figure 2.7 *Block diagram of Booth's multiplication algorithm*

Step-4

As Q[0]=0 and Q[–1]=1				
A=A + M	1010	1010	1100 1	—
ARS(AQ)	1010	1101	0110 0	0

Since, the Size register becomes 0, the algorithm is terminated and the product is = AQ = 1101 0110, which shows that the product is a negative number. To get the number in familiar form, take the 2's complement of the magnitude. The result is – 42.

Advantages of the Booth's multiplication method:

(i) Pre-processing steps are unnecessary, so the Booth's algorithm treats signed numbers in a uniform way with unsigned numbers.

(ii) Less number of additions and subtractions are required, compared to the sequential multiplication method.

2.8 DIVISION OF UNSIGNED INTEGERS

The division is more complex operation than multiplication. Given a dividend (D) and a divisor (V), the quotient (Q) and the remainder (R) are related according to the following expression:

$$D = QV + R, \text{ where } 0 \leq R < V.$$

Subtractive division algorithms are derived from the paper and pencil (manual) method. This process is illustrated in Fig. 2.8; using D = 42 (in decimal) = 101010 and Q = 11 (in decimal) = 1011. In this method, every iteration generates one quotient bit. First, align the divisor below the dividend from msb and try to subtract it from the dividend. If the result of the subtraction is positive, then put a 1 for the quotient, and shift the divisor one position to the right. The process is repeated. However, if the divisor cannot be subtracted from the dividend for a positive result, put a 0 for the quotient and shift the divisor to the right. Then, try to subtract the same from the dividend. This process continues until all bits of the dividend are covered.

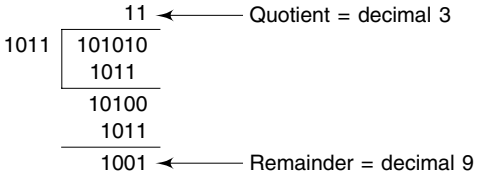


Figure 2.8 *Paper and pencil (manual) division method*

2.8.1 Restoring Division Method

Now, the manual division method can be modified in the following way to obtain restoring division method.

Instead of shifting the divisor, shift the dividend to the left. The restoring division method uses three n-bit registers A, M, Q for dividing two n-bit numbers. The register M is used to hold the divisor. Initially, A contains 0 and Q holds the n-bit dividend. In each iteration, the contents of register-pair AQ are shifted to the left first. The content of M is then subtracted from A. If the result of subtraction is positive, a 1 is placed into the vacant position created in lsb position of Q by the left shift operation; otherwise a 0 is put into this position and before beginning the next iteration, restore the content of A by adding the current content of A register with M. For this step, the algorithm is referred to as a restoring division algorithm. When the algorithm terminates, the A register contains the remainder result and the Q register contains the quotient result.

The restoring division algorithm to divide two n-bit numbers is described using the flowchart shown in Fig. 2.9. The circuit block diagram of the restoring division method is shown in Fig. 2.10.

Example 2.6 To illustrate restoring division algorithm, let us consider an example where dividend Q = 7 = 0111 and divisor M = 3 = 0011.

	M	A	Q	Size
Initial Configuration	00011	00000	0111	4
Step-1				
LS(AQ)	00011	00000	111–	—
A=A – M	00011	11101	111–	—
As Sign of A= –ve				
Set Q[0]=0				
& Restore A	00011	00000	1110	3
Step-2				
LS(AQ)	00011	00001	110–	—
A=A – M	00011	11110	110–	—
As Sign of A= –ve				
Set Q[0]=0				
Restore A	00011	00001	1100	2
Step-3				
LS(AQ)	00011	00011	100–	—
A=A – M	00011	00000	100–	—
As Sign of A= +ve				
Set Q[0]=1	00011	00000	1001	1
Step-4				
LS(AQ)	00011	00001	001–	—
A=A – M	00011	11110	001–	—
As Sign of A= –ve				
Set Q[0]=0	00011	00001	0010	0
Restore A				

From the above result, we see that the quotient = Q = 0010 = 2 and remainder = A = 00001 = 1.

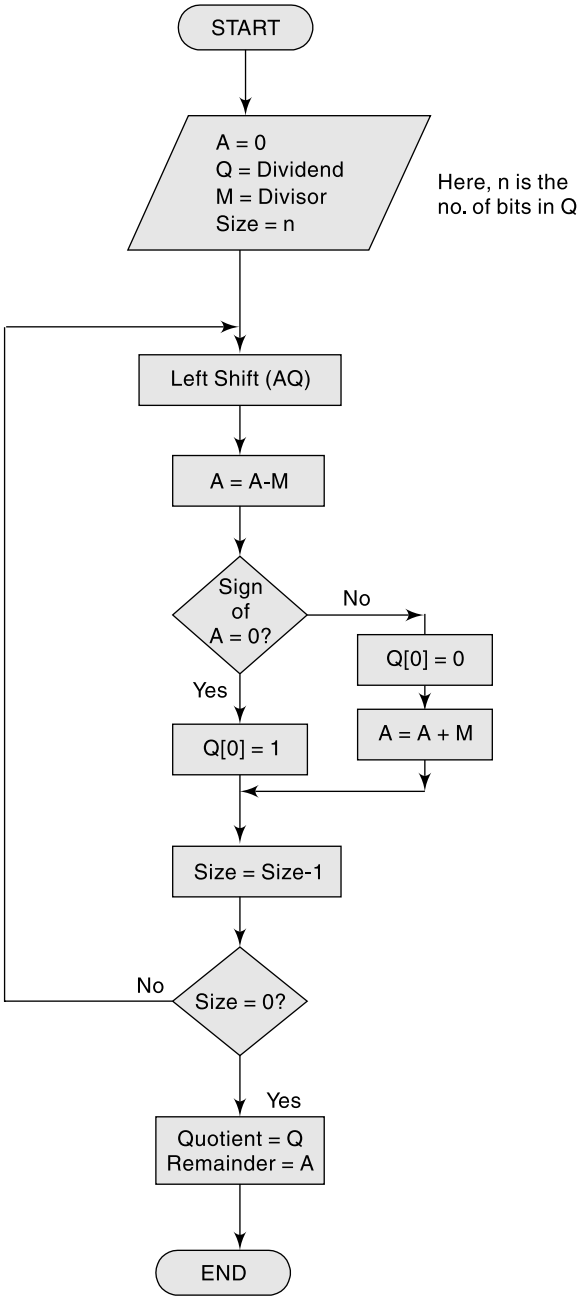


Figure 2.9 *Restoring division algorithm*

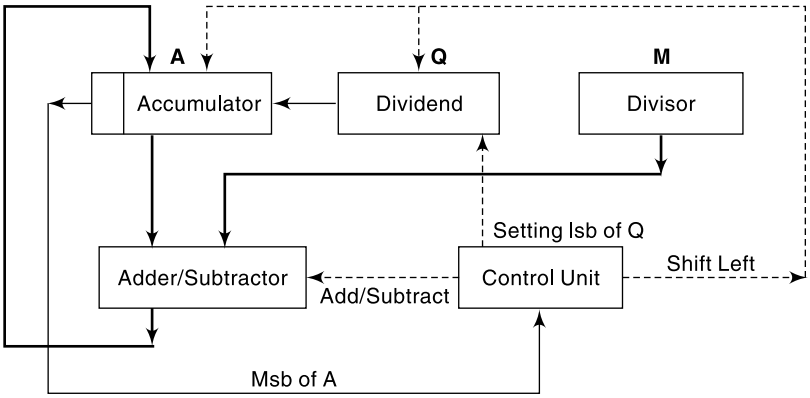


Figure 2.10 *Block diagram of restoring division algorithm*

2.8.2 Non-restoring Division Method

In the previous restoring method, we see that some extra additions are required to restore the number, when A is negative. Proper restructuring of the restoring division algorithm can eliminate that restoration step. This is known as the non-restoring division algorithm.

The three main steps in restoring division method were:

1. Shift AQ register pair to the left one position.
2. A = A – M.
3. If the sign of A is positive after the step 2, set Q[0] = 1; otherwise, set Q[0] = 0 and restore A.

Now, assume that the step 3 is performed first and then step 1 followed by step 2. Under this condition, the following two cases may arise.

Case 1: When A is positive:
Note that shifting A register to the left one position is equivalent to the computation of 2A and then subtraction. This gives the net effect on A as 2A – M.

Case 2: When A is negative:
First restore A by adding the content of M register and then shift A to the left one position. After that A will be subtracted from M register. So, all together they give rise the value of A as 2(A + M) – M = 2A + M.

Basis on these two observations, we can design the non-restoring division method and it is described in the flowchart, as shown in Fig. 2.11.

This algorithm removes the restoration step, though it may require a restoration step at the end of algorithm for remainder A, if A is negative.

Example 2.7 To illustrate this method, let us take an example where dividend Q = 0111 and divisor M = 0011.

	M	A	Q	Size
Initial Configuration	00011	00000	0111	4

Step-1				
As Sign of A= +ve				
LS(AQ)	00011	00000	111–	—
A=A – M	00011	11101	111–	—
As sign of A= –ve				
Set Q[0]=0	00011	11101	1110	3
Step-2				
As sign of A= –ve				
LS(AQ)	00011	11011	110–	—
A=A + M	00011	11110	110–	—
As sign of A= –ve				
Set Q[0]=0	00011	11110	1100	2
Step-3				
As sign of A= –ve				
LS(AQ)	00011	11101	100–	—
A=A + M	00011	00000	100–	—
As sign of A= +ve				
Set Q[0]=1	00011	00000	1001	1

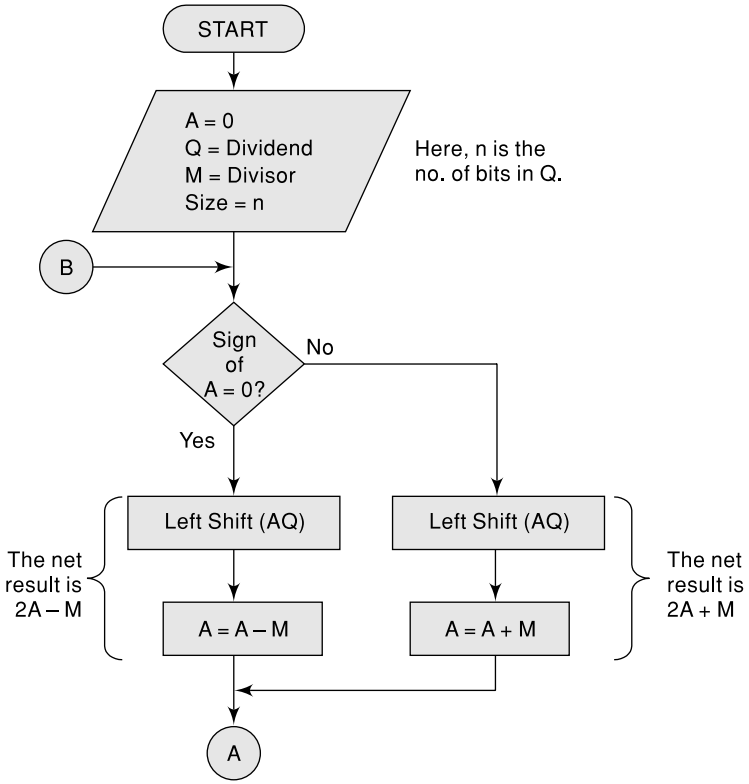


Figure 2.11 *Non-restoring division method*

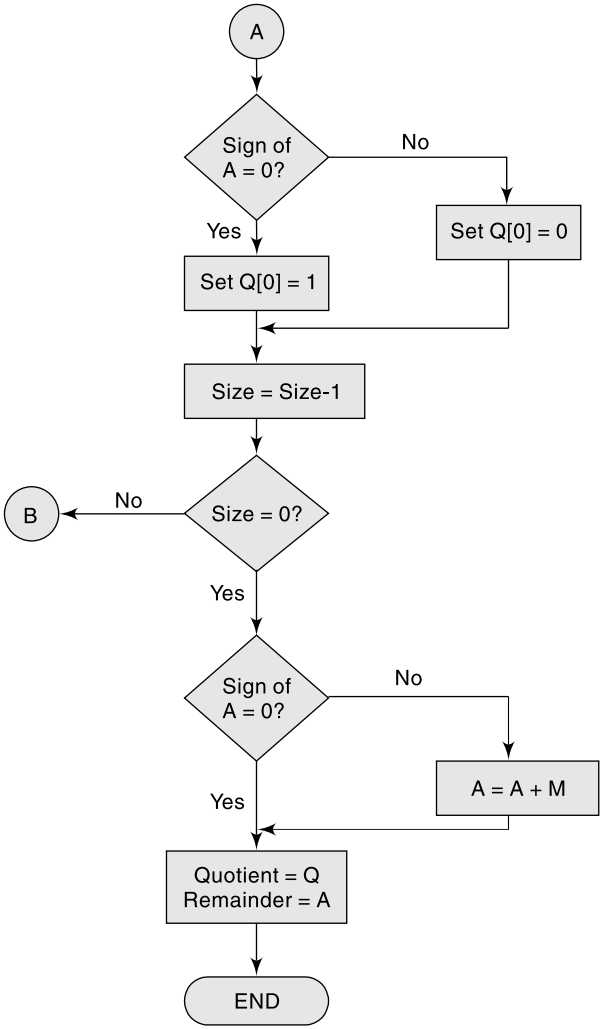


Figure 2.11 (Contd.)

Step-4

As sign of A= +ve				
LS(AQ)	00011	00001	001–	—
A=A – M	00011	11110	001–	—
As sign of A= –ve				
Set Q[0]=0	00011	00001	0010	0
Restore A				

From the above last step, we conclude that quotient = 0010 = 2 and remainder = 00001 = 1.

These two algorithms can be extended to handle signed numbers as well. The sign of the result must be treated separately as in case of multiplication and the positive magnitudes of the dividend and divisor are performed using either of the last two techniques for quotient and remainder.

2.9 ERROR DETECTION AND CORRECTION

Error detection and correction are essential parts of any communication, data storage application. Some examples are magnetic drives, CDs and mobile phones. Words (i.e. messages) that are transmitted over a communication channel can be damaged; their bits can be masked or inverted by noise. Some simple codes can detect but cannot correct these errors; others can detect and correct one or more errors. Here we address one Hamming code that can correct a single-bit error and detect a double-bit error.

2.9.1 Parity Bit and Parity Checker

Binary data, when transmitted and processed, is susceptible to noise that can alter its 1s to 0s and 0s to 1s. To detect such errors, an additional bit called the *parity bit* is added to data bits and the word containing data bits and the parity bit is transmitted. At the receiving end, the number of 1s in the word received is counted and the error, if any, is detected. However, this parity check detects only single-bit errors.

A parity bit of a 0 or a 1 is attached to the data bits such that the total number of 1s in the word is even for even parity and odd for odd parity. The parity bit can be attached to the code group either at the beginning or at the end depending on system design. A given system operates with either even or odd parity but not both. So, a word always contains either an even or odd number of 1s. At the receiving end, if the word received has an even number of 1s in the odd parity system or an odd number of 1s in the even parity system, it implies that an error has occurred.

This simple check does have limitation: it only detects errors, without being able to correct them.

2.9.2 Hamming Code Approach

Hamming codes are an extension of this simple method that can be used to detect and correct a larger set of errors. In the late 1940's, Claude Shannon was developing information theory and coding as a mathematical model for communication. At the same time, Richard Hamming, a colleague of Shannon at Bell Laboratories, found a need for error correction in his work on computers. Hamming realized that a more sophisticated pattern of parity checking allowed the correction of single errors along with the detection of double errors. The codes that Hamming devised, the single-error-correcting binary Hamming codes and their single-error-correcting, double-error-detecting extended versions marked the beginning of coding theory. These codes remain important to this day, for theoretical and practical reasons as well as historical.

A computer memory, prone to errors, could be the unreliable channel. Storage of data into memory could be the sending process and the reading of data from memory the receiving process. Protecting data in computer memories was one of the earliest applications of Hamming codes. We now describe the clever scheme invented by Hamming in 1948. To keep things simple, we describe the binary length 7 Hamming code.

2.9.3 Design of Single Error Detecting and Correcting Hamming Code

In a code where each code word contains several message bits (i.e. data bits) and several check bits

(i.e. parity bits), each check bit must be some function of the message bits. In the Hamming code each check bit is taken to be a mod 2 sum of a subset of the message bits. Assume that the rate of the code is 4/7, so that for every four information bits transmitted, there are three check bits introduced in the codeword. Call these the *parity check bits*. Three check bits c_1, c_2, c_4 are added to 4 information bits d_7, d_6, d_5 and d_3. Each of the check bits maintains even parity for specified bit positions of the 7-bit code. For such a design, the check bits c_1, c_2, c_4 are computed for 4 information bits as shown in Figure 2.12.

Decimal	Hamming Code Bit Position						
	d_7	d_6	d_5	c_4	d_3	c_2	c_1
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0

c_1 maintains even parity for c_1, d_3, d_5 and d_7.
c_2 maintains even parity for c_2, d_3, d_6 and d_7.
c_4 maintains even parity for c_4, d_5, d_6 and d_7.

Figure 2.12 *7-bit Hamming code with 4 information bits and 3 check bits c_1, c_2, c_4*

Thus as example, for four information bits 0101, $c_1 = 1$, $c_2 = 0$ and $c_4 = 1$. This 7-bit encoded word is sent to the destination where check bits c^*_1, c^*_2, c^*_4 are recomputed and the error vector e is computed as $e = (c^*_4 + c_4)\ (c^*_2 + c_2)\ (c^*_1 + c_1)$. The error correcting Hamming code is so designed that the decimal value of the error vector directly specifies the error position. Thus for e = 000, there is no error. On the other hand, if the code word 0101101 (say) is transmitted as 0001101 with an error in 6^{th} bit position, then at the destination $c^*_1 = 1$, $c^*_2 = 1$ and $c^*_4 = 0$, while $c_1 = 1$, $c_2 = 0$ and $c_4 = 1$. Hence, error vector e = 110, that is the code word has an error in the bit position 6. By inverting this bit position, the correct code word is resorted at the destination.

2.9.4 Encoding in the Hamming Code

Let us define the check bits c_1, c_2, c_4 as follows:

$c_4 \equiv d_5 + d_6 + d_7 \pmod 2$
$c_2 \equiv d_3 + d_6 + d_7 \pmod 2$
$c_1 \equiv d_3 + d_5 + d_7 \pmod 2$

After the message sequence is encoded into the code word $c = (d_7\ d_6\ d_5\ c_4\ d_3\ c_2\ c_1)$, the codeword is transmitted across the noisy channel. The channel adds an error pattern $e = (e_7\ e_6\ \ldots\ e_1)$ to the code

word, to obtain the received pattern r = c + e. The decoder then has to estimate the original message from the distorted code word.

Let us rewrite the equations above as

$d_5 + d_6 + d_7 + c_4 \equiv 0 \pmod 2$
$d_3 + d_6 + d_7 + c_2 \equiv 0 \pmod 2$
$d_3 + d_5 + d_7 + c_1 \equiv 0 \pmod 2$

Every code word satisfies these equations. Therefore, the matrix equation below

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} d_7 \\ d_6 \\ d_5 \\ c_4 \\ d_3 \\ c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

is just a restatement of the three equations above. The first matrix on the left-hand side is called the *parity check matrix* H. Thus every codeword c satisfies the equation

$$Hc^T = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Therefore, another way of describing the code is by specifying its parity check matrix H. Note that the seven columns of the parity check matrix are the seven distinct non zero combinations of three bits.

The Hamming code of length 7 is an example of a linear algebraic code. It can be proved that a binary code of length *n* is a subspace of the space of all vectors with n components, over the field $F_2 = \{0, 1\}$, i.e. each component being either a 0 or a 1. Since this Hamming code has four information bits and the check bits are completely determined by the information bits, the subspace has $2^4 = 16$ vectors. These sixteen vectors constitute the code, which is itself a subspace of the vector space containing a total of $2^7 = 128$ vectors, each of length 7 bits.

2.9.5 Syndrome Decoding

Given the received pattern r, the decoder must eventually decide what the transmitted codeword was. If the decoder is able to find the error pattern e, then the code word is c = r + e. To estimate e, it first forms the product $H r^T = H c^T + H e^T = H e^T$. This product is called the *syndrome*, and reveals the pattern of parity check failures on the received pattern.

It is desirable to design the code such that the error syndrome $H e^T$ directly specifies the error bit position. In that case, it is evident that the *i*th column of the matrix H should have the binary combination for the decimal value *i*. Because the error pattern will have a single 1 in the position in which the error has occurred and 0s everywhere else. Therefore, the error syndrome will denote the binary combination of the error position. For example, if 7-bit code word c = 0101010 is distorted by the channel and the error pattern e is 1000000, then r = c + e = 1101010. Therefore, the error syndrome specifying the error position $H (c + e)^T = (1\ 1\ 1)^T$. This proves that there is an error in the 7^{th} position.

Decoding in the Hamming code then consists of the following three steps:

1. Form the syndrome H r^T from the received vector r.
2. If the syndrome is the all zero vector, assume no errors have occurred. If it is not, then find out which column of H the syndrome matches. If the column index is i, then the estimated error pattern e is the vector with a 1 in the *i*th position and 0s everywhere else.
3. Form c = r + e as the decoder's estimate of the transmitted codeword.

2.9.6 Single Error Correcting Code for 8-bit Information

For 8-bit information, the number of check bits required is 4. The check bits c_1 , c_2 , c_4 and c_8 are defined as follows:

$c_8 \equiv d_9 + d_{10} + d_{11} + d_{12} \pmod 2$
$c_4 \equiv d_5 + d_6 + d_7 + d_{12} \pmod 2$
$c_2 \equiv d_3 + d_6 + d_7 + d_{10} + d_{11} \pmod 2$
$c_1 \equiv d_3 + d_5 + d_7 + d_9 + d_{11} \pmod 2$

The parity check matrix H for this 12-bit single error correcting code is thus given below.

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The error syndrome will denote the binary combination of the error position. For example, if 12-bit code word c = 101110111110 is distorted by the channel and the error pattern e is 010000000000, then r = c + e = 111110111110. Therefore, the error syndrome specifying the error position H $(c + e)^T$ = $(1\ 0\ 1\ 1)^T$, implying the error in 11^{th} position.

2.9.7 Single Error Correcting and Double Error Detecting Code

Under the assumption that the probability of the channel flipping a bit during transmission is less than 1/2, and that bit errors occur independently of one another, the more probable error pattern is the one with fewer 1s. Thus, the decoder follows what is known as a *maximum likelihood* strategy and decodes into the codeword that is at the smallest Hamming distance from the received pattern. *Hamming distance* between code words is the number of corresponding positions in which the code words differ. In other words, even if there is a double error, the decoder will mistake it for a single error, as there will be a single error with an identical syndrome, which the maximum likelihood strategy will choose as its estimate of the error pattern. Thus, if this Hamming code is used for single error correction, it cannot correctly *detect* double errors.

For double error detection along with single error correction, the minimum Hamming distance should be 4. Such a code can be designed by adding an extra check bit to the single-bit error correcting code. For the 7-bit single error correcting code, the extra check bit c_8 can be added to maintain parity for each of the eight bits. The check bits c_1, c_2, c_4 and c_8 are defined as follows:

$c_8 \equiv d_7 + d_6 + d_5 + c_4 + d_3 + c_2 + c_1 \pmod 2$
$c_4 \equiv d_5 + d_6 + d_7 \pmod 2$
$c_2 \equiv d_3 + d_6 + d_7 \pmod 2$
$c_1 \equiv d_3 + d_5 + d_7 \pmod 2$

The corresponding parity check matrix H is thus given below.

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

For such a design, if most significant bit of error syndrome is 1, then the remaining 3 positions specify the single bit error position. On the other hand, if most significant bit of the non-zero error syndrome is 0, then it indicates a double bit error situation. Since error in any single bit position will be invariably reflected in the check bit c_8. So in the syndrome, if the check bit c_8 is 0 with non-zero state for other check bits, it implies a double error case. Consider following two examples:

Example 1: Codeword c = 00101101 and error pattern e = 01000000.
So, c + e = 01101101
Therefore, $H (c + e)^T = (1\ 1\ 1\ 1)^T$ which indicates an error in 7^{th} bit position.

Example 2: Codeword c = 00101101 and error pattern e = 00011000.
So, c + e = 00110101
Therefore, $H (c + e)^T = (0\ 0\ 0\ 1)^T$ which indicates double bit error situation.

SOLVED PROBLEMS

1. *Directly convert the following decimal numbers into hexadecimal numbers:*
   (a) *70*
   (b) *130*
   (c) *1348*

Answer
(a) 70 = (4 × 16) + 6 = 46 in HEX.
(b) 130 = (8 × 16) + 2 = 82 in HEX
(c) 1348 = (5 × 16 × 16) + (4 × 16) + 4 = 544 in HEX.

2. *Directly convert the following hexadecimal numbers into decimal numbers:*
   (a) *7A*
   (b) *1F*
   (c) *13C*

Answer
(a) 7A = (7 × 16) + 10 = 122
(b) 1F = (1 × 16) + 15 = 31
(c) 13C = (1 × 16 × 16) + (3 × 16) + 12 = 316

3. *What is the radix of number if the solution of the quadratic equation:*
$x^2 - 10x + 31 = 0$ *is* $x = 5$ *and* $x = 8$.

Answer

The solution of the quadratic equation:

$$x^2 - 10x + 31 = 0 \text{ is } x = 5 \text{ and } x = 8.$$

Let r be the radix of the number.

This means,

$$(x^2 - 10x + 31)_r = [(x - 5)(x - 8)]_{10}$$
$$= [x^2 - (5 + 8)_{10}\, x + (40)_{10}] = [x^2 - (13)_{10}\, x + (40)_{10}]$$

Therefore, $(10)_r = (13)_{10}$

$\Rightarrow \quad 1 \times r = 13$

$\Rightarrow \quad r = 13$

Also, $(31)_r = (40)_{10}$. This holds true for $r = 13$.

So, the radix of the number is 13.

4. *Represent integer number –19 in 8-bit format using*
   (a) *signed magnitude method*
   (b) *signed 1's complement method*
   (c) *signed 2's complement method*

Answer

In signed magnitude method, the representation is: 1001 0011

In signed 1's complement method, the representation is: 1110 1100

In signed 2's complement method, the representation is: 1110 1101

5. *What are the minimum and maximum integers representable in n-bit value using*
   (a) *signed magnitude method?*
   (b) *signed 1's complement method?*
   (c) *signed 2's complement method?*

Give the argument for each.

Answer

(a) In signed magnitude method, one bit is used to record the sign of the number, giving the representable range in n-bit is: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$.

(b) Like signed magnitude method, signed 1's complement method reserves one bit for the sign of the number, giving the representable range in n-bit is: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$.

(c) In signed-2's complement representation, only one representation for 0 is used, allowing an odd number of non-zero values to be represented. Thus the range for numbers using n bits is: $-(2^{n-1})$ to $+(2^{n-1} - 1)$.

6. *Use 8-bit two's complement integers, perform the following computations:*
   (a) *–34 + (–12)* (b) *17 – 35*
   (c) *–22 – 7* (d) *18 – (–5)*

Answer

(a) In 2's complement representation, –34 = 1101 1110

–12 = 1111 0100

Adding these two numbers, we get, 11101 0010, which is 9-bit result. By addition rule discard the 9th bit and get the result: 1101 0010, which shows that the number is negative. To get the result in its familiar form, take 2's complement of the result. The result is –46.

(b) 17 – 35: This is subtraction and we know that 2's complement of (35) is to be added with 17. The representation of 17 is 0001 0001 and 2's complement of 35 is 1101 1101. After addition, we get 1110 1110. This is negative number and its value is –18.

(c) –22 – 7 = (–22) – 7. This is a subtraction. So, 2's complement of 7 is to be added with (–22). The 2's complement of 7 = 1111 1001 and representation of (–22) = 1110 1010. After addition of these two we get, 11110 0011. This is 9-bit result. So, by rule, discard 9th bit and get the result as 1110 0011, which shows it is negative number. The result in equivalent decimal is –29.

(d) 18 – (–5) = 18 + 5. The representation of 18 is 0001 0010 and representation of 5 is 0000 0101. We get after addition, 0001 0111. The result is equivalent to decimal 23.

7. *Can you add 8-bit signed numbers 0110 0010 and 0100 0101? If not, why? Suggest a solution.*

Answer

Carries: 01

```
        0110 0010
        0100 0101
(add)   1010 0111
```

This result suggests that the number is negative, which is wrong. However, if the carry out from the sign bit position is treated as the sign of the result, the 9-bit answer thus obtained will be correct answer. So, there is an overflow.

To detect an overflow condition the carry into the sign bit position and the carry out from the sign bit position are examined. If these two carries are both 0s or both are 1s, there is no overflow. If these two carries are not equal (i.e., if one is 0 and other is 1), an overflow condition exists. Considering the carry from the sign bit position with (i.e. 0, here) along with the 8-bit result will give correct answer.

8. *Write down the Boolean expression for overflow condition when adding or subtracting two binary numbers expressed in two's complement.*

Answer

If one number is positive and the other is negative, after an addition overflow cannot occur, since addition of a positive number to a negative number produces a number that is always smaller than the larger of the two original numbers. However, an overflow may occur if the two numbers added are of same sign i.e., both are positive or both are negative. Let's consider following examples.

Carries: 01		Carries: 10	
+69	0 1000101	–69	1 0111011
+78	0 1001110	–78	1 0110010
+147	1 0010011	–147	0 1101101

Observe that the 8-bit result that should have been positive (first example) has a negative sign bit and the 8-bit result that should have been negative (second example) has a positive sign bit. However, if the carry out from the sign bit position is treated as the sign of the result, the 9-bit answer thus obtained will be correct answer. Since the 9-bit answer cannot be accommodated with 8-bit register, we say that an overflow results.

To detect an overflow condition the carry into the sign bit position (i.e. C_{n-1}) and the carry out from the sign bit position (i.e. C_n) are examined. If these two carries are both 0s or both are 1s, there is no overflow. If these two carries are different, an overflow condition exists. The overflow occurs if the Boolean expression $C_n \oplus C_{n-1}$ is true.

9. *Give the merits and demerits of the floating point and fixed point representations for storing real numbers*

Answer

Merits of fixed-point representation:

(a) This method of representation is suitable for representing integers in registers.
(b) Very easy to represent, because it uses only one field: magnitude field.

Demerits of fixed-point representation:

(a) Range of representable numbers is restricted.
(b) It is very difficult to represent complex fractional numbers.
(c) Since there is no standard representation method for it, it is some time confusing to represent a number in this method.

Merits of floating-point representation:

(a) By this method, any type and any size of numbers can be represented easily.
(b) There are several standardized representation methods for this.

Demerits of floating-point representation:

(a) Relatively complex representation, because it uses basically two fields: mantissa and exponent fields.
(b) Length of register for storing floating-point numbers is large.

10. *Add 2.56 and 2.34 × 10^2, assuming three significant decimal digits. Round the sum to the nearest decimal number with three significant decimal digits.*

Answer

First we must shift the smaller number to the right to align the exponents, so 2.56 becomes 0.0256 × 10^2. The sum of mantissas is

$$\begin{array}{r} 2.3400 \\ +\ 0.0256 \\ \hline 2.3656 \end{array}$$

Thus, the sum is 2.3656×10^2. Since, we have two digits to round, we want values 0 to 49 to round down and 51 to 99 to round up, 50 being the tiebreaker. Rounding the sum up with three significant digits gives 2.37×10^2.

11. *Represent following decimal numbers in IEEE 754 floating point format:*
    (a) *–1.75*
    (b) *21*

Answer

(a) The decimal number – 1.75 = – 1.11 in binary = -1.11×2^0
The 23-bit mantissa M = 0.110000 000000 000000 00000
The biased exponent E′ = E + 127 = 0 + 127 = 127 = 0111 1111

Since the number is negative, the sign bit S = 1
Therefore, the IEEE single-precision (32-bit) representation is:

1	0111 1111	110000 000000 000000 00000

(b) The decimal number 21 = + 10101 in binary = + 1.0101×2^4
The 23-bit mantissa M = 0.010100 000000 000000 00000
The biased exponent E′ = E + 127 = 4 + 127 = 131 = 1000 0011
Since the number is positive, the sign bit S = 0
Therefore, the IEEE single-precision (32-bit) representation is:

0	1000 0011	010100 000000 000000 00000

12. *What value is represented by the IEEE single precision floating point number: 0101 0101 0110 0000 0000 0000 0000 0000?*

Answer
The sign of the number = 0, biased exponent value = 10101010 = 170. So the exponent value = 170 – 127 = 43. The mantissa field = 110 0000 0000 0000 0000 0000.
Therefore, the value of the number = + $(1.11)_2 \times 2^{43} = 1.75 \times 2^{43} = 1.539 \times 10^{13}$ (approx.).

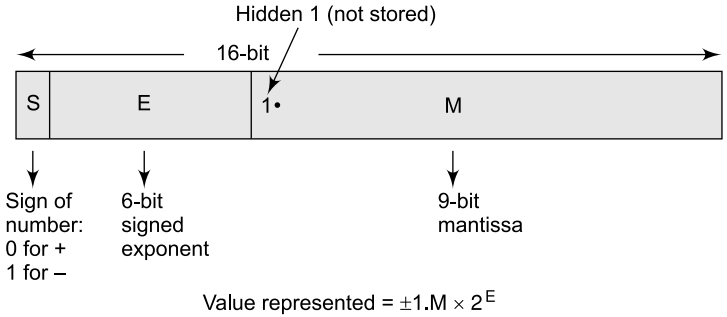
13. *How NaN (Not a Number) and Infinity are represented in IEEE 754 standard?*

Answer
Not a Number (NaN) is represented when biased exponent E′ = 255 and mantissa M ≠ 0. NaN is a result of performing an invalid operation such as 0/0 and $\sqrt{-1}$.

Infinity is represented when E′ = 255 and M = 0. The infinity is the result of dividing a normal number by 0.

14. *A floating point number system uses 16 bits for representing a number. The most significant bit is the sign bit. The least significant nine bits represent the mantissa and remaining 6 bits represent the exponent. Assume that the numbers are stored in the normalized format with one hidden bit*
    (a) *Give the representation of -1.6×10^3 in this number system.*
    (b) *What is the value represented by 0 000100 110000000?*

Answer
The format of the 16-bit floating-point representation is as follows:



(a) The representation of -1.6×10^3 in this system:
The decimal number $-1.6 \times 10^3 = -1600 = -11001000000$ in binary $= -1.1001000000 \times 2^{10}$.
Mantissa (M) in 9-bit = 0.6 = 0.100100000
Exponent (E) in 6-bit = 10 = 001010
Since the number is negative, the sign bit S = 1
Therefore, the 16-bit representation is:

1	001010	100100000

(b) The binary number is 0 000100 110000000
The msb indicates the number is positive.
The biased exponent (E) = 000100 = 4
The mantissa (M) = 110000000 = 384.
Thus the value represented by this number = $+1.\ M \times 2^E = +1.384 \times 2^4 = 22.144$.

15. *Compute the product of the following pair of unsigned integers. Generate the full 8-bit result.*
(a) *1001 × 0110*
(b) *1111 × 1111*

Answer

(a) 1001 × 0110.
This can be written as
(1001 × 100) + (1001 × 10) = 100100 + 10010 = 0011 0110.

(b) 1111 × 1111.
This can be written as
(1111 × 1000) + (1111 × 100) + (1111 × 10) + (1111 × 1) = 1111000 + 111100 + 11110 + 1111 = 1110 0001.

16. *Multiply –13 and +14 by the method of partial products.*

Answer

–13 = 10011 (multiplicand)
+14 = 01110 (multiplier)

```
    00000
   10011
  10011
 10011
100001010
```

In case of negative multiplicand, 2's complement multiplier is added additionally with n-bit shift due to sign extension of partial product on the right shifting.

The 2's complement of multiplier left shifted by n i.e. 5 bits is 1001000000.

Add this shifted number with added result of partial products, to get the correct result of signed numbers multiplication.

Thus, 100001010 + 1001000000 = 1101001010 = –182.

17. *Give the recoded Booth's multiplier representations for the following:*
    (a) *1100 1010*
    (b) *1110 1101*

Answer

(a) Original multiplier: 1 1 0 0 1 0 1 0 = 0 1 1 0 0 1 0 1 0
Recoded pattern: 1 0 $\bar{1}$ 0 1 $\bar{1}$ 1 $\bar{1}$ 0

(b) Original multiplier: 1 1 1 0 1 1 0 1 = 0 1 1 1 0 1 1 0 1
Recoded pattern: 1 0 0 $\bar{1}$ 1 0 $\bar{1}$ 1 $\bar{1}$

18. *For Booth's algorithm, when do worst case and best case occur?*

Answer

Worst case is one when there are maximum number of pairs of (01)s or (10)s in the multipliers. Thus, maximum number of additions and subtractions are encountered in the worst case.

Best case is one when there is a large block of consecutive 1s in the multipliers, requiring minimum number of additions and subtractions.

19. *Multiply +12 and +14 using Booth's recoding technique.*

Answer

Since 5-bit number can be in the range –16 to +15 only and the product 12 × 14 will be outside this range, we use 10-bit numbers.

Multiplicand (+12) = 00000 01100
Multiplier (+14) = 00000 01110
After Booth's recoding, multiplier = 00000 100 $\bar{1}$ 0
1st partial product = 0000000000
2nd partial product= 111110100 (2's complement of multiplicand)
3rd partial product= 00000000
4th partial product= 0000000
5th partial product= 001100
(6th –10th) partial products are all 0s

After addition, result = 0010101000 = 168.

20. *Describe Booth's modified algorithm and show that just N/2 partial products are required to multiply two N-bit binary numbers. Describe the method using the two numbers A* = 10101010 *and B* = 11001110.

Answer

A faster version of Booth's multiplication algorithm for signed numbers, known as the *modified Booth's algorithm*, examines three adjacent bits $Q[i + 1]$ $Q[i]$ $Q[i - 1]$ of the multiplier Q at a time, instead of two. Apart from three basic actions performed by original Booth's algorithm, which can be expressed as: add 0, 1 × M (multiplicand) and $\bar{1}$ × M to Ac (the accumulated partial products), this modified algorithm performs two more actions: add 2 × M and $\bar{2}$ × M to Ac. These have the effect of increasing the radix from 2 to 4 and allow an N × N multiplication requiring only N/2 partial products.

Observe that bit pair (1, $\bar{1}$) is equivalent to pair (0, 1). That is instead of adding $\bar{1}$ times of the multiplicand M at shift position i to 1 × M at position i + 1, the same result is obtained by adding

$1 \times M$ at position i. Other cases are: (1, 0) is equivalent to (0, 2), ($\bar{1}$, 1) is equivalent to (0, $\bar{1}$), and so on. The following table shows the multiplicand selection decisions for all possibilities.

Q[i + 1]	*Q[i]*	*Q[i – 1]*	*Multiplicand selected at position i*
0	0	0	$0 \times M$
0	0	1	$1 \times M$
0	1	0	$1 \times M$
0	1	1	$2 \times M$
1	0	0	$\bar{2} \times M$
1	0	1	$\bar{1} \times M$
1	1	0	$\bar{1} \times M$
1	1	1	$0 \times M$

Operands	Values	i	Q[i + 1]	Q[i]	Q[i – 1]	Action
Multiplicand M = A	1010 1010					
Multiplier Q = B	1100 1110					
P0	0000 0000 1010 1100	0		100		Add $\bar{2} \times M$ to Ac
P2	0000 0000 0000 00	2		111		Add $0 \times M$ to Ac
P4	1111 1010 1010	4		001		Add $1 \times M$ to Ac
P6	0001 0101 10	6		110		Add $\bar{1} \times M$ to Ac
Product	0001 0000 1100 1100 = P0 + P2 + P4 + P6.					

21. *How can the non-restoring division algorithm be deduced from restoring division algorithm?*

Answer

The three main steps in restoring division method are:

1. Shift AQ register pair to the left one position.
2. A = A – M.
3. If the sign of A is positive after the step 2, set Q[0] = 1; otherwise, set Q[0] = 0 and restore A.

Now, assume that the step 3 is performed first and then step 1 followed by step 2. Under this condition, the following two cases may arise.

Case 1: When A is positive:

Note that shifting A register to the left one position is equivalent to the computation of 2A and then subtraction. This gives the net effect on A as 2A – M.

Case 2: When A is negative:

First restore A by adding the content of M register and then shift A to the left one position. After that A will be subtracted from M register. So, all together they give rise to the value of A as 2(A + M) – M = 2A + M.

Basis on these two observations, we can design the non-restoring division method.

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:
   (i) A binary number with n digits has the value
   (a) $n^2 - 1$ (b) 2^4 (c) $2^{(n-1)}$ (d) $2^n - 1$
   (ii) The ASCII code is for information interchange by a binary code. It is for
   (a) numbers only
   (b) alphabet only
   (c) alphanumeric and other common symbols
   (d) none of these.
   (iii) The Excess-3 code for number 3 is
   (a) 0110 (b) 0011 (c) 0001 (d) 1001
   (iv) The maximum unsigned binary number in 8-bit is
   (a) 255 (b) 256 (c) 128 (d) 127.
   (v) The minimum and maximum 8-bit numbers in signed magnitude representation are
   (a) 0 and 255 (b) – 127 and 127 (c) – 128 and 127 (d) none.
   (vi) The minimum and maximum 8-bit numbers in signed 2's complement representation are
   (a) –127 and 127 (b) –128 and 127 (c) 0 and 255 (d) none.
   (vii) The complement operations are useful for
   (a) logical operations
   (b) addition and subtraction operations
   (c) arithmetic operations
   (d) subtraction and logical operations.
   (viii) The number – 0.125 is represented in IEEE single-precision format as:

(a)	1	0111 1100	1000…… 0
(b)	1	0111 1110	1000…… 0
(c)	1	0111 1111	0010…… 0
(d)	1	0111 1100	0000…… 0

   (ix) In floating-point representation, biased exponent is used to:
   (a) facilitate representation of zero
   (b) increase the range of representation
   (c) reduce the overhead of comparing the sign bits of exponent in floating point arithmetic
   (d) both (a) and (c).
   (x) The floating-point numbers are normalized
   (a) to enhance to range of representation
   (b) to increase the precision of the number
   (c) to make the number simple
   (d) both (a) and (b).
   (xi) The hidden one-bit to the immediate left of mantissa of IEEE 754 floating-point representation is used to facilitate:
   (a) enhancement of the range of representation

(b) representation of NaN (not a number) representation
(c) enhancement of the precision of number
(d) representation of sign bit.

(xii) The overflow in floating-point representation is detected by inspecting
(a) the carry into sign bit and carry-out from the sign bit positions and if they are different
(b) these two carries and if they are same
(c) the size of the numbers
(d) the sign of the number.

(xiii) Floating-point representation is used to store
(a) Boolean values (b) whole numbers
(c) real numbers (d) integers

(xiv) Guard bits are
(a) least significant some bits used to increase the precision of number
(b) least significant some bits used to guard against virus attacks on the stored data
(c) most significant some bits used to hold bits which would be shifted out after left shift operations
(d) used to perform logical shift operations efficiently.

(xv) Which of the following the truncation technique has the smallest unbiased rounding error?
(a) chopping (b) Von Neumann rounding
(c) rounding (d) both (b) and (c).

(xvi) The number of AND gates and number of full adders required in 4×4 array multiplier are
(a) 4 and 4 (b) 16 and 4
(c) 4 and 16 (d) 16 and 16, respectively.

(xvii) The sequential multiplication method is generally used to multiply two unsigned numbers, but can be used for multiplication of signed numbers, where the sign of the product is processed separately using one
(a) OR gate (b) NOT gate (c) XOR gate (d) AND gate.

(xviii) The maximum number of additions and subtractions are required for which of the following multiplier numbers in Booth's algorithm?
(a) 0100 1111 (b) 0111 1000 (c) 0000 1111 (d) 0101 0101.

(xix) Which multiplier out of the following gives the worst case for implementing Booth's algorithm?
(a) 0111 0000 (b) 0111 0110 (c) 0000 0111 (d) 0101 0101

(xx) Which multiplier out of the following gives the best case for implementing Booth's algorithm?
(a) 0111 1100 (b) 0111 0110 (c) 0000 0111 (d) 0101 0101

(xxi) The number 237.5 in IEEE format will be truncated to
(a) 237 (b) 238 (c) 1.237.5 (d) 0.2735

(xxii) Only one restoration step may be required in non-restoring division algorithm, if
(a) the sign of accumulator register is negative
(b) the sign of accumulator register is positive
(c) the accumulator register produces overflow
(d) the accumulator register produces underflow.

(xxiii) To transmit data bits 1011, the correct even parity 7-bit Hamming Code is
(a) 0101101 (b) 1010101 (c) 1100111 (d) 0110111

Group B

2. Why do digital computers use binary numbers for their operations?
3. Describe two complement methods. Prove that complement of a complement is the original number.
4. Discuss the fixed-point representation method with example.
5. What are representations of integer number – 19 in 8-bit format using
   (a) signed magnitude method?
   (b) signed 1's complement method?
   (c) signed 2's complement method?
6. Compare different integer representation methods.
7. What are the minimum and maximum integers representable in n-bit value using
   (a) signed magnitude method?
   (b) signed 1's complement method?
   (c) signed 2's complement method?
   Give the argument for each.
8. Discuss the overflow problem in fixed-point representation and its detection method using example.
9. Describe floating-point representation using examples.
10. When is a floating-point called normalized floating-point number? Give the reason(s) for converting a non-normalized floating-point number into normalized one.
11. What is biased exponent and why?
12. Give the IEEE single-precision representation for floating-point numbers. Represent the number – 7.75 in this representation.
13. Discuss the overflow and underflow problems in floating-point representation.
14. What is a guard bit? Describe different truncation methods with their errors.
15. Discuss the 4×4 array multiplier method with block diagram.
16. Discuss the sequential multiplication method and use this to multiply decimal numbers 27 and 56. Can you apply this method to multiply two signed numbers?
17. Give the recoded Booth's multiplier representations for the following:
   (a) 1001 0011
   (b) 1110 1010
18. Describe the Booth's multiplication method and use this to multiply decimal numbers –23 and 9. What are the advantages of this method?
19. Discuss the restoring division algorithm and use this to divide decimal number 23 by 6. Can you apply this method to divide two signed numbers?
20. Deduce the non-restoring division algorithm from restoring division algorithm.
21. Describe the non-restoring division algorithm and use this to divide decimal number 29 by 7.
22. What is a parity bit? How is the syndrome for the Hamming code interpreted?
23. For the 8-bit word 0011 1001, the check bits stored with it would be 0111. Suppose when the word is read from memory, the check bits are calculated to be 1101. What is the data word that was read from memory?

CHAPTER

3
Datapath and Design of Arithmetic Logic Unit

3.1 INTRODUCTION

As discussed in Section 1.6, the Central Processing Unit (CPU) consists of two major parts: a datapath (data processing) unit and a Control Unit (CU). The *datapath* is a collection of the Arithmetic Logic Unit (ALU) and various registers capable of performing certain (micro)operations on the data. The ALU simply executes the instructions in the order as dictated by the CU. The ALU performs the instruction execution on the operand data stored in registers.

In this chapter, we will discuss various registers that are used in any processor, register transfer micro-operation. Also, we will design one hypothetical ALU and prior to that, some important arithmetic units will be discussed.

3.2 MICRO-OPERATION

A CPU with many registers reduces the number of references to the main memory, and thus simplifying the programming task and shortening the execution time. As instruction execution progresses, data are transferred from one register to another, often passing through the ALU to perform some arithmetic or logic operation.

An operation performed on the data stored in registers is called micro-operation. The result of the micro-operation may replace the previous binary information of a register or may be transferred to another register. Examples of micro-operations are add, subtract, shift, load and clear, etc.

The internal hardware structure of a computer is characterized by the following attributes:

1. The types of micro-operations performed on the binary information stored in registers.
2. The control signals that initiate the sequence of micro-operations.
3. The set of registers it contains and their functions.

The frequently used micro-operations in digital computers are classified into four categories:

1. Register Transfer Micro-operations: Transfer of binary information from one register to another.

2. Arithmetic Micro-operations: Arithmetic operations performed on the data stored in registers.
3. Logical Micro-operations: Bit manipulation operations on non-numeric data stored in registers.
4. Shift Micro-operations: Shift operations on data stored in registers.

3.3 CPU REGISTERS

Computers contain some registers within CPU for faster execution. The number of registers differs from processor to processor. A register is nothing but a collection of flip flops (see Appendix, for details) each capable of storing one bit of information. Registers are available in the following forms:

(a) Accumulator (AC).
(b) General-purpose registers,
(c) Special-purpose registers.

Computer registers are designated by capital letters (sometimes followed by numerical) to denote the function of the register.

Accumulator (AC) The accumulator is a register which holds one of the operands before the execution of an instruction, and receives the result of most of the arithmetic and logical micro-operations. Thus, an accumulator is the most frequently used register. Some CPUs have a single accumulator and some have several accumulators. An accumulator is denoted by AC or sometimes A.

General-purpose Registers General-purpose registers or processor registers are used for storing data and intermediate results during the execution of a program. These registers are donated by capital letter R followed by some number. The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1, starting from 0 in the rightmost position and increasing the numbers towards the left. The Fig. 3.1 shows the representation of 16-bit register in block diagram.

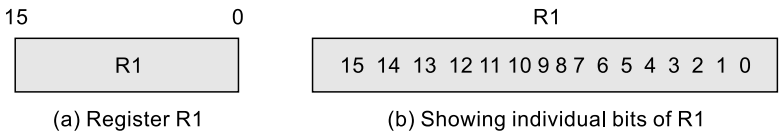


Figure 3.1 *Block diagram of a register*

Special-purpose Registers Each processor contains a number of special purpose registers for various purposes. Commonly used special-purpose registers and their functions are summarized below:

Register	*Function*
PC (Program Counter)	Holds the address of the next instruction to be executed.
IR (Instruction Register)	Holds the instruction code (operation code) currently being executed.
SP (Stack Pointer)	Holds the address of the top element of the memory stack.
BR (Base Register)	Holds the starting address of the memory stack.
MAR (Memory Address Register)	Holds the address of the data item to be retrieved from the main memory.
MBR or DR (Memory Buffer Register or Data Register)	Holds the data item retrieved from the main memory.
SR or PSW (Status Register or Program Status Word)	Holds the condition code flags and other information that describe the status of the currently executing program.

3.4 REGISTER TRANSFER LANGUAGE (RTL)

Register Transfer Language (RTL) is the symbolic notation used to describe the micro-operation transfer between registers. Information transfer from one register to another is characterized in symbolic form by means of a replacement operator. The symbolic code R1 ← R2 indicates a transfer of the content of register R2 into R1. The transfer micro-operation means the content of source register R2 is copied into the destination register R1, but the content of R2 remains same. This transfer micro-operation overwrites the content of R1 by the content of R2.

As far as internal hardware connectivity is concerned, a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability. Normally, the register transfer occurs under a predetermined control condition. This can be illustrated by means of an if-then symbolic code:

If (C = 1) then (R1 ← R2)

where C is a control signal generated in the control circuit. A control function is sometimes specified to separate the control variables from the register transfer operation. A control function is nothing, but a Boolean variable that is equal to 1 or 0. Thus the above symbolic code is equivalent to:

C: R1 ← R2

A colon is used to terminate the control condition. This code means that the transfer micro-operation be performed only if C = 1. Fig. 3.2(a) shows the block diagram that illustrates the transfer from R2 to R1. Register R1 has a load control input C that is controlled by the control circuit. A common clock is used to synchronize all the activities in the total circuit for transfer operation.

As shown in the timing diagram (Fig. 3.2(b)), in the rising edge of a clock pulse at time t, the control section activates C. The next positive transition of the clock at time t + 1 finds the load input enabled and the data inputs of R1 are then loaded with the data outputs of register R2 in parallel. At time t + 1, C should be disabled; otherwise if C remains active, the transfer will occur with every clock pulse transition.

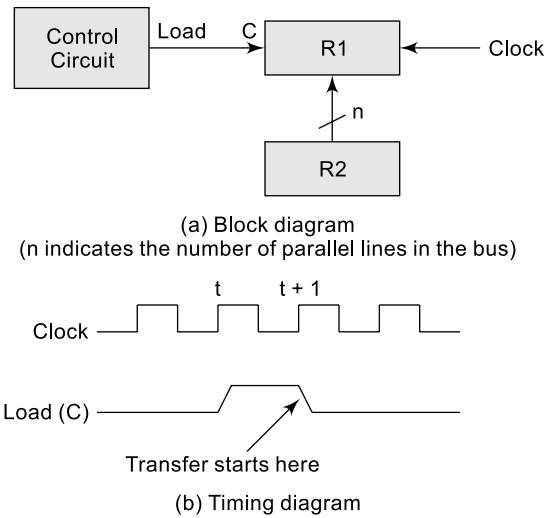


Figure 3.2 *Transfer from R2 to R1 when C = 1 (in symbolic code C: R1 ← R2)*

3.5 BUS TRANSFER

Many registers are provided in the CPU of a computer for fast execution. Therefore several paths must be provided to transfer information from one register to another. If a separate communication line is used between each register pair in the system, the number of lines will be excessive and thus cost of communication will be huge. Thus it is economical to have a common bus system for

transferring information between registers in a multiple-register configuration. A bus system consists of a group of common communication lines, where each line is used to transfer one bit of a register at a time. Thus, a shared communication path consisting of one or more connection lines is known as a *bus* and the transfer of data through this bus is known as *bus transfer*. Sometimes, it is said that n-bit bus or n-line bus, the meaning of which is that the bus consists of n parallel lines to transfer n-bit of data all at a time. The n is called width of the bus. The width of the bus has an impact on a computer's performance. The wider the bus, the greater the number of bits transferred at a time.

We will present two ways to construct common bus system. One way is using multiplexers (simply known as MUXs, see appendix for MUX) and another way is using tri-state buffers.

3.5.1 Construction of a Common Bus Using MUXs

The Fig. 3.3 shows an n-line common bus system using multiplexers for register transfer, where four registers are used each of n-bit. This common bus is used to transfer a register's content to other register or memory at a single time. A multiplexer selects one source register whose all n-bit information is then placed on the bus. Two multiplexers are shown in the figure one for the low-order significant bit and another for the high-order significant bit. The bus consists of n 4×1 multiplexers each having four data inputs, 0 through 3 and two common selection lines for all multiplexers.

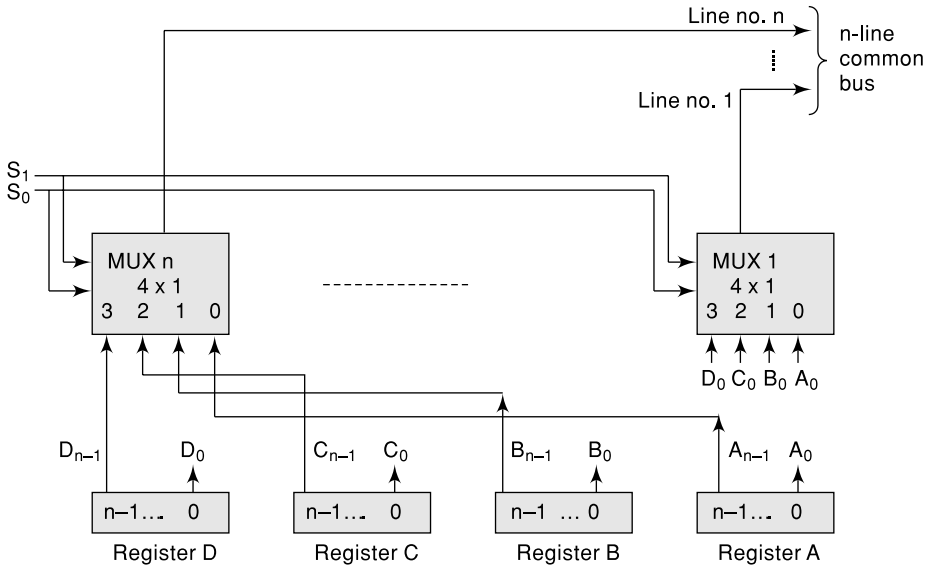


Figure 3.3 *Bus system for four registers*

Each MUX has four input lines each is connected to all four registers' bits marked. The two selection lines S_0 and S_1 are connected to the selection inputs of all n MUXs. The selection lines choose all n bits of one register and transfer them into the n-line common bus. For example, when S_1 $S_0 = 00$, the 0^{th} data inputs of all n MUXs are selected and placed to the outputs that form the bus. The function table for bus shown in Fig. 3.3 is given below.

S_1	S_0	*Register selected*
0	0	A
0	1	B
1	0	C
1	1	D

General case Suppose an n-line bus system is to be constructed for k registers of n bits each. The number of MUXs needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k × 1, since it multiplexes k data lines.

3.5.2 Construction of a Common Bus Using Tri-state Buffers

Another way to construct a common bus system is using tri-state buffers. A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that no output is produced though there is an input signal and does not have logic significance. The gate is controlled by one separate control input C. If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low the gate does not produce any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown in Fig. 3.4.

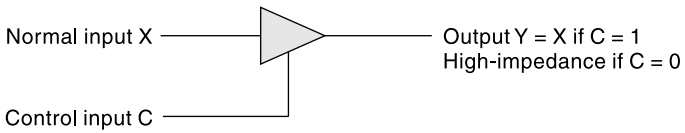


Figure 3.4 *Graphic symbol for a tri-state buffer gate*

A common bus system with tri-state buffers is described in Fig. 3.5. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

General case Suppose an n-line common bus for k registers of n bits each using tri-state buffers needs to be constructed. We need n circuits with k buffers in each as shown in Fig. 3.5. Therefore, total number of buffers needed is k * n. Only one decoder is required to select among the k registers. Size of the decoder should be $\log_2$k-to- k.

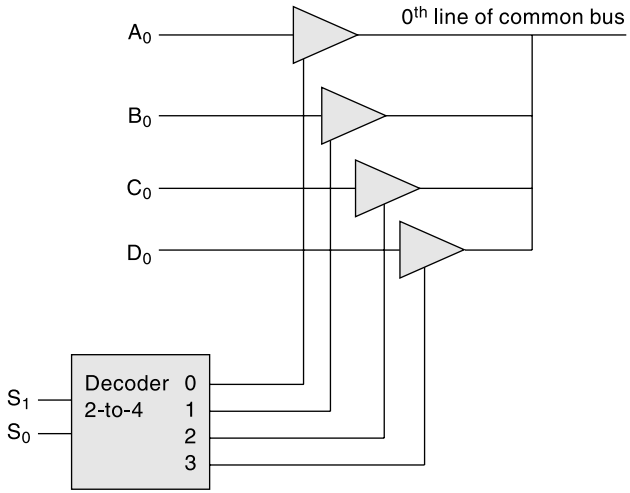


Figure 3.5 *A single line of a bus system with tri-state buffers*

3.6 MEMORY TRANSFER

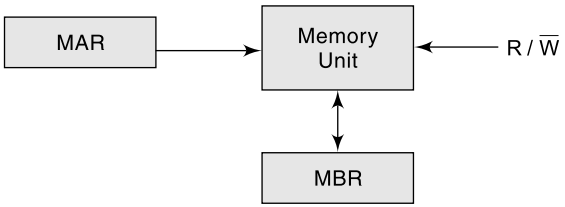


Figure 3.6 *Memory unit communicating with external registers*

When the information is transferred from a memory word it is called a *read* operation and when the information is stored into a memory it is called *write* operation. In both cases the memory word is specified by an address. This memory word is designated by the symbol M. A memory address is specified to select a particular memory word among many available words during the transfer. Consider a memory unit that receives the address from a register, called the memory address register (MAR), as shown in Fig. 3.6. The data from memory is transferred to another register, called memory buffer register (MBR) or data register (DR).

The read operation can be stated as:

Read: MBR ← M[MAR]

This symbolic instruction causes a transfer of data into MBR from the memory word M selected by the address information in MAR.

Let the data is to be transferred from a general-purpose register R1 into a memory word M selected by the address in MAR. The write operation can be stated as:

Write: M[MAR] ← R1

By this symbolic instruction, a data word is transferred from R1 register to the memory word M selected by the register MAR.

3.7 ARITHMETIC MICRO-OPERATION

The basic arithmetic micro-operations are addition, subtraction, increment, decrement and shift. The arithmetic addition micro-operation is defined by the statement

$$R1 \leftarrow R2 + R3$$

This micro-operation states that the content of R2 register is added to the content of R3 register and the result is stored into R1 register. In order to implement this operation with hardware we need three registers and a digital circuit to perform addition. The basic arithmetic micro-operations are listed in the Table 3.1.

Table 3.1 *Basic Arithmetic Micro-operations*

Symbolic notation	*Description*
$R1 \leftarrow R2 + R3$	Added contents of R2 and R3 transferred to R1
$R1 \leftarrow R2 - R3$	Contents of R2 minus R3 transferred to R1
$R1 \leftarrow R1 + 1$	Incrementing the content of R1 by 1
$R1 \leftarrow R1 - 1$	Decrementing the content of R1 by 1
$R1 \leftarrow \overline{R1}$	Complementing the content of R1 (1's complement)
$R1 \leftarrow \overline{R1} + 1$	2's complement the content of R1 (negate)
$R1 \leftarrow R2 + \overline{R3} + 1$	Content of R2 added with 2's complement of R3 (subtraction) and transferred to R1.

In this table, the subtraction micro-operation is performed using 2's complement method. The multiplication and division are not included in this table; though, the operations are two valid micro-operations. The multiplication operation can be implemented by a sequence of add and shift micro-operations. The division can be implemented with a sequence of subtract and shift micro-operations.

3.8 DESIGN OF SOME ARITHMETIC UNITS

3.8.1 Binary Adder

Binary adder is an essential part of every computer, because the micro-operations addition and subtraction of two binary numbers stored in two registers are performed by this unit. A binary adder is a digital circuit that generates the arithmetic sum of two binary numbers of any lengths. The binary adder is basically constructed with full adders (for details of full adder, see Appendix). Binary adders are of two types:

1. Serial Adder
2. Parallel Adder.

Serial Adder A serial adder is an adder, which performs the addition of two binary numbers serially bit by bit starting with lsb. Addition of one bit position takes one clock cycle. The circuit for this adder is shown in Fig. 3.7. The operands are provided bit by bit starting with lsbs. Thus, for an n-bit serial adder, n clock cycles are needed to complete the n-bit numbers' addition. At each cycle, the carry produced by a bit position should be stored in a D-flip-flop and it is given as input during the next cycle through carry-in. Therefore, serial adder is a sequential circuit.

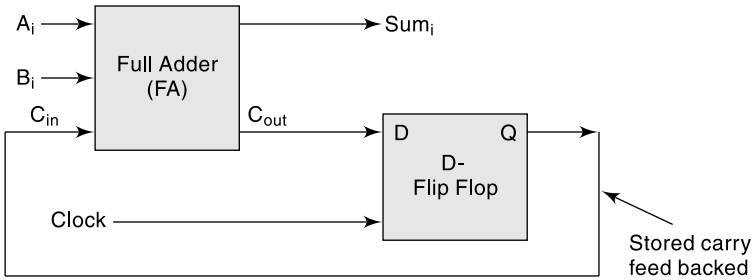


Figure 3.7 *A serial adder*

Advantage The serial adder circuit is small and hence, it is very inexpensive irrespective of the number of bits to be added.

Disadvantage The serial adder is very slow since it takes n clock cycles for addition of two n-bit numbers.

Parallel Adder A parallel adder is an adder, which adds all bits of two numbers in one clock cycle. It has separate adder circuit for each bit. Therefore, to add two n-bit numbers, parallel adder needs n separate adder circuits. There are basically two types of parallel adders, depending on the way of carry generation:

(a) Carry-Propagate Adder (CPA) or Ripple Carry Adder (RCA)
(b) Carry Look-ahead Adder (CLA).

Carry-Propagate Adder (CPA) For addition of two n-bit numbers, n full adders (FAs) are required. Each full adder's carry output will be the input of the next higher bit full adder. Each full adder performs addition for same position bits of two numbers. An n-bit CPA circuit is shown in the Fig. 3.8.

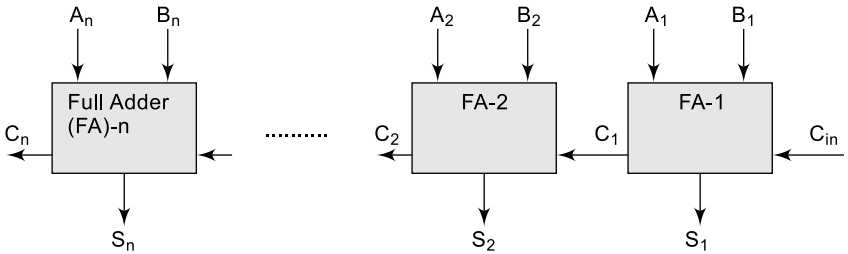


Figure 3.8 *An n-bit Carry-Propagate Adder (CPA)*

The addition time is decided by the delay introduced by the carry. In worst case, the carry from the first full adder stage has to propagate through all the full adder stages. Therefore, the maximum propagation delay for n-bit CPA is $\Delta \times n$, where Δ is the time delay for each full adder stage and n is the number of bits in each operand.

Advantage This adder, being a combinational circuit, is faster than serial adder. In one clock period all bits of two numbers are added.

Disadvantages

1. The addition delay becomes large, if the size of numbers to be added is increased.
2. The hardware cost is more than that of serial adder. Because, number of full adders needed is equal to the number of bits in operands.

Building Long Adder Since carry is propagated serially through each full adder, smaller size CPAs can be cascaded to obtain a large CPA. As an example, construction of 16-bit CPA using four 4-bit CPAs is shown in the Fig. 3.9

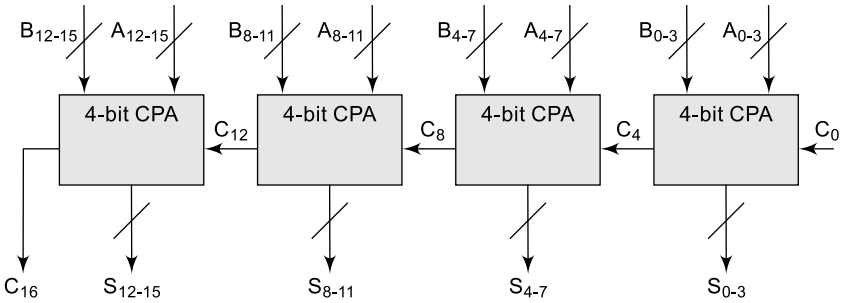


Figure 3.9 *Implementation of a 16-bit CPA using 4-bit CPAs*

Carry Look-ahead Adder (CLA) A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1}=A_iB_i + (A_i+B_i)C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in Fig. 3.10. The sum bit $S_i = A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

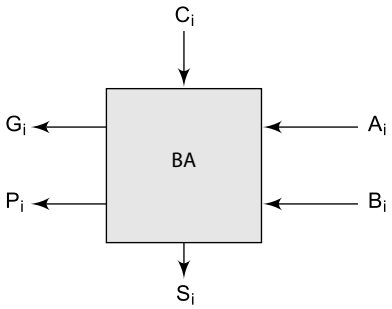


Figure 3.10 *Basic adder*

Now, we want to design a 4-bit CLA, for which four carries C_1, C_2, C_3 and C_4 are to be generated. Using equation number (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1= G_0 + P_0C_0$$
$$C_2= G_1 + P_1C_1$$
$$C_3= G_2 + P_2C_2$$
$$C_4= G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$
$$= G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$

$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$
$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \quad (5)$$

The equations (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in Fig. 3.11.

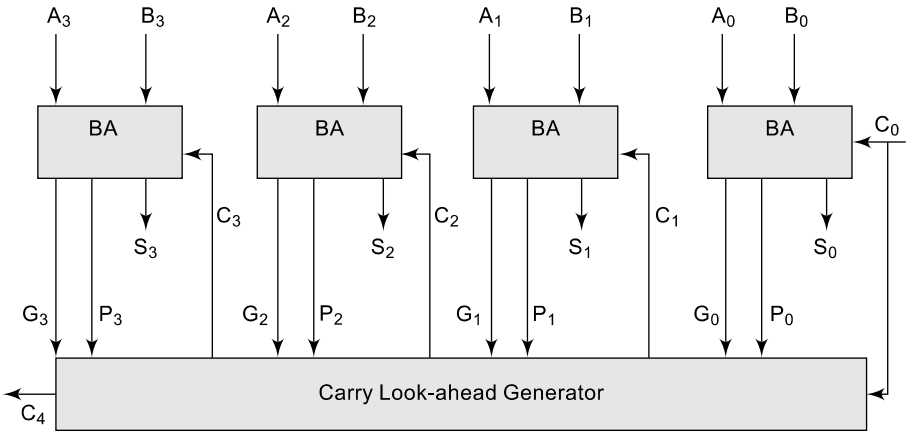


Figure 3.11 *4-bit carry look-ahead adder (CLA)*

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

Carry-Save Adder (CSA) The various types of adders we have discussed so far can add two numbers only. In parallel processing and in multiplication and division, multi-operand addition is often encountered. More powerful adders are required which can add many numbers instead of two together. Such type high-speed multi-operand adder is called a carry-save adder (CSA). To see the effectiveness, consider the following example:

34
62
58
76
---
10 ← Sum vector
22 ← Carry vector
---
230 ← Final result.

In this example, four decimal numbers are added. First, the unit place digits are added, and producing a sum of 0 and a carry digit of 2. Similarly the ten place digits are added, producing a sum of 1 and a carry digit of 2. These summations can be performed in parallel to produce a sum vector of 10 and a carry vector of 22, because there is no carry propagation from the unit place digit to the tenth place digit. When all digits of the operands are added, the sum and the shifted carry vector are added in the conventional manner, i.e. using either CPA or CLA, which produces the final answer.

The CSA takes three numbers as inputs, say, X, Y and Z, and produces two outputs, the sum vector S and carry vector C. The sum vector S and carry vector C are obtained by the following relations:

$$S = X \oplus Y \oplus Z$$

$C = XY + YZ + XZ$; Here all logical operations are performed bit-wise.

The final arithmetic sum of three inputs, i.e. Sum = X + Y + Z, is obtained by adding the two outputs, i.e. Sum = S + C, using a CPA or CLA.

Let us take one example to illustrate the CSA technique.

X =	0 1 0 1 1 0
Y =	1 1 0 0 1 1
Z =	0 0 1 1 0 1
S =	1 0 1 0 0 0
C = 0	1 0 1 1 1
Sum = S + C =	1 0 1 0 1 1 0

The carry-save addition process can be implemented in fully parallel mode or in series-parallel mode. Let us consider the 4-operand summation:

$$Sum = X + Y + Z + W$$

where X, Y, Z and W are 4-bit operands. The block diagram representation of this summation process is shown in Fig. 3.12(a).

In this system, the first carry-save adder (CSA) adds X, Y and Z and produces a sum vector (S^1) and a carry vector (C^1). The sum vector, the shifted carry vector and the fourth operand W are applied as the inputs to the second CSA. The results produced by the second CSA are then added by a CPA to generate the final summation, Sum.

The carry is propagated only in the last step. So, the total time required to add 4 operands is:

Time (4) = 2 * [CSA add time] + [CPA add time]

In general, time to add n operands by this method is:

Time (n) = (n–2) * [CSA add time] + [CPA add time]

This result can again be improved by using a CLA in the last stage, instead of CPA.

3.8.2 Binary Incrementer Unit

The binary incrementer unit is used to perform the increment micro-operation. The increment micro-operation adds one to the number stored in a register. For example, if a 4-bit register has a binary value 1001, after increment operation, it will be 1010. This micro-operation can be implemented two ways, one is by using binary up counter (for counter, see Appendix) and other is by using combinational circuit. Some times, it is required to perform the operation using combinational circuits. The

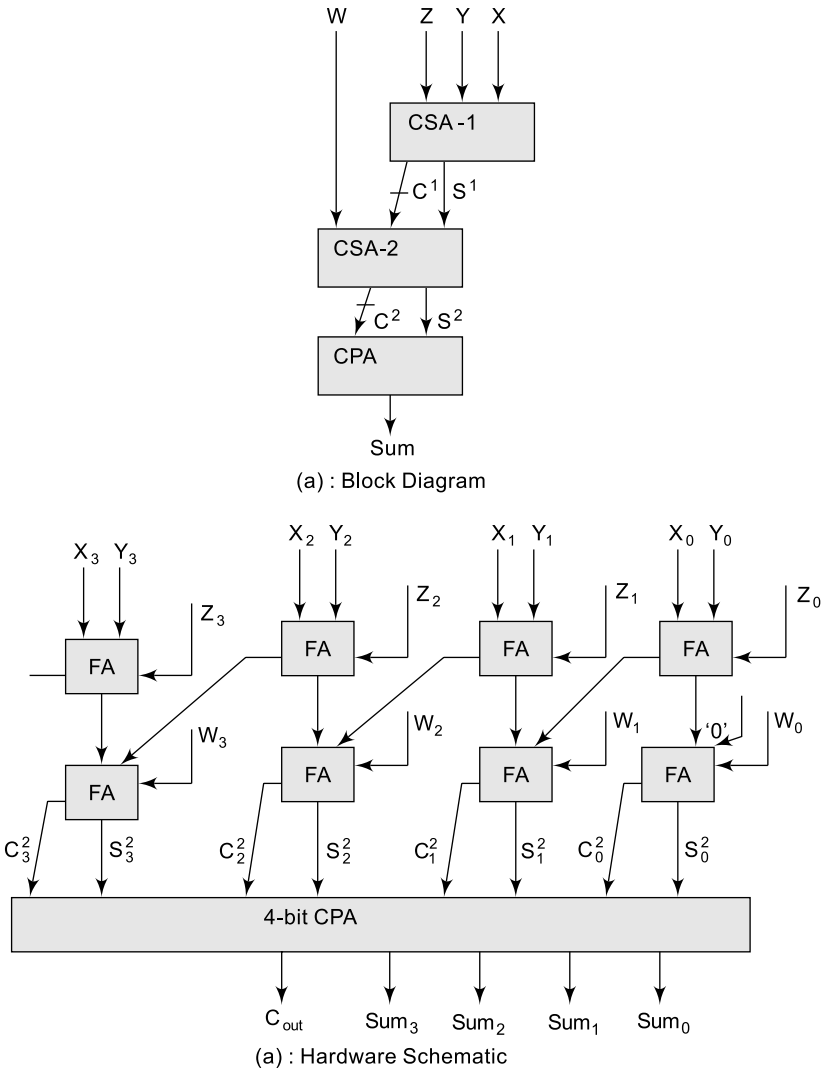


Figure 3.12 *4-operand summation*

diagram of a 4-bit combinational circuit incrementer is shown in Fig. 3.13. Here, four half adders (HA) (for half adder, see Appendix) are connected in cascade. Note that, the one of the inputs of the least significant stage HA is connected to logic '1'.

The circuit in the Fig. 3.13 can easily be extended to design an n-bit incrementer by using more number of half adders. Instead of half adders, full adders can be used in the incrementer circuit where one of the inputs of each full adder is connected to logic '0' and first stage full adder's carry input is fixed to logic '1'.

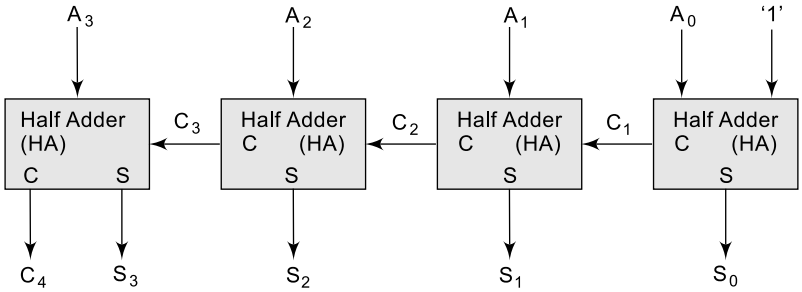


Figure 3.13 *4-bit incrementer circuit*

3.8.3 Binary Decrementer Unit

The binary decrementer unit performs the decrement micro-operation. The decrement micro-operation subtracts value one from the number stored in a register. For example, if a 4-bit register has a binary value 1001, it will be 1000 after the decrement operation. The same operation can easily be implemented using combinational circuit half subtractors or sequential circuit binary down counter (see Appendix). There may be occasions when the decrement micro-operation must be realized with combinational circuit full adders. The subtraction of two binary numbers can be performed by taking the 2's complement of the subtrahend and then adding it to the minuend, as discussed in Section 2.4.1. The diagram of a 4-bit combinational decrementer circuit has been implemented using full adders, shown in Fig. 3.14.

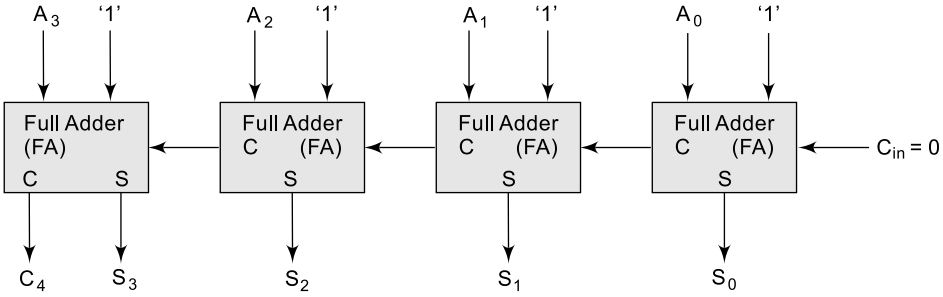


Figure 3.14 *4-bit decrementer circuit*

Here, we are adding a bit 1 as one of the inputs to the full adder. This means that binary number (1111) is added with the operand number A. The binary number (1111) means –1 in decimal, since the negative number is represented in computers using signed 2's complement method. That means, we are adding –1 with the operand value stored in register A.

3.8.4 Binary Adder-Subtractor Unit

Recall that the subtraction A–B is equivalent to A+2's complement of B (i.e. 1's complement of B +1). The addition and subtraction can be combined to a single circuit by using exclusive-OR (XOR) gate with each full adder. The circuit is shown in the Fig. 3.15.

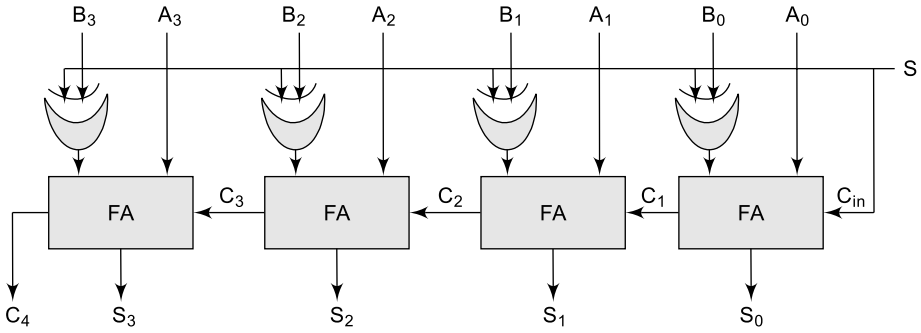


Figure 3.15 *4-bit binary adder-subtractor*

The selection input S determines the operation. When S = 0, this circuit performs the addition operation and when S = 1, this circuit performs subtraction operation. The inputs to each XOR gate are S-input and B- input. When S = 0, we have $0 \oplus B = B$ (It can be verified from the truth table of XOR gate). This means that the direct B-value is given as input into a full adder (FA) and the carry-input into first full adder is 0. Thus, the circuit performs addition. When S = 1, we have $1 \oplus B = \overline{B}$ (It can be verified from the truth table of XOR gate) and carry-input is 1. This means that the circuit performs the addition of A with 2's complement of B. For unsigned numbers, A – B if A ≥ B or the 2's complement of (B – A) if A < B. For signed numbers, the result is A – B provided that there is no overflow.

3.8.5 Arithmetic Unit

The basic arithmetic micro-operations listed in Table 3.1 can be implemented in one composite arithmetic unit. The diagram of a 4-bit arithmetic circuit is shown in Fig. 3.16. The circuit has a 4-bit parallel adder and four multiplexers for 4-bit arithmetic unit. There are two 4-bit inputs A and B, and the 5-bit output is K. The size of each multiplexer is 4:1. The two common selection lines for all four multiplexers are S_0 and S_1. C_{in} is the carry input of the parallel adder and the carry out is C_{out}. The four inputs to each multiplexer are B- value, $\overline{B}$-value, logic–1 and logic–0.

The output of the circuit is calculated from the following arithmetic sum:

$$K = A + Y + C_{in}$$

where A is a 4-bit number, Y is the 4-bit output of multiplexers and C_{in} is the carry input bit to the parallel adder. By this circuit it is possible to get 8 arithmetic micro-operations, as listed in the Table 3.2.

Case 1: When $S_1\ S_0 = 00$.

In this case, the values of B are selected to the Y inputs of the adder. If $C_{in} = 0$, output K = A + B. If $C_{in} = 1$, output K = A + B + 1. In both cases the micro-operation addition is performed without carry or with carry input.

Case 2: When $S_1\ S_0 = 01$.

The complements of B are selected to the Y inputs of the adder. If $C_{in} = 0$, output $K = A + \overline{B}$. This means the operation is subtraction with borrow. If $C_{in} = 1$, output $K = A + \overline{B} + 1$, which is equivalent to A + 2's complement of B. Thus this gives the subtraction A – B.

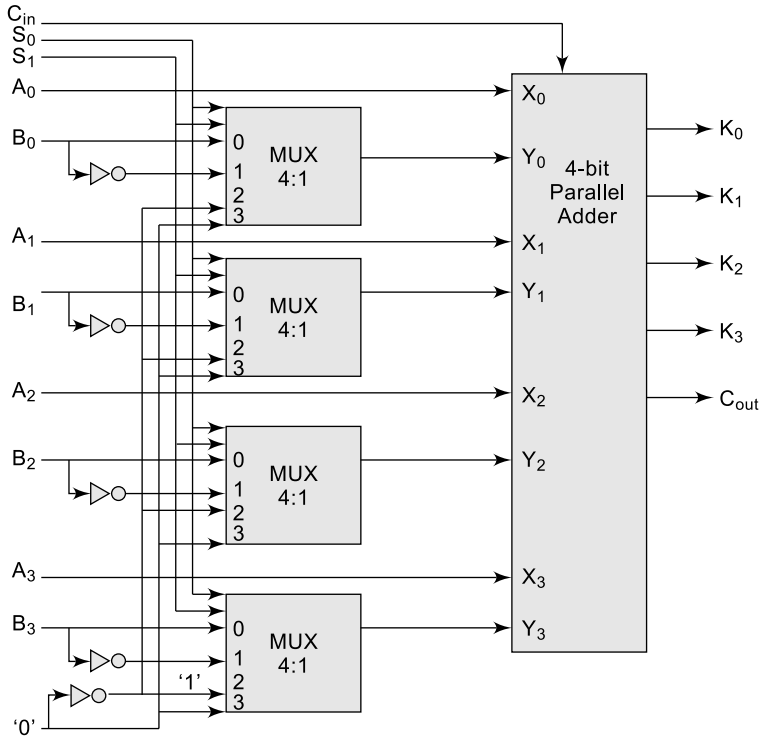


Figure 3.16 *4-bit arithmetic unit*

Case 3: When $S_1\ S_0 = 10$.
Here, all 1s are selected to the Y inputs of the adder. This means Y = (1111), which is equivalent to 2's complement of decimal 1, that means, Y = –1. If $C_{in} = 0$, the output K = A –1, which is a decrement operation. If $C_{in} = 1$, the output K = A – 1 + 1 = A. This causes the direct transfer of A to K.

Case 4: When $S_1\ S_0 = 11$.

In this case, all 0s are selected to the Y inputs of the adder. If $C_{in} = 0$, the output K = A, which is a transfer operation. If $C_{in} = 1$, output K = A + 1. This means the value of A is incremented by 1.

Observe that only seven different arithmetic micro-operations are deduced, because the transfer operation is generated twice.

Table 3.2 *Arithmetic unit Function Table*

S_1	S_0	C_{in}	Y	$K = A + Y + C_{in}$	*Operation*
0	0	0	B	K = A + B	Addition
0	0	1	B	K = A + B + 1	Addition with carry
0	1	0	$\overline{B}$	K = A+ $\overline{B}$	Subtraction with borrow
0	1	1	$\overline{B}$	K = A + $\overline{B}$ +1	Subtraction
1	0	0	1	K = A – 1	Decrement
1	0	1	1	K = A	Transfer
1	1	0	0	K = A	Transfer
1	1	1	0	K = A + 1	Increment

3.9 LOGIC UNIT

Logic unit is needed to perform the logical micro-operations such as OR, AND, XOR (exclusive-OR), complement, etc on individual pairs of bits stored in registers. For example, the OR micro-operation between the contents of two registers R2 and R3 can be stated as

$$C: R1 \leftarrow R2 \vee R3.$$

This symbolic instruction specifies the OR micro-operation to be performed on the contents of registers R2 and R3 bitwise, provided that the control variable C = 1.

Special symbols are used for logic micro-operations XOR, AND and complement (some times called NOT). The symbol $\oplus$ is used to denote XOR micro-operation. The symbol $\wedge$ is used to designate AND micro-operation and the symbol $^-$ (or ‘) used as a bar on the top of a register name indicates the (1’s) complement or NOT micro-operation.

Now, we will design a logic unit that can perform the four basic logic micro-operations: OR, AND, XOR and complement. Because from these four micro-operations, all other logic micro-operations can be derived. A one-stage logic unit for these four basic micro-operations is shown in the Fig. 3.17. The logic unit consists of four gates and a 4:1 multiplexer. The outputs of the gates are applied to the data inputs of the multiplexer. Using two selection lines, one of the data inputs of the multiplexer is selected as the output. The ith stage is shown using subscript i. For a logic unit with n bits, the diagram must be repeated n times for i = 1, 2, 3, …n. Then the common selection lines are applied to all the stages. For example, to design a 4-bit logic unit, four 4:1 multiplexers and 16 gates (out these, 4 OR-gates, 4 AND-gates, 4 XOR-gates and 4-NOT gates) are required. The corresponding function table is shown in the Table 3.3.

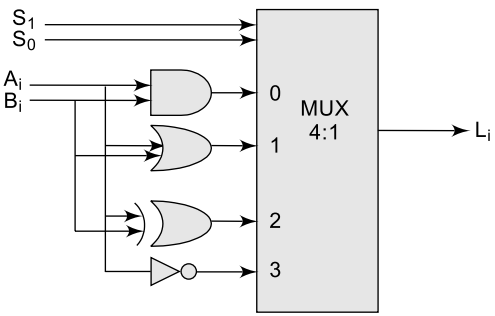


Figure 3.17 *One ith stage logic unit*

Table 3.3 *Function Table for Logic Unit*

S_1	S_0	*Output (L)*	*Operation*
0	0	$L = A \wedge B$	AND
0	1	$L = A \vee B$	OR
1	0	$L = A \oplus B$	XOR
1	1	$L = \overline{A}$	Complement of A

3.10 SHIFTER UNIT

Shifter unit is used to perform shift micro-operations. Shift micro-operations are used to transfer stored data serially. The shifting of bits of a register can be in either direction, left or right. Shift micro-operations can be classified into three categories:

(a) Logical (b) Circular (c) Arithmetic.

In logical shift, all bits including sign bit take part in the shift operation. A bit 0 is entered in the vacant extreme bit position (left most or right most). As a result, the left-most bit is lost, if it is the left shift operation. Similarly, the right-most bit is lost, if it is the right shift operation. We use the symbols lsl and lsr for left shift and right shift micro-operations respectively.

In circular shift (also known as rotation operation), one bit shifted out from one extreme bit position enters the other extreme side's vacant bit position as shown in Fig. 3.18. No bit is lost or added.

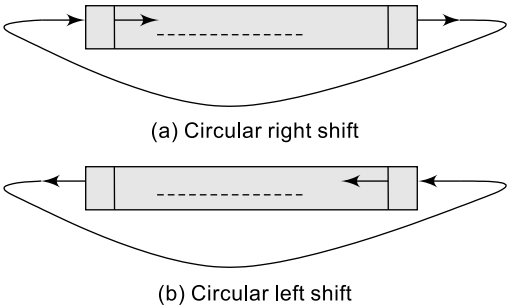


Figure 3.18 *Circular shift micro-operation*

In arithmetic shift, sign bit remains unaffected and other bits (magnitude bits) take part in shift micro-operation, as shown in Fig. 3.19. As a result of the arithmetic left shift operation, the left-most bit of the magnitude part is lost and extreme right vacant bit is filled in with 0. Similarly, the right-most bit is lost and vacant left most bit of the magnitude part is filled in with the sign bit of the number, if it is the arithmetic right shift operation.

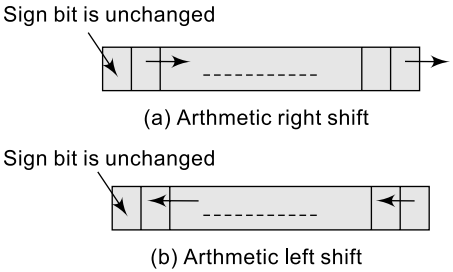


Figure 3.19 *Arithmetic shift micro-operation*

Shifter unit can be constructed using bidirectional shift register with clock circuit. However, it would be more efficient for a processor having many registers to implement the shifter unit with a combinational circuit. A combinational shifter unit can be constructed with multiplexers as shown in Fig. 3.20. The content of a register that has to be shifted

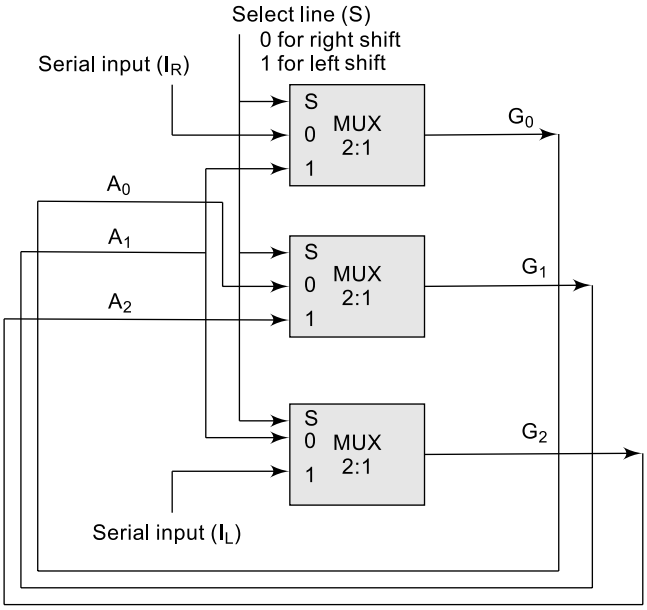


Figure 3.20 *3-bit combinational shifter unit*

is first placed onto a common bus and the shifted number is then loaded back into the register. This requires one clock pulse for loading the shifted number into the register. A 3-bit shifter is shown in figure whose three data inputs are A_0, A_1 and A_2, and three data outputs are G_0, G_1 and G_2. There are two serial inputs, one I_L for left shift operation and another I_R for right shift operation. If the selection line S is 0, the stored input data is shifted right. If the selection line S is 1, the stored input data is shifted left. The function table shown in Table 3.4 illustrates shift micro-operations.

Table 3.4 *Function Table for Combinational Shifter Unit*

Selection line	*Output*		
S	G_0	G_1	G_2
0	I_R	A_0	A_1
1	A_1	A_2	I_L

A shifter unit for n data inputs and outputs requires n multiplexers.

3.11 ARITHMETIC LOGIC UNIT (ALU)

The arithmetic, logic and shifter units introduced earlier can be combined into one ALU with common selection lines. The shift micro-operations are often performed in a separate unit, but sometimes the shifter unit is made part of the overall ALU. Since the ALU is composed of three units, namely, arithmetic, logic and shifter units. For 4-bit ALU, four multiplexers for arithmetic unit are needed each of size 4×1, four multiplexers for logic unit are needed each of size 4×1 and four multiplexers for shifter unit are needed each of size 2×1. A complete block diagram schematic of a 4-bit ALU is shown in Fig. 3.21.

The set of four multiplexers each of 4:1 at output end chooses among arithmetic output in K, logic output in L and shift output in G. A particular arithmetic or logic micro-operation is selected with selection inputs S_1 and S_0. The final output of the ALU is determined by the set of multiplexers with selection lines S_3 and S_2. The function table for the ALU is shown in the Table 3.5. The table lists 14 micro-operations: 8 for arithmetic, 4 for logic and 2 for shifter unit. For shifter unit, the selection line S_1 is used to select either left or right shift micro-operation.

Table 3.5 *Function Table for ALU*

S_3	S_2	S_1	S_0	C_{in}	*Output (F)*	*Operation*
0	0	0	0	0	F = A + B	Addition
0	0	0	0	1	F = A + B + 1	Addition with carry
0	0	0	1	0	F = A + B′	Subtraction with borrow
0	0	0	1	1	F = A + B′ + 1	Subtraction
0	0	1	0	0	F = A – 1	Decrement A
0	0	1	0	1	F = A	Transfer A
0	0	1	1	0	F = A	Transfer A
0	0	1	1	1	F = A + 1	Increment A

(*Contd.*)

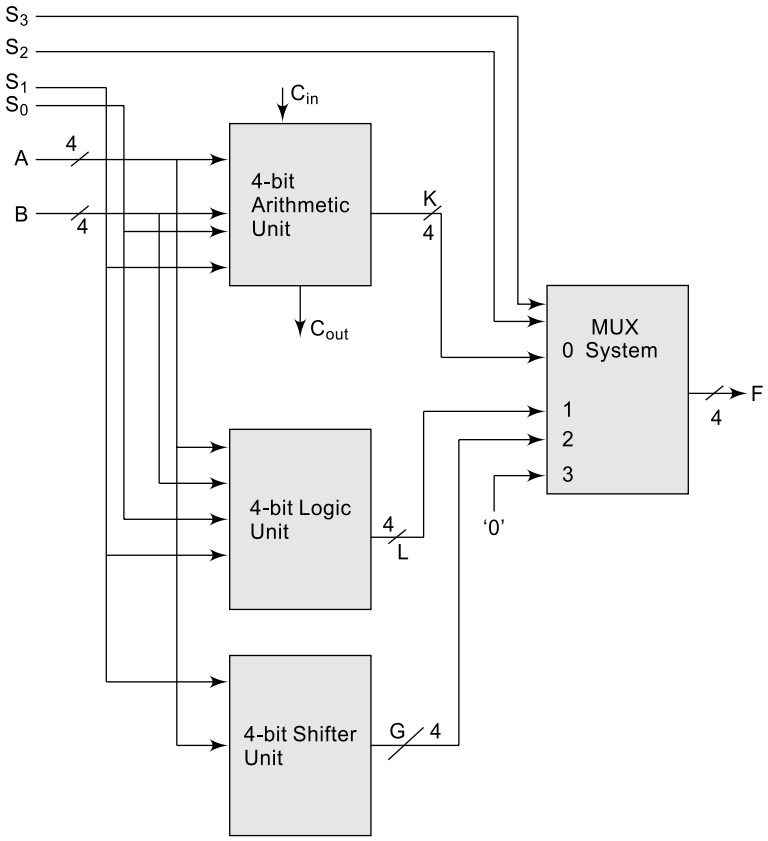


Figure 3.21 *4-bit 14-function ALU*

(*Contd.*)

0	1	0	0	x	F = A ∧ B	AND
0	1	0	1	x	F = A ∨ B	OR
0	1	1	0	x	F = A ⊕ B	XOR
0	1	1	1	x	F = $\bar{A}$	Complement of A
1	0	0	x	x	F = lsr A	Shift right A into F
1	0	1	x	x	F = lsl A	Shift left A into F

The rapid growth in IC technology permitted the manufacturers to produce an ALU as an MSI block. Such systems implement many operations and their use as a system component reduces the hardware cost, board space, debugging effort and failure rate. Usually, each MSI ALU chip is designed as a 4-bit slice. However, a designer can easily interconnect *n* such chips to get a 4*n*-bit ALU. The most popular 4-bit ALU chips are the 74381 and 74181. The 74381 ALU performs 3 arithmetic, 3 Boolean and 2 miscellaneous operations on 4-bit operands. The 74181 ALU performs 16 arithmetic and 16 Boolean operations on two 4-bit operands.

3.12 BIT-SLICE PROCESSORS

The chip count which results in high power dissipation and poor reliability is the primary concern of MSI functional blocks.

To eliminate this difficulty, a new approach called the bit-slice *technique* became feasible because of advances in IC technology. A bit-slice processor chip typically includes an ALU and a few registers. Using bit slices, a designer can build a processor of any word length. For example, a 4-bit slice processor includes a register file (i.e., collection of registers) and an ALU for performing operations on 4-bit data, so that four such chips can be combined to build a 16-bit processor unit. Examples of bit-slice processors are Intel's 3000 and AMD's (Advanced Micro-Devices') AM2901.

Figure 3.22 shows how a 16-bit processor can be constructed from four 4-bit processor slices.

The data buses and register files of the individual slices are effectively cascaded to increase their size from 4 to 16 bits. The control lines that select and sequence the operations to be performed are connected to every slice so that all slices execute the same actions in lockstep with one another. Thus each slice performs the same operation on a different 4-bit part (slice) of the input operands and

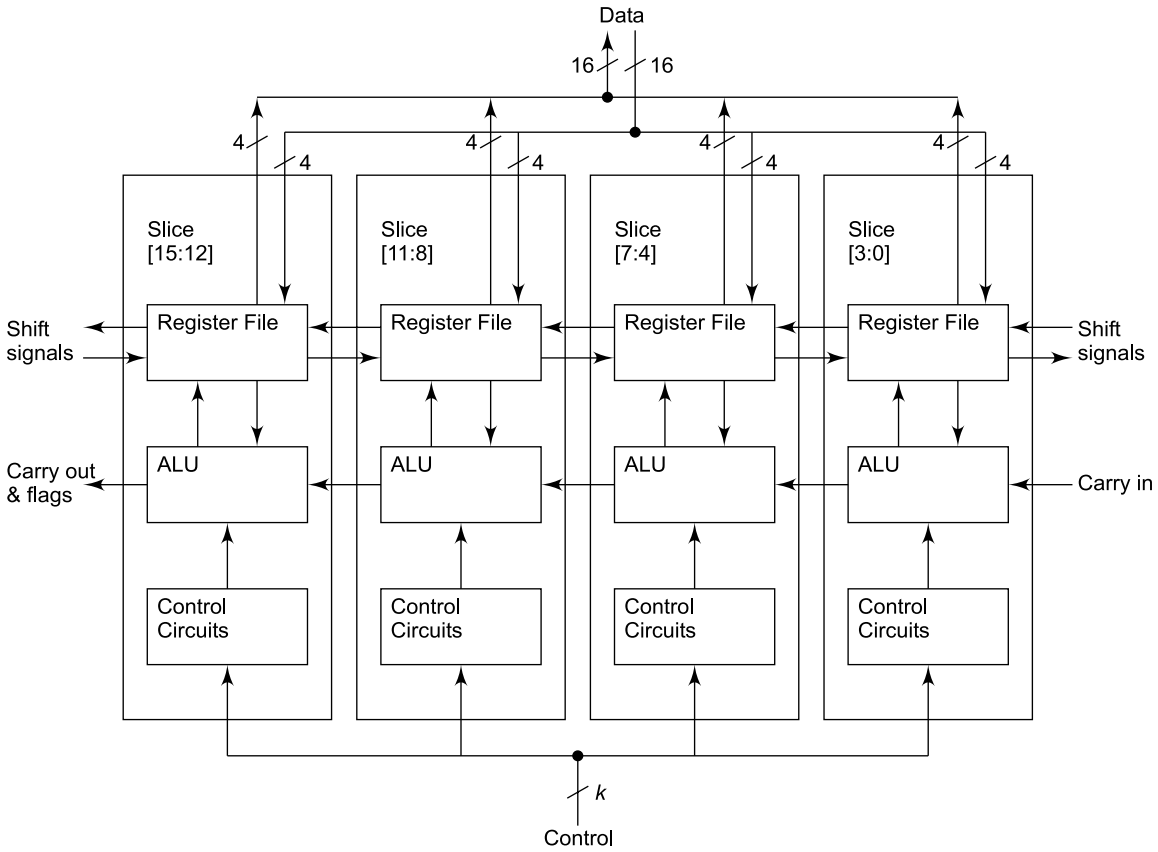


Figure 3.22 *16-bit processor composed of four 4-bit slices*

produces only the corresponding part of the results. The required control signals are derived from an external control unit, which can be hardwired or microprogrammed. Certain operations require information to be exchanged between slices. For example, to implement a shift operation, each slice must be able to send a bit to, and receive a bit from, its left or right neighbours. Similarly, when performing addition or subtraction, carry bits must be transmitted between neighbouring slices. For this purpose horizontal connections are provided between the slices as shown in Fig. 3.22.

SOLVED PROBLEMS

1. *Why every computer system is associated with a set of general-purpose registers?*

Answer

Some general-purpose registers are used inside processor to enhance the effective execution speed. The registers are fastest storage devices whose speed is almost same as processors in computer systems and are used to hold the instructions and data temporarily. The processors do not have to wait for required instructions and data, if they are available in registers.

2. *A digital computer has a common bus system for k-registers of n bits each. The bus is constructed with multiplexers.*
   (a) *What size of multiplexers is needed?*
   (b) *How many multiplexers are there in the bus?*

Answer

For an n-line bus system for k-registers of n bits each:

(a) The size of each multiplexer must be k × 1, since it multiplexes k data lines each from a register.
(b) Each multiplexer transfers one bit of the selected register. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.

3. *A digital computer has a common bus system for k-registers of n bits each. The bus is constructed with tri-state buffers.*
   (a) *How many decoders are needed and what size of decoder is needed?*
   (b) *How many tri-state buffers are there in the bus?*

Answer

For an n-line bus system for k-registers of n bits each:

(a) Only one decoder is required to select among the k-registers. Size of the decoder should be $\log_2$k-to- k.
(b) The total number of buffers needed is k * n.

4. *Show the circuit diagram for implementing the following register transfer operation: if ($\bar{a}b$ = 1) then R1 ← R2 else R1 ← R3; where a and b are control variables.*

Answer

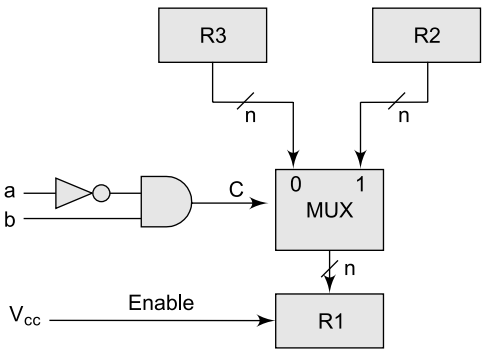
The control function C is $\bar{a}b$. The register transfer operation can be written as:

C: R1 ← R2

C': R1 ← R3

The circuit diagram for the register transfer operations is shown below.

The R2 register is selected by the MUX if control condition C = 1; otherwise register R3 is selected as source register.



Hardware implementation of "if ($\bar{a}b$ = 1) then R1 ← R2 else R1 ← R3"

5. *Two unsigned numbers of 2-bit each are to be added. Which adder is faster: serial adder or ripple carry adder?*

Answer

In serial adder, the propagation delay of the flip-flop, t_f also contributes to total addition delay. If t_s is the delay for single stage adder, the minimum period of the clock must be $(t_s + t_f)$. Hence, the minimum time needed by the serial adder is $2 \times (t_s + t_f)$.

In ripple carry adder, addition time = $2 \times t_s$.

Thus, the ripple carry adder is faster.

6. *Suppose a 16-bit ripple carry adder (RCA) is constructed using 4-bit RCA as building block. What is the maximum addition delay of the adder?*

Answer

To generate S_{15}, C_{15} must be available. The generation of C_{15} depends on the availability of C_{14}, which in turns must wait for C_{13} to become available. The maximum delay for such adder can be computed as:

$15 \times 2\Delta$ (for carry to propagate through 15 full adders) + 3Δ (for S_{15} generation from C_{15}) = 33Δ.

7. *Suppose a 16-bit carry look-ahead adder (CLA) is constructed using 4-bit CLA as building block. What is the maximum addition delay of the adder?*

Answer

The maximum delay for such adder can be computed as:

Δ(for G_i, P_i generation) + 2Δ (for C_4 generation from C_0) + 2Δ (for C_8 generation from C_4) + 2Δ (for C_{12} generation from C_8) + 2Δ (for C_{15} generation from C_{12}) + 3Δ (for S_{15} generation from C_{15}) = 12Δ.

8. *Why CLA is called fast parallel adder?*

Answer

In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages. These input carries depend only on the initial carry C_0. For this reason, CLA is fast parallel adder.

9. *If the average gate delay is 4 ns, what is the delay for an 8-bit carry look-ahead adder?*

Answer

For carry look-ahead adder, the addition delay is 6 × gate delay = 6 × 4 ns = 24 ns.

10. *Two 4-bit unsigned numbers are to be multiplied using the principle of carry save adders. Assume the numbers to be $A_3\ A_2\ A_1\ A_0$ and $B_3\ B_2\ B_1\ B_0$. Show the arrangement and interconnection of the adders and the input signals so as to generate an 8-bit product as $P_7\ P_6 P_5\ P_4\ P_3\ P_2\ P_1\ P_0$.*

Answer

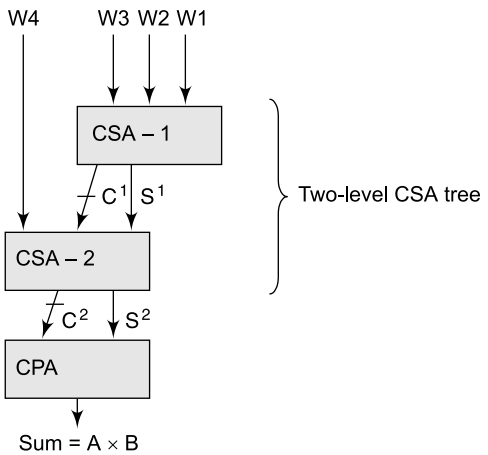
The multiplication of two unsigned is done by repeated add-shift operations. Add-shift multiplication of two 4-bit numbers is illustrated in figure below.

				A3	A2	A1	A0 = A
				B3	B2	B1	B0 = B
				A3B0	A2B0	A1B0	A0B0 = W1
			A3B1	A2B1	A1B1	A0B1	= W2
		A3B2	A2B2	A1B2	A0B2		= W3
	A3B3	A2B3	A1B3	A0B3			= W4
P7	P6	P5	P4	P3	P2	P1	P0 = A × B = Product

Figure *Add-shift multiplication of two 4-bit numbers (A × B = Product)*

The additions of partial products W1, W2, W3 and W4, which are generated using bit-wise AND logic operations, can be done using CSA-tree as shown in figure below to realize the multiplier for 4-bit numbers.

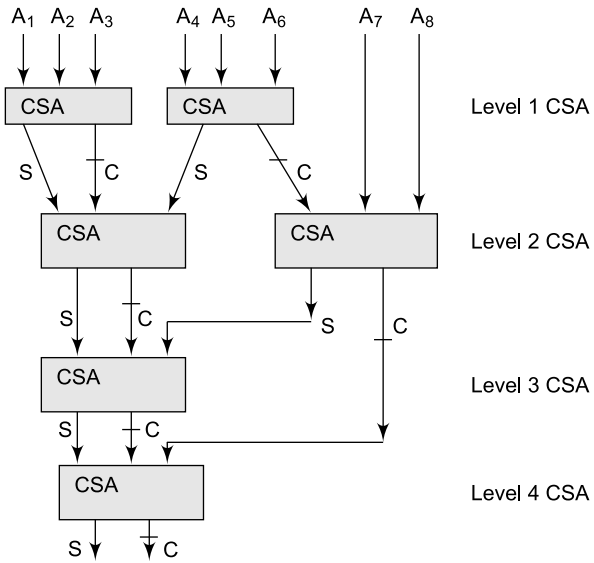
The first carry-save adder (CSA-1) adds W1, W2 and W3 and produces a sum vector (S^1) and a carry vector (C^1). The sum vector, the shifted carry vector and the fourth partial product W4 are applied as the inputs to the second CSA. The results produced by the second CSA are then added by a CPA to generate the final summation Sum.



11. *How many CSA levels are needed to reduce 8 summands to 2?*

Answer

Let, 8 summands be denoted as A_1, A_2, A_3, … A_8. The schematic representation of the carry save addition is shown in figure below. The diagram shows that addition requires four-level CSAs.



12. *Suppose register A holds the 8-bit number 11011001. Determine the B operand and the logic micro-operation to be performed in order to change the value in A to:*

(a) *01101101*

(b) *11111101*

Answer

(a) If B = 1011 0100 and the XOR logic micro-operation is performed with A, then the content of A will be 0110 1101. That is, $A \leftarrow A \oplus B$.

(b) If OR logic micro-operation is performed between A and B, then the content of A will be 1111 1101. That is, $A \leftarrow A \vee B$.

13. *Suppose register A holds the 8-bit number 11011001. Determine the sequence of binary values in A after an arithmetic shift-right, followed by a circular shift-right and followed by a logical shift-left.*

Answer

The value in A = 1101 1001

After arithmetic shift right, the value in A = 1110 1100

After circular shift right, the value in A = 0111 0110

After logical shift left, the value in A = 1110 1100

14. *What is bit-slice processor? Give some examples.*

Answer

A bit-slice processor is constructed from processor modules of smaller bit width. Each of these processes one bit field of an operand. Each processor module chip typically includes an ALU and a few registers. Using smaller size processors, one can design a processor of any word length. For example, a 4-bit bit slice processor includes a register file and an ALU to perform operations on 4-bit data, so that four such chips can be combined to build a 16-bit processor unit.

Some of bit-slice processors are Intel's 3000-family and AMD's AM2900 family.

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:

(i) Microoperation in computers is an operation
- (a) in ALU
- (b) on stored data in register
- (c) in control unit
- (d) performed by the operating system.

(ii) A register is
- (a) a part of main memory
- (b) a part of CPU
- (c) collection of flip-flops
- (d) Both (b) and (c).

(iii) The address of the next instruction to be executed is held by
- (a) AC (accumulator) register
- (b) IR (instruction register)
- (c) PC (program counter) register
- (d) SP (stack pointer).

(iv) A bus in the computer system is
- (a) collection of some individual lines each is used to send a bit randomly
- (b) collection of parallel lines each is used to send one bit synchronously
- (c) collection of lines through which control signals are sent
- (d) collection of lines through which data are sent.

(v) To construct an n-line common bus using MUX for k-registers of n bits each, the number of MUXs and size of each MUX are
- (a) k and $n \times 1$
- (b) n and 2^k
- (c) n and $k \times 1$
- (d) k and 2^n, respectively.

(vi) To construct an n-line common bus using tri-state buffers for k-registers of n bits each, the number of buffers and size of common decoder are
- (a) n and $\log_2$k-to- k
- (b) n*k and $\log_2$n-to- n
- (c) k and $\log_2$n-to- n
- (d) k*n and $\log_2$k-to- k, respectively.

(vii) To add two n-bit numbers in parallel adder, the number of clock periods required is
- (a) n
- (b) 2*n
- (c) 1
- (d) n/2.

(viii) The number of clock periods needed in n-bit serial adder to add two numbers is
- (a) n
- (b) 2*n
- (c) 1
- (d) n/2.

(ix) The maximum propagation delay for n-bit CPA is
- (a) $\Delta * n$
- (b) $6 * \Delta$
- (c) n
- (d) Δ, where Δ is the time delay for each full adder stage.

(x) All carries in the CLA depend on
  (a) final stage carry only
  (b) initial input carry only
  (c) previous stage carry
  (d) Both (a) and (b).

(xi) The maximum propagation delay for n-bit CLA is
  (a) $\Delta * n$
  (b) $6 * \Delta$
  (c) n
  (d) Δ, where Δ is the average gate delay.

(xii) The CSA is
  (a) 2-to-1 converter
  (b) 3-to-1 converter
  (c) 3-to-2 converter
  (d) n-to-2 converter; where n is any positive integer.

(xiii) The minimum number of CSA-levels and minimum number of CSAs required in CSA tree to add seven n-bit operands are
  (a) 3 and 5
  (b) 4 and 4
  (c) 4 and 5
  (d) 5 and 4, respectively.

(xiv) In arithmetic shift operation, 0s are padded in vacant positions for
  (a) left/right shift of sign magnitude numbers
  (b) right shift of 2's complement negative numbers
  (c) left shift of 2's complement negative numbers
  (d) right/left shift of 1's complement numbers.

(xv) The difference between an 8-bit composite adder-subtractor made from full adder subunits is in the use of
  (a) 8 extra carry bits
  (b) 8 XORs
  (c) 8-bit inverter and one 8-bit increment circuit
  (d) 8 NOT gates.

(xvi) How many 8-bit ALU slices can be used for designing a 32-bit ALU? Assume that 32-bit operations take nearly four times the 8-bit slice time.
  (a) 4 in parallel
  (b) 4 in parallel plus 4 in series
  (c) 8 in series
  (d) 4 in series.

Group B

2. Show the circuit diagram for implementing the following register transfer operation:
   if ($\bar{a}b = 1$) then R1 $\leftarrow$ R2 else R1 $\leftarrow$ R3; where a and b are control variables.
3. What is bus and what is bus transfer? Why do most computers have a common bus system?
4. Construct a common bus system using MUXs for three registers, each of 4-bits.
5. A digital computer has a common bus system for k-registers of n-bits each. The bus is constructed with MUXs.
   (i) What sizes of MUXs are needed?
   (ii) How many MUXs are there in the bus?
6. What is tri-state buffer? Construct a common bus system using tri-state buffers for two registers of 4-bits each.
7. A digital computer has a common bus system for k-registers of n-bits each. The bus is constructed with tri-state buffers.
   (iii) What size of common decoder is needed?
   (iv) How many tri-state buffers are there in the bus?

8. Explain the significance of timing signals in a computer system.
9. What is memory transfer? What are the different registers associated for memory transfer? Discuss.
10. What is binary adder? What are different types of binary adders?
11. What is serial adder? Discuss it using diagram. What are its merits and demerits?
12. What is parallel adder? What are different types of parallel adders? What are its merits and demerits?
13. Compare and contrast serial adder and parallel adder.
14. What is carry propagate adder (CPA)? Design a 4-bit CPA. What are its merits and demerits? Estimate the maximum propagation delay for n-bit CPA.
15. What is carry look-ahead adder (CLA)? Design a 4-bit CLA. What are its merits and demerits? Estimate the maximum propagation delay for n-bit CLA.
16. Why CLA is called fast parallel adder? Explain.
17. How do you design a 32-bit CPA using 8-bit CPAs? Give the block diagram.
18. What is carry save adder (CSA)? Use one example to illustrate its operation.
19. How many CSA levels are needed to reduce 16 summands to 2 using CSA-tree pattern? Draw the diagram. What is the addition time for n summands?
20. Design a 4-bit incrementer circuit using full adders.
21. Design a 4-bit combinational decrementer circuit.
22. Design an n-bit adder/subtractor composite unit.
23. Design a 3-bit arithmetic unit, which will perform addition, subtraction, increment, decrement and transfer operations.
24. Design a logic circuit that performs four logic operations of XOR, XNOR, NOR and NAND.
25. Suppose register A holds the 8-bit number 11011001. Determine the B operand and the logic micro-operation to be performed in order to change the value in A to:
    (i) 01101101
    (ii) 11111101
26. Suppose register A holds the 8-bit number 11011001. Determine the sequence of binary values in A after an arithmetic shift-right, followed by a circular shift-right and followed by a logical shift-left.
27. Design a 4-bit combinational shifter circuit.
28. Design a 2-bit ALU that performs addition, subtraction, logical AND, logical OR and logical shift operations.
29. Design an arithmetic circuit with one selection variable S and two *n*-bit data inputs A and B. The circuit generates the following four arithmetic operations in conjunction with the carry C_{in}. Draw the logic diagram for the first two stages:

S	$C_{in} = 0$	$C_{in} = 1$
0	K = A + B	K = A + 1
1	K = A – 1	K = A + B' + 1

CHAPTER

4
Memory Organization

4.1 INTRODUCTION

Memory system of a computer is just as important as the CPU in determining its performance because programs and data they operate on are stored in the memory of a computer. The execution speed of programs is highly dependant on the speed with which instructions and data can be transferred between the processor and memory. It is also very important to have a large memory to enhance the execution of programs that are large and deal with huge amounts of data.

Ideally, we would like to have the memory which would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three requirements simultaneously. If we increase the speed and capacity, then cost will increase. We can achieve these goals at optimum level by using several types of memories.

4.2 MEMORY PARAMETERS

There are three basic parameters in choosing a memory:

- Capacity
- Speed
- Bandwidth or Data Transfer Rate

Capacity The capacity of the memory is an important factor that characterizes the size of a computer. Memory can be viewed as a storage unit containing m number of locations (addresses), each of which stores n numbers of bits, as shown in Fig. 4.1. In other words, the memory has m addresses and with word length of n bits. Each word is addressed uniquely by $\log_2$ m number of bits. All n bits of a word are read or stored in one basic operation. The total capacity of the memory is expressed as m × n-bit or m-word memory.

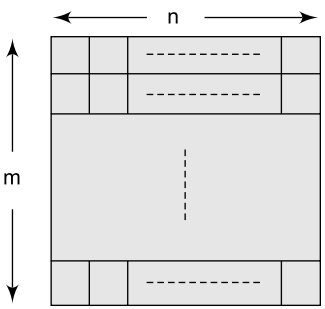


Figure 4.1 *A memory with m locations of each n bits*

The maximum capacity of a memory is determined by the addressing scheme. For example, a 16-bit computer that generates 16-bit addresses is capable of addressing up to 2^{16}= 64K memory locations.

Speed A useful parameter of the memory is its speed of operation, which is the time that elapses between the initiation of an operation and the completion of that operation. This is measured in terms of two parameters: access time, t_A and cycle time, t_C. Sometimes speed is measured in terms of access time and sometimes in terms of cycle time. For example, to perform a read operation, first the address of the location is sent to memory followed by the 'read' control signal. The memory decodes the address to select the location and reads out the contents of the location. The *access time* is the time taken by the memory to complete a read operation from the moment of receiving the 'read' control signal. Generally, access times for read and write are equal. The memory *cycle time* is the minimum time delay required between the initiations of two successive memory operations. For example, the time delay between two successive memory read operations is the memory cycle time. During the first read operation, the information read from memory is available after the access time. This data can be immediately used by CPU. However, the memory is still busy with some internal operation for some more time called recovery time, t_R. During this time, another memory access, read or write cannot be initiated. Only after the recovery time, next operation can be started. The cycle time is the total time including access time and recovery time: $t_C = t_A + t_R$. This recovery time varies with memory technology.

Bandwidth or Data Transfer Rate The maximum amount of information that can be transferred to or from the memory per unit time is called bandwidth and is expressed as number of bytes or words per second. It depends on the speed of access and the width of data bus.

4.3 MEMORY HIERARCHY

The total memory capacity of a computer can be considered as being a hierarchy of components. The memory hierarchy system consists of all storage devices used in a computer system and are broadly divided into following four groups, shown in Fig. 4.2.

- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory

Secondary Memory The slow-speed and low-cost devices that provide backup storage are called secondary memory. The most commonly used secondary memories are magnetic disks, such as hard disk, floppy disk and magnetic tapes. This type of memory is used for storing all programs and data, as this is used in bulk size. When a program not residing in main memory is needed to execute, it is transferred from secondary memory to main memory. Programs not currently needed in main memory (in other words, the programs are not currently executed by the processor) are transferred into secondary memory to provide space for currently used programs and data.

Main Memory This is the memory that communicates directly with CPU. Only programs and data currently needed by the CPU for execution reside in the main memory. Main memory occupies

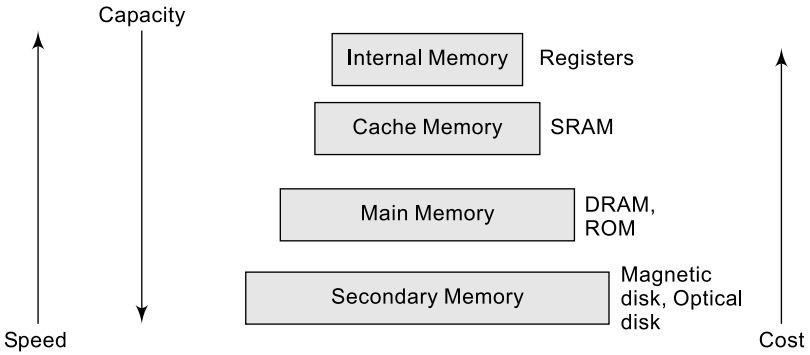


Figure 4.2 *Memory hierarchy*

central position in hierarchy by being able to communicate directly with CPU and with secondary memory devices through an I/O processor, as depicted in Fig. 4.3.

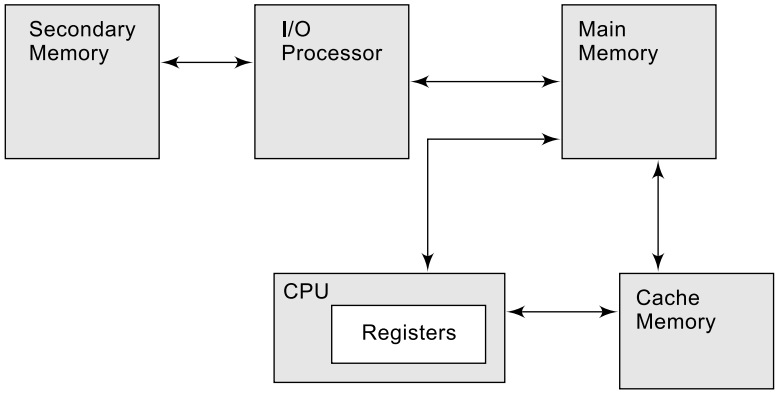


Figure 4.3 *Interconnection between memories and the CPU*

Cache Memory This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as a buffer between the CPU and main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of computer.

Cache memory is one high-speed main memory (SRAM). The cache memory can be placed in more than one level. Most of the recent microprocessors - starting from Intel 80486- have on-chip (memory chip is placed inside the CPU chip) cache memory also known as internal cache. High performance microprocessors such as- Pentium pro and later have two levels of cache memory on-chip. These are known as level 1 (L1) and level 2 (L2) caches. An on-chip cache is slightly faster than an off-chip cache of same technology.

Internal Memory This memory refers to the high-speed registers used inside the CPU. These registers hold temporary results when a computation is in progress. There is no speed disparity between these registers and the CPU because they are fabricated with the same technology. However, since registers are very expensive, only a few registers are used as internal memory.

4.4 ACCESS METHOD

A basic characteristic of a memory is the order or sequence in which information can be accessed. The methods of accessing include the following:

- Sequential or Serial access
- Random access
- Direct or Semi-random access
- Associative

Sequential Access In this method, the memory is accessed in a specific linear sequential manner. For example, if fourth record (collection of data) stored in a sequential access memory needs to be accessed, the first three records must be skipped. Thus, the access time in this type of memory depends on the location of the data. Magnetic disks, magnetic tapes and optical memories like CD-ROM use this method.

Random Access In this mode of access, any location of the memory can be accessed randomly. In other words, the access to any location is not related with its physical location and is independent of other locations. For random access, a separate mechanism is there for each location. Semiconductor memories (RAM, ROM) are this type.

Direct Access This method is basically the combination of previous two methods. Memory devices such as magnetic hard disks contain many rotating storage tracks. If each track has its own read/write head, the tracks can be accessed randomly, but access within each track is sequential. In this case the access is semi-random or direct. The access time depends on both the memory organization and the characteristic of storage technology.

Associative Access This is a special type of random access method that enables one to make a comparison of desired bit locations within a word for a specific match and to do this for all words simultaneously. Thus, based on a portion of a word's content, word is retrieved rather than its address. Cache memory uses this type of access mode.

4.5 MAIN MEMORY

The central storage unit in a computer system is the main memory which is directly accessible by the CPU. It is a relatively large and fairly fast external memory used to store programs and data during the computer operation. Most of the main memory in a general-purpose computer is made up of RAM (Random Access Memory) integrated circuit chips, which are volatile (i.e. if power goes off, the stored information is lost) in nature. But a small part of the main memory is also constructed with ROM (Read Only Memory) chips, which are non-volatile. Originally, RAM was used to refer to a

random-access memory, but now it is used to mean a read-write memory (RWM) to distinguish it from a read-only memory, although ROM's access mechanism is also random.

RAM is used to store the most of the programs and data that are modifiable. Integrated RAM chips are available in two forms: one is static RAM (SRAM) and another is dynamic RAM (DRAM). The SRAM memories consist of circuits capable of retaining the stored information as long as power is applied. That means this type of memory requires constant power. SRAM memories are used to build cache memory. On the other hand, DRAM stores the binary information in the form of electric charges that applied to capacitors. The stored information on the capacitors tend to loss over a period of time and thus the capacitors must be periodically recharged to retain their state. The main memory is generally made up of DRAM chips.

Comparison of SRAM and DRAM

1. The SRAM has lower access time, which means it is faster compared to the DRAM.
2. The SRAM requires constant power supply, which means this type of memory consumes more power; whereas, the DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
3. Due to the relatively small internal circuitry in the one-bit memory cell of DRAMs, the large storage capacity in a single DRAM memory chip is available compared to the same physical size SRAM memory chip. In other words, DRAM has high packaging density compared to the SRAM.
4. SRAM is costlier than DRAM.

Another part of the main memory consists with ROMs (read only memories), whose contents are not generally altered by the users/programmers. In other words, the ROM is generally used for storing the programs and data that are permanently resident in the computer. In this connection, it is worth noting that the ROM portion of the main memory is needed for storing an initial start-up program called a *Bootstrap Loader*. The Bootstrap Loader is a system program whose task is to load a portion the operating system from secondary memory (hard-disk) to main memory (RAM).

4.5.1 RAM and ROM Chips

Most part of the main memory is consisted of RAM chips, since RAM chip is used to read and write operations on programs and data. A block diagram of a RAM chip of size 512×8 is shown in Fig. 4.4.

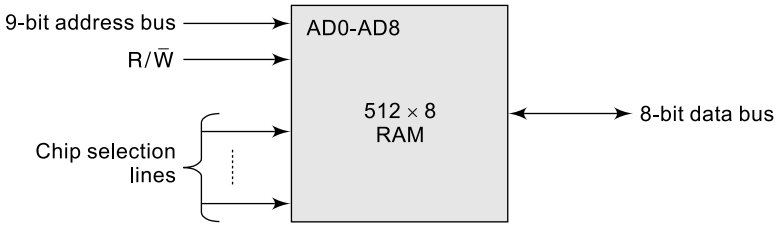


Figure 4.4 *Block diagram of a RAM chip*

The chip has 512 locations each location capable of storing 8 bits. This requires a 9-bit address bus and 8-bit bidirectional data bus. Here, note that the data bus is bidirectional, since it allows the

transfer of data either from memory to CPU during read operation or from CPU to memory during write operation. The R/$\overline{W}$ control line specifies either read or write operation. When this line is high, the control line sends read signal and when it is low, it sends a write signal. Some chip selection lines are required to enable the desired chip from multiple chips in a large memory system before read or write operation on it.

A ROM is used to read the information from it. So, it does not have any R/$\overline{W}$ line, because when a chip is selected, it will be used to read the binary information from it. Also, the data bus is unidirectional. A ROM chip is organized externally in a similar manner as RAM. A block diagram for a ROM chip of size 512 × 8 is shown in Fig. 4.5.

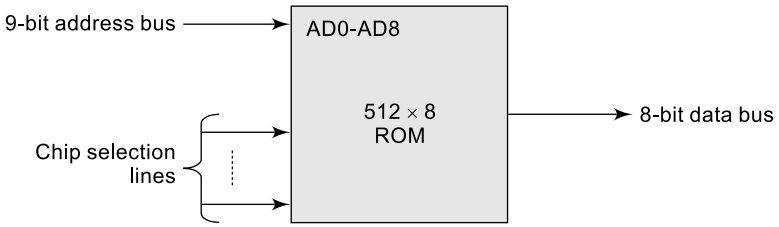


Figure 4.5 *Block diagram of a ROM chip*

For the same physical size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM, which can be easily understood by their internal circuits discussed in Section 4.5.2. In other words, ROM has high packaging density compared to the RAM.

Construction of Large Memory Using Small Chips The large memory can be constructed by expanding some small size chips in either horizontally or vertically. In horizontal expansion, the word is increased; whereas in vertical expansion, number of locations is increased. For example, two RAM chips each of size 512 × 4 can be horizontally expanded to obtain a large memory of size 512 × 8 and the same number of 512 × 4 RAM chips can be connected vertically to construct a large memory of size 1K × 4. Sometimes large memory is constructed using either horizontal or vertical technique or sometimes using both techniques.

Large memory to be constructed can be of heterogeneous (i.e. mixture of both RAM and ROM) or homogeneous (i.e. either all chips are RAM or ROM, but not both). We will discuss it using some examples.

RAM and ROM chips are available in a variety sizes. If the memory needed for a computer is larger than the size of a single chip, then it is necessary to combine a number of smaller chips to form the required memory size.

To illustrate this construction, we will take two examples. First is of heterogeneous and second is homogeneous connection.

Heterogeneous Case

Example 4.1 Suppose, we have two sets of memories with RAM of size 512 × 8 and ROM of size 512 × 8 to design a memory of capacity 1024 × 8.

This is already mentioned that for the same physical size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy

less space than in RAM. Therefore, we will take a ROM of size 512 bytes and the four RAM chips each of 128 bytes. The Fig. 4.6 shows the interconnection diagram of these memories with the CPU having16 address lines.
The address lines 1 to 7 are connected to each memory and address lines 8, 9 are used in dual purposes. In case of a RAM selection out of four RAMs, these two lines are used through a 2-to–4 decoder and the lines are also connected to the ROM as address lines along with lines 1 to 7 giving a total of 9 address lines in the ROM, since the ROM has 512 locations. The CPU address line number 10 is used for separation between RAM and ROM.

Memory Address Map The interconnection between memory and CPU is established from the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be designed by means of a table, known as *memory address map*, which specifies the memory address space assigned to each chip. The address map table for the memory connection to the CPU shown in Fig. 4.6 is constructed in Table 4.1. The CPU generates 16-bit address for memory read or write operation. In case of any RAM operation, only 7-bit address is required. Since there are 4 RAM chips, a 2-to-4 decoder is required for selecting any one RAM at a time. For this 8 and 9 lines are required. Also, 10^{th} line is used for separation of RAM with ROM. The ROM chip has 512 bytes and so it needs 9 address lines. For the ROM, along with lower-order 7 address lines, 8 and 9 lines are used as address lines. The other 11 to 16 lines of CPU are unused and for simplicity we assume that they carry 0s as address signals.

Table 4.1 *Memory address map table for the Fig. 4.6*

Chip selected	*Address space (in HEX)*	*Address bus*									
		10	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
RAM1	0200 – 027F	1	0	0	x	x	x	x	x	x	x
RAM2	0280 – 02FF	1	0	1	x	x	x	x	x	x	x
RAM3	0300 – 037F	1	1	0	x	x	x	x	x	x	x
RAM4	0380 – 03FF	1	1	1	x	x	x	x	x	x	x
ROM	0000 – 01FF	0	x	x	x	x	x	x	x	x	x

Homogeneous Case Suppose the required large RAM memory size is K × L and the small size RAM chip capacity is m × n, then the number of small size chips required can be calculated as:
The number of chips each of size m × n = s = $\lceil (K * L)/(m * n) \rceil$.

Example 4.2 Suppose, we have to construct a large RAM-type memory of size 1K × 8 using same size smaller RAM chips each of size 256 × 2.
The block diagram for 256 × 2 RAM chip is shown in figure 4.7.

The larger 1K × 8 RAM memory requires 10 address lines and 8 data lines. The construction of this memory using smaller RAMs each of 256 × 2 needs 1K/256= 1024/256 = 4 rows and 8/2 = 4 columns of smaller chips, as in shown in figure 4.8. Total number of smaller chips required is s = 4*4 = 16. In the Fig. 4.8, all chips are

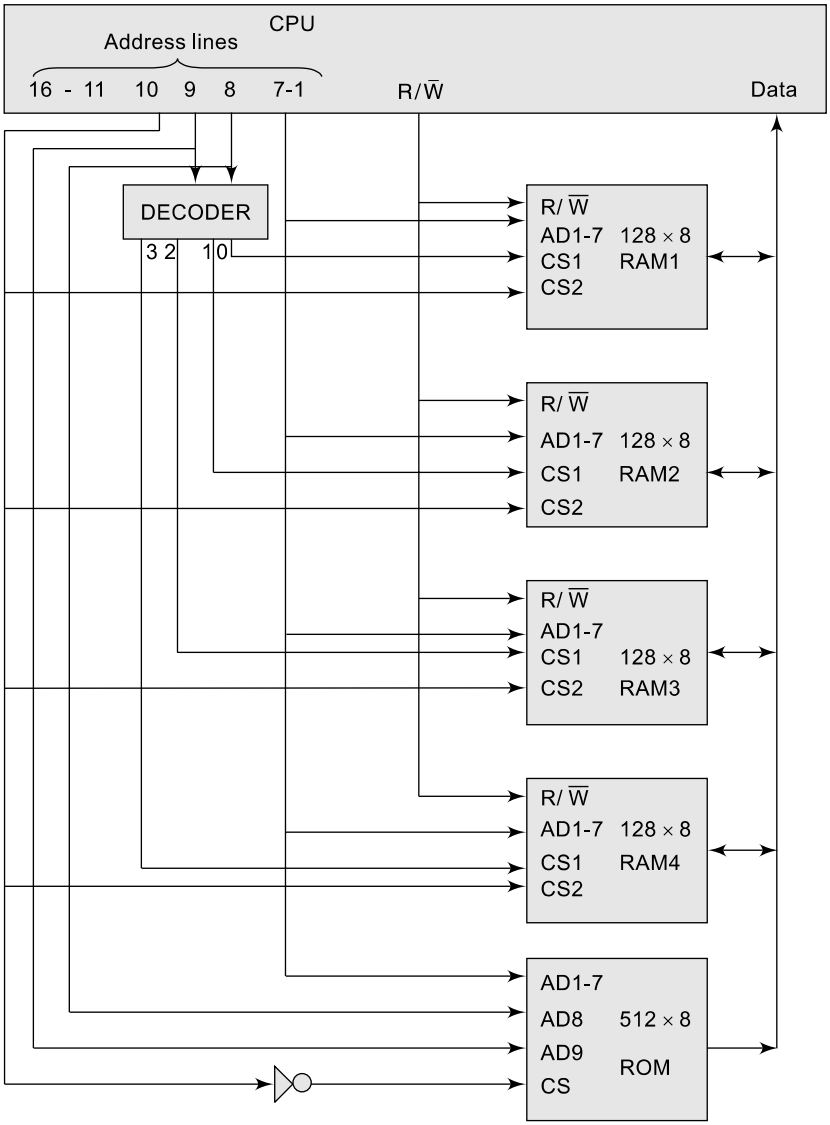


Figure 4.6 *Memory connection with 16-bit CPU*

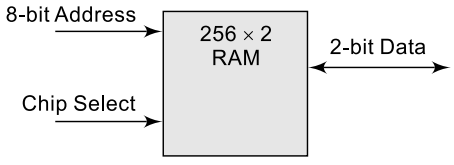


Figure 4.7 *Block diagram for 256 × 2 RAM chip*

of equal size, i.e. 256 × 2. The address lines A_8 and A_9 are connected to a 2-to-4 decoder, which activates (selects) one of four rows of chips at a time. For example, if A_8 A_9 = 00, then decoder selects first row of four chips for read or write operation. Since same 8-bit address is sent to all chips at a time. When one row of four chips is selected, 8 bits of data from a common location in all four chips are accessed simultaneously.

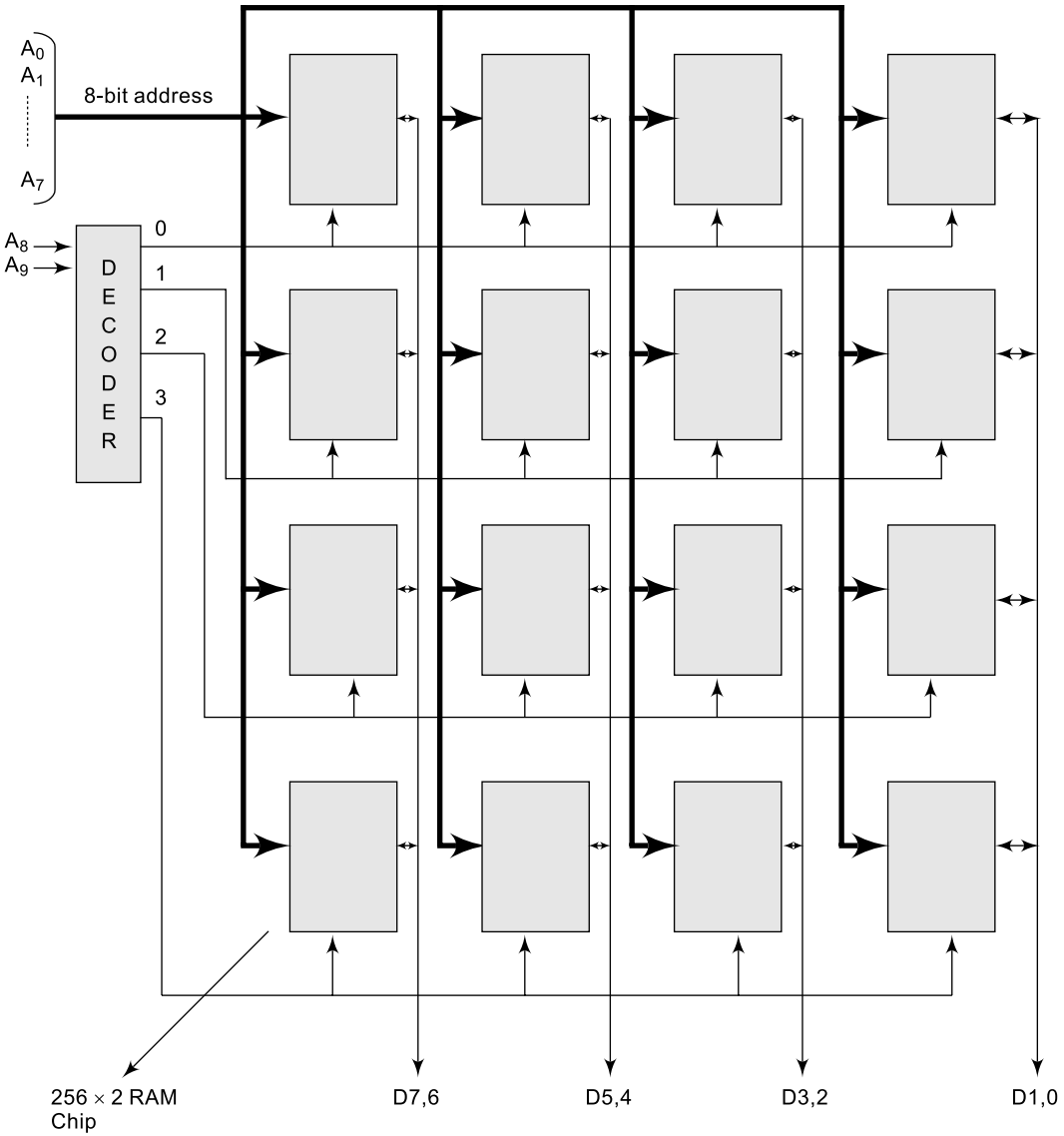


Figure 4.8 *Realization of 1K × 8 RAM using 256 × 2 chips*

Example 4.3 Suppose a large memory of 1K × 4 is to be constructed using 512 × 2 RAM chips.

For small size RAM chip of 512 × 2, number of address lines required is 9 and number of data lines is 2. For large memory of 1K × 4, number of address lines required is 10 and number of data lines is 4.

Therefore, in the interconnection diagram:

The number of rows = 1K/512 = 1024/512 = 2

The number of columns = 4/2 = 2

Hence, the number of small size RAMs required = 2*2 = 4.

The interconnection diagram is shown in Fig. 4.9. Here in the diagram, only two rows are there. So, the first row is selected (activated) by A_9 line of the address bus directly and the second row is selected by its complement bit information. In other words, if A_9 line contains logic 1, then first row of chips will be selected and otherwise the second row will be selected.

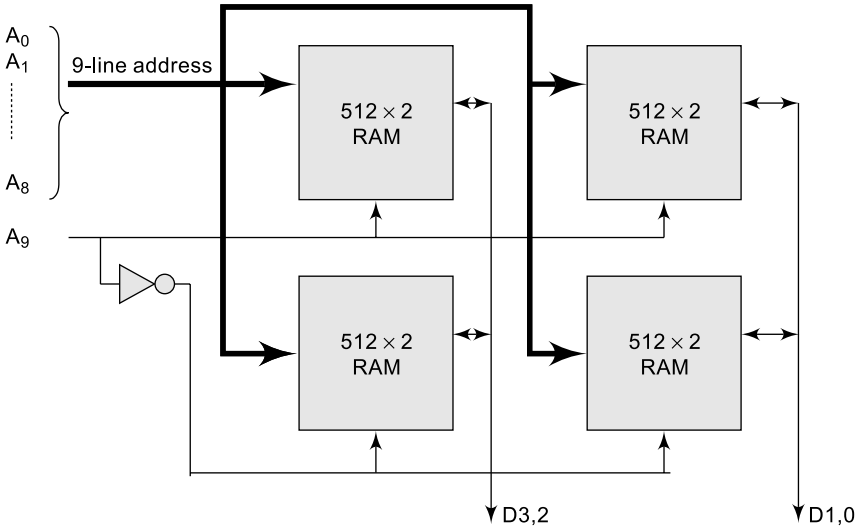


Figure 4.9 *Construction of 1K × 4 memory using 512 × 2 RAM chips*

4.5.2 Internal Organization of Memory Chips

A memory consists of cells in the form of an array, in which each cell is capable of storing one bit of information. Each row of the cells constitutes a memory word and all cells of a row are connected to a common line referred to as a *word line*. Thus, a $w \times b$ memory has w number of words, each word having b number of bits. The main storage array is referred to random access memory in the sense that each memory location (word) has a unique wired-in addressing mechanism. As a result, corresponding to given address of a word the bits of that word can be accessed randomly and the time to access any location is equal to the memory cycle time. Depending on the access mechanism, different types of memory organization have evolved. These are referred to as 2D, 3D and 2.5D, where D stands for dimension.

2D Organization This is the simplest type of organization. An example of size 16×4 memory is shown in Figure 4.10. The cells are organized in the form of a two-dimensional array with rows and columns.

Each row refers to a word line. For a 4-bit per word memory, 4 cells are interconnected to a word line. Each column in the array refers to a bit line. The Memory Address Register (MAR) holds the address of the location where read/write operation is executed. For a $w \times b$ memory, MAR has $\log_2 w = n$ bits. Here in this example, $n = 4$. The content of MAR is decoded by an address decoder on the chip to activate each word line. The cells in each column are connected to a sense/write circuit by two bit lines. Two bit lines are complement to each other. The sense/write circuits are activated by the chip select (CS) lines. The sense/write circuits are connected to the data lines of the chip. During a read operation, these circuits sense or read the information stored in the cells selected by a word line and transmit this information to the data lines. During a write operation, the sense/write circuits receive or write input information from the data lines and store it in the selected cells.

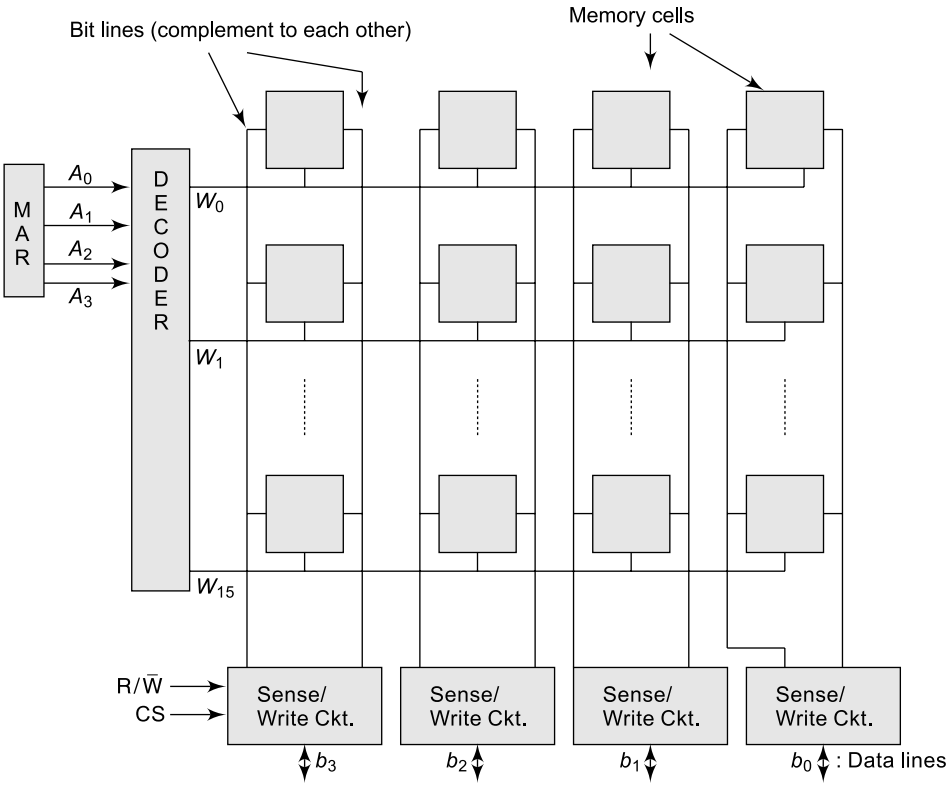


Figure 4.10 *2D organization of a memory chip of size 16×4*

3D Organization For an n-bit MAR in 2D organization the word lines are linearly selected and hence the number of decoder gates is 2^n. By contrast, in 3D organization as shown in Figure 4.11, the number of decoder gates reduces to $2.2^{n/2}$ for $x = y = n/2$. Such saving in the circuit cost has motivated the designers to design 3D organized memory cell array.

The n-bit address is divided into two parts having x and y number of bits. For a square array, each half is decoded and $2^{n/2}$ X and Y drive lines are fed into each array of bit plane. For b-bit word memory, there are b number of planes each referring to a bit. Corresponding to each bit plane there is a sense/write circuit. The read/write operations in 3D organization is same as to 2D with the modifications that a cell in a bit plane is selected by activating X and Y drive lines simultaneously, and bit information passed through the selected cell in a bit plane. Thus, each cell in the array needs 3 terminals—X, Y and bit line connected to sense/write circuit. More the number of terminals (wires) through a cell, larger the cell size and consequently switching speed is less. Also the design of the overall circuit becomes very complex.

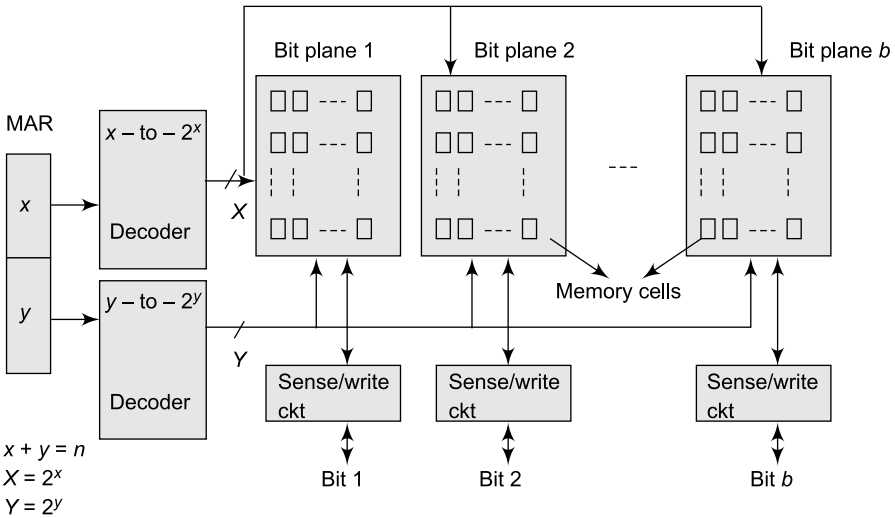


Figure 4.11 *3D memory organization*

2.5D Organization To cope with the above difficulties experienced in 3D organization, the design of 2.5D memory organization has evolved which combines the function of bit lines and Y drive lines. In 2.5D organization there exists a segment, corresponding to bit plane of 3D organization. The content of MAR is divided into two parts—x and y number of bits. The number of segments S is equal to 2^y. X = 2^x drive lines are fed into the cell array and y number of bits decode one bit line out of S lines fed into a segment of the array. In total, there are Sb number of bit lines for a b bit per word memory.

Thus, for any given address in the MAR, the column decoder decodes b out of Sb bit lines by using the y bits of the MAR while a particular word line is activated by using the x bits. Thus only the b number of bits in the array are accessed by enabling the word line and b number of bit lines simultaneously. A general 2.5D memory organization is shown in Figure 4. 12.

Let us consider an example to realize a 256K × 8 (256K word, 8-bit/word) memory with 512 × 512 cell array in a 2.5D organized configuration. The cell array in a chip, as shown in Figure 4.13(a), can be organized with 512 rows and 64 segments with 8 columns per segment. The chip select line is used as decoder enable signal within the chip that realizes 32K × 8-bit memory. Eight such chips can be organized as noted in Figure 4.13(b), to realize the 256K × 8-bit memory. In this configuration, the

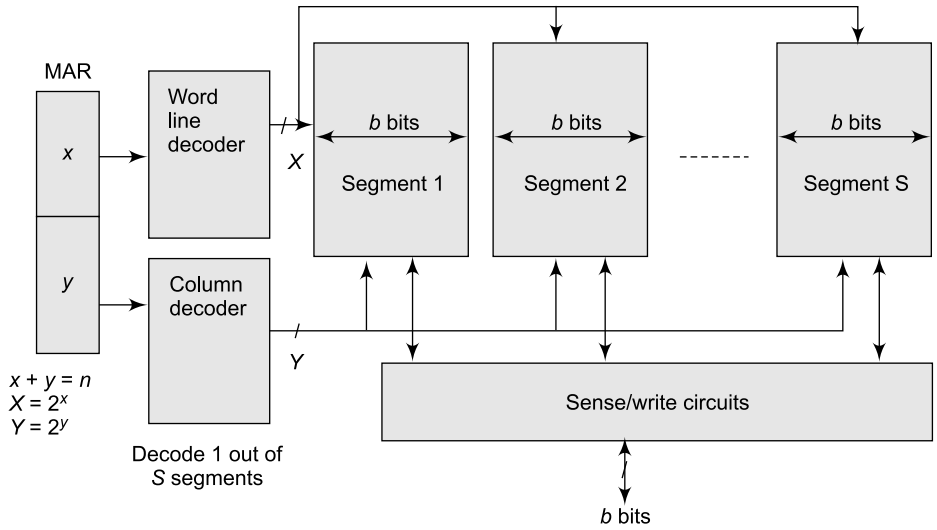


Figure 4.12 *Memory organization*

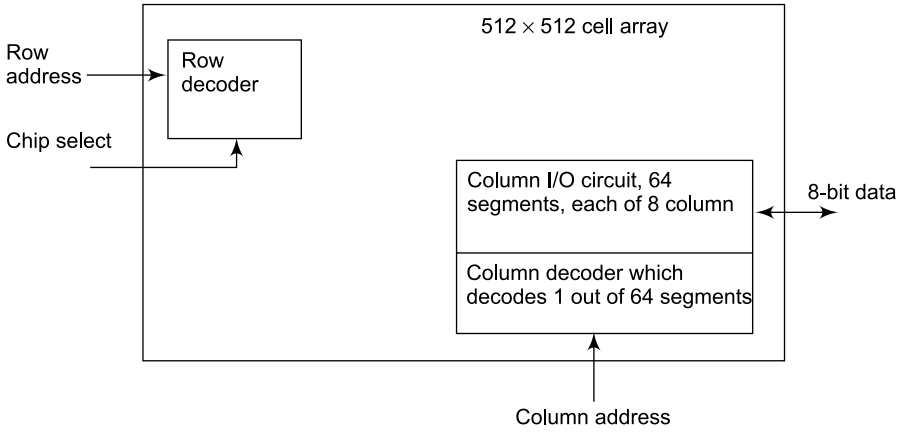
MAR is divided into three parts—the 9-bit field is fed as the row address for each chip while the 6-bit field is input as the column address. The remaining three bits of 18-bit MAR (for 256K of word addresses) are used to select 1 out of 8 chips a noted in Figure 4.13(b).

Though 2.5D organized memory may need lesser chip decoding logic, it suffers from one drawback. With high density chips, a simple failure, such as external pin connection opening or a failure on one bit can render the entire chip inoperative.

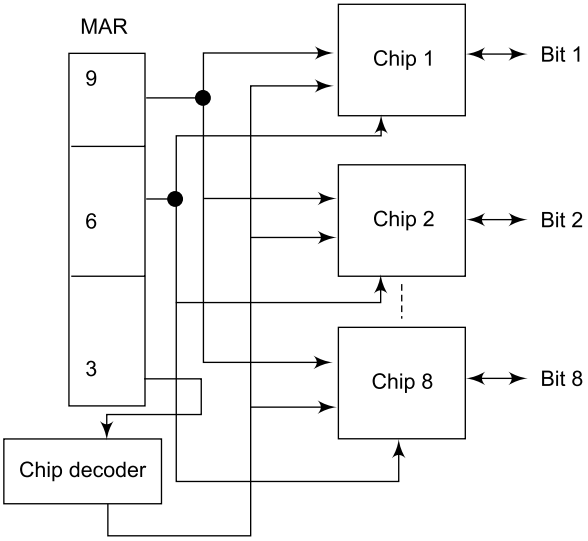
SRAM Memory Cell Static memories (SRAMs) are memories that consist of circuits capable of retaining their state as long as power is applied. Thus, this type of memories is called volatile memories. The Fig. 4.14 shows a cell diagram of SRAM memory. A latch is formed by two inverters connected as shown in the figure. Two transistors T_1 and T_2 are used for connecting the latch with two bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch retains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point B is 0. This state is retained as long as the word line is not activated.

Read Operation For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.



(a) 2.5D organized memory chip built with 512 × 512 cell array



(b) Organization of chip array

Figure 4.13 *256K × 8 memory configured out of 2.5D 512 × 512 memory chips*

CMOS (Complementary Metal Oxide Semiconductor) Realization of SRAM One SRAM cell using CMOS is shown in Fig. 4.15. Four transistors (T_3, T_4, T_5 and T_6) are cross connected in such a way that they produce a stable state. In state 1, the voltage at point A is maintained high and voltage at point at B is low by keeping transistors T_3 and T_6 on (i.e. closed), while T_4 and T_5 off (i.e. open). Similarly, in state 0, the voltage at A is low and at point B is high by keeping transistors T_3 and T_6 off, while T_4 and T_5 on. Both these states are stable as long as the power is applied on it. Thus, for state 1, if T_1 and T_2 are turned on (closed), bit lines b and $\bar{b}$ will have high and low signals, respectively. The state of the cell is read or written as above.

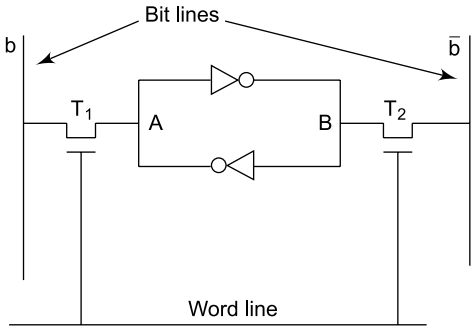


Figure 4.14 *A SRAM cell*

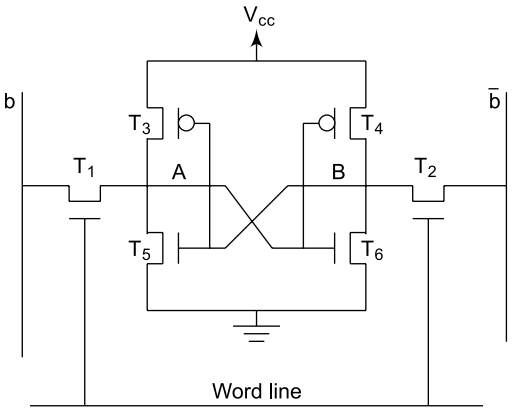


Figure 4.15 *A CMOS SRAM cell*

The main advantage of using CMOS SRAMs is the low power consumption. Since, when the cell is being accessed the current flows in the cell only. Otherwise, T_1, T_2 and one transistor in each inverter are turned off, ensuring that there is no active path between V_{cc} and ground.

DRAM Memory Cell Though SRAM is very fast, but it is expensive because of its each cell requires several transistors. Relatively less expensive RAM is DRAM, due to the use of one transistor and one capacitor in each cell, as shown in the Fig. 4.16, where C is the capacitor and T is the transistor. Information is stored in a DRAM cell in the form of a charge on a capacitor and this charge needs to be periodically recharged.

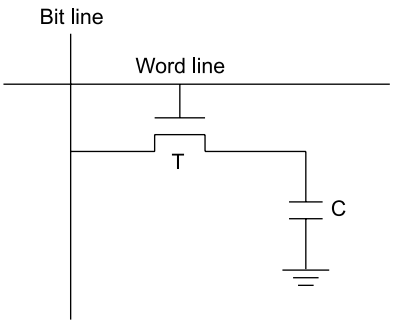


Figure 4.16 *A DRAM cell*

For storing information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitor drops below some threshold value.

Types of RAM

Asynchronous DRAM (ADRAM) The DRAM described above is the asynchronous type DRAM. The timing of the memory device is controlled asynchronously. A specialized memory controller circuit generates the necessary control signals to control the timing. The CPU must take into account the delay in the response of the memory.

Synchronous DRAM (SDRAM) These RAM chips' access speed is directly synchronized with the CPU's clock. For this, the memory chips remain ready for operation when the CPU expects them to be ready. These memories operate at the CPU-memory bus without imposing wait states. SDRAM is commercially available as modules incorporating multiple SDRAM chips and forming the required capacity for the modules.

Double-Data-Rate SDRAM (DDR SDRAM) This faster version of SDRAM performs its operations on the both edges of the clock signal; whereas a standard SDRAM performs its operations on the rising edge of the clock signal. Since they transfer data on both edges of the clock, the data transfer rate is doubled. To access the data at high rate, the memory cells are organized into two groups. Each group is accessed separately.

Rambus DRAM (RDRAM) The RDRAM provides a very high data transfer rate over a narrow CPU-memory bus. It uses various speedup mechanisms, like synchronous memory interface, caching inside the DRAM chips and very fast signal timing. The Rambus data bus width is 8 or 9 bits.

Cache DRAM (CDRAM) This memory is a special type DRAM memory with an on-chip cache memory (SRAM) that acts as a high-speed buffer for the main DRAM.

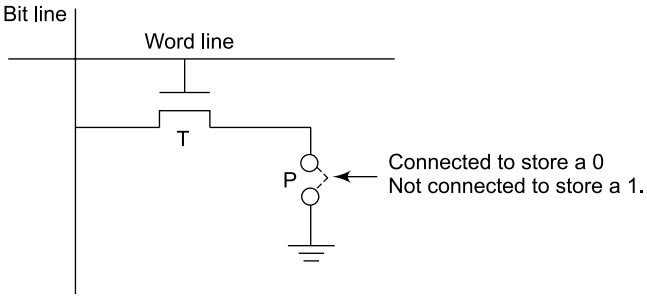


Figure 4.17 *A ROM memory cell*

ROM Memory Cell ROM is another part of main memory, which is used to store some permanent system programs and system data. A ROM cell structure is shown in Fig. 4.17. A logic value 1 is stored in the cell if the transistor is not connected to the ground at point P; otherwise, a binary 0 is stored. The bit line is connected through a resistor to the power supply.

In order to read the state of the cell, the word line is activated to close the transistor, which acts as a switch. The voltage on the bit line drops to near zero if point P is connected. The point P is not connected to retain the state of cell as 1. When it is manufactured, data are written into ROM cells.

Variety of ROM chips is available, which are discussed briefly next.

Types of ROM

PROM Memory Cell Some ROM designs allow the data to be loaded into the cell by user, and then this ROM is called PROM (Programmable ROM). Inserting a fuse at point P in Fig. 4.14 achieves programmability. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses of cells at these locations using high-voltage currents. The PROM's cells are once programmable, i.e. the user can store the desired bits in the cells only once and these bits cannot be altered.

EPROM An erasable PROM (EPROM) uses a transistor in each cell that acts as a programmable switch. The contents of an EPROM can be erased (set to all 1s) by burning out the device to ultraviolet light for a few (20 to 30) minutes. Since ROMs and PROMs are simpler and thus cheaper than EPROMs. The EPROMs are used during system development and debugging.

EEPROM (Electrically Erasable PROM): In many applications, permanent data have to be generated in a program application and need to be stored. For example, in a mobile phone the telephone numbers are to be kept permanently till the user wants to erase those data. Similarly, the user may wish to erase previously entered information. EEPROMs have an advantage in that the information in them can be selectively erased by writing 1s and each bit in the information can be stored again by writing the desired bit. An EEPROM needs two write operations at an address, one for erase and one

for writing. RAM writes the information directly without first erasing that information at that address. But in the EEPROM, the stored information is non-volatile.

Flash Memory A currently popular type of EEPROM, in which erasing is performed in large blocks rather than bit by bit, is known as flash EPROM or flash memory. Erasing in large blocks reduces the overhead circuitry, thus leading to greater density and lower cost. The current trend is "memory stick" made of flash memory that is used to Universal Serial Bus (USB) of the personal computer for data exchange between computers.

4.6 SECONDARY (AUXILIARY) MEMORY

The largest capacity and less expensive memory in the system is secondary memory. The following hierarchy diagram in Fig. 4.18 illustrates some the various devices that are available for secondary (auxiliary) storage of data. The most common secondary memory devices used are magnetic tapes and magnetic disks.

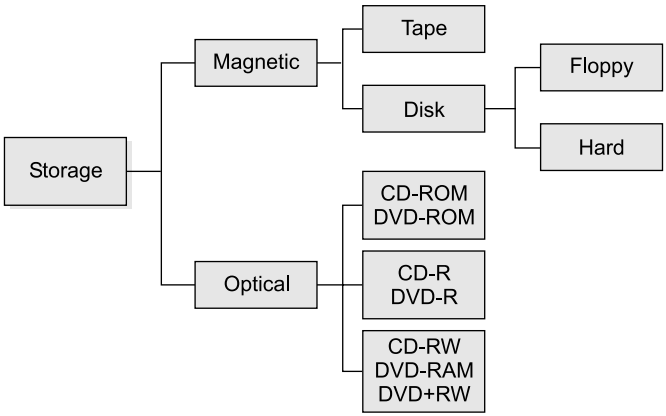


Figure 4.18 *Classification of secondary memory*

4.6.1 Magnetic Tape

Magnetic tapes were the first kind of secondary memory used in computer systems. A tape is flexible polyester coated with special magnetic material. A magnetic tape is similar to a home tape recorder. However, a magnetic tape holds digital information, whereas a tape recorder holds analog information. A magnetic tape is divided vertically into frames and horizontally into nine parallel tracks, as in Fig. 4.19.

Each frame is capable of storing 9 bits of data. The first 8 bits form a data byte and the 9^{th} bit holds the parity. The parity bit is used for error correction and detection. Information is stored along tracks using read-write heads. Read-write heads are designed in such a way that they can access all nine tracks contained in a frame simultaneously. Data is written on the tape by varying the current through the read-write heads. Data is read or written in contiguous records. The records are separated by gaps referred to as inter-record gaps. The length of a magnetic tape is typically 2400 feet and it is stored on a reel. The major difficulty with this device is the particle contamination caused by improper manual handling.

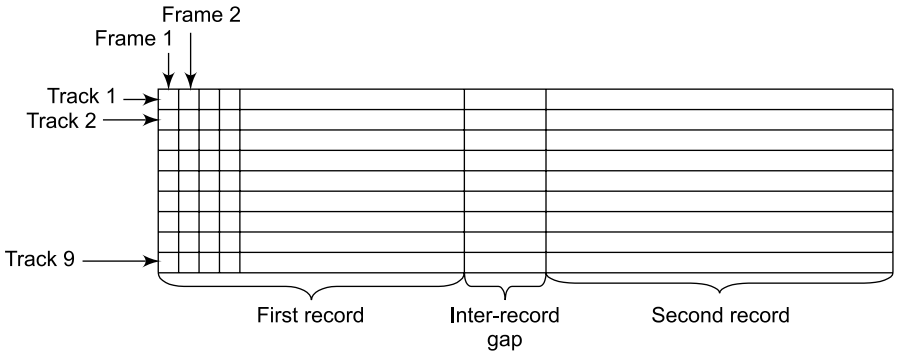


Figure 4.19 *A part of a magnetic tape*

4.6.2 Magnetic Disk

Disks that are permanently attached to the unit assembly and cannot be removed by the general user are called hard disks. A disk drive with removable disks is called a floppy disk drive. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.

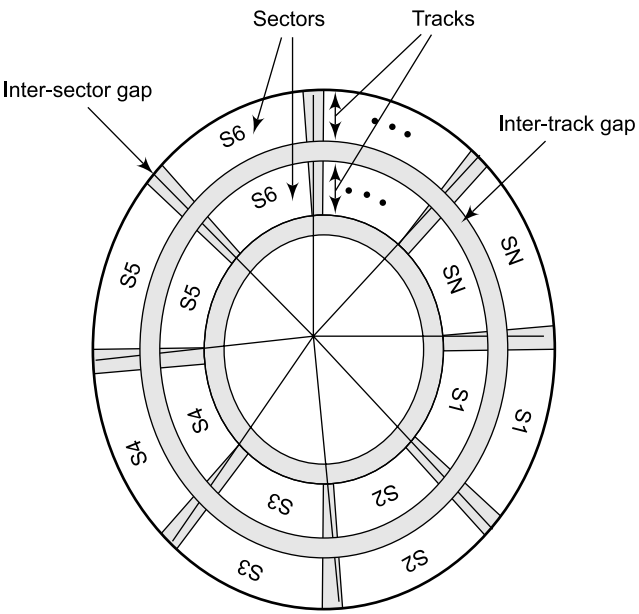


Figure 4.20 *A single disk view*

The magnetic disk is made of either aluminium or plastic coated with a magnetic material so that information can be stored on it. The recording surface is divided into a number of concentric circles called *tracks*. The tracks are commonly divided into sections called *sectors*. To distinguish between two consecutive sectors, there is a small *inter-sector gap*. In most systems, the minimum quantity of information transfer is a sector. Generally, the innermost track has maximum storage density (i.e. bits per linear inch) and outermost track has minimum density. The subdivision of one disk surface into tracks and sectors is shown in Fig. 4.20.

The information is accessed onto the tracks using movable read-write heads that move from the innermost to the outmost tracks and vice-versa. Generally, several identical disks are stacked over one another with some separation between them to form a *disk pack*. A typical disk pack is shown in Fig 4.21. There is one read-write head per surface. Therefore, if there are n disks, there are 2n surfaces. During normal operation, disks are rotated continuously at a constant angular velocity. Same

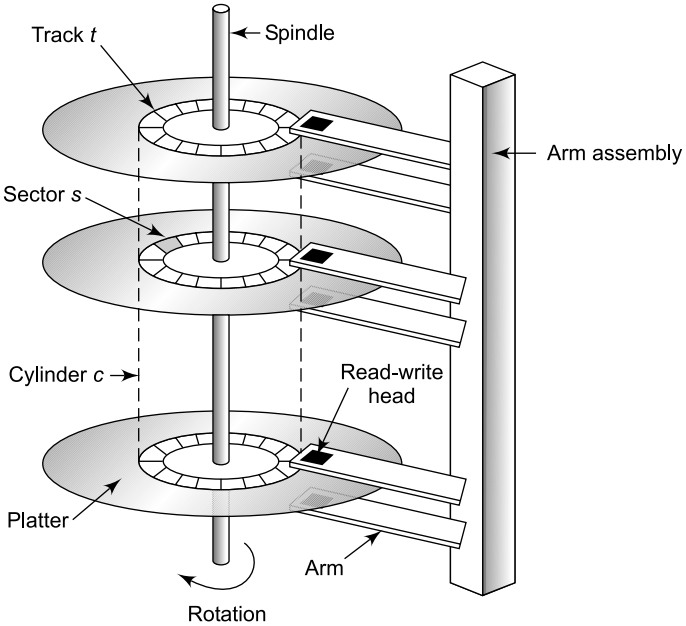


Figure 4.21 *A disk pack*

radius tracks on different surfaces of disks form a logical *cylinder*. A disk pack with n disks has 2n tracks per cylinder. Another part of the disk is the electronic circuitry that controls the operation of the disk, which is called *disk controller*.

To access data, the read-write head must be placed on the proper track based on the given cylinder address. The time required to position the read-write head over the desired track is known as the seek time, t_s. This depends on the initial position of the head relative to the specified track or cylinder address. Seeking the required track is the most time-consuming operation because it involves moving the read-write head arm. After positioning the read-write head on the desired track, the disk controller has to wait until the desired sector is under the read-write head. This waiting time is known as rotational latency, t_l. Rotational latency depends on the rotation speed of the disk. The *access time* of the disk is the sum of t_s and t_l.

There are two ways an n-bit word can be stored:

1. Consecutively store the entire word in one track of the same surface.
2. Store the word in n different tracks of a cylinder.

For the second approach, it is possible to read or write all n bits at the same time, because there is a read-write head for every surface. However, both cases involve the same seek-time and rotational latency overhead.

Problem 4.1 A disk pack has 19 surfaces. Storage area on each surface has an inner diameter of 22 cm and outer diameter of 33 cm. Maximum storage density on any track is 2000 bits/cm and minimum spacing between tracks is 0.25 mm.

(a) What is the storage capacity of the pack?
(b) What is the data transfer rate in bytes per second at a rotational speed of 3600 rpm?

Solution

Given, no. of surfaces = 19
Inner track diameter = 22 cm
Outer track diameter = 33 cm
So, total track width = (33-22)/2 cm = 5.5 cm
Track separation = 0.25 mm
Thus, no. of tracks/surface = (5.5*10)/0.25 = 220
Minimum track circumference = 22 * Π cm
Maximum track storage density = 2000 bits/cm, which will be on innermost track.
So, data storage capacity/track = 22 * Π *2000 bits = 138.23 Kbits
Disk speed = 3600 rpm
So, rotation time = 1/3600 minute = 16.67 msec (1 msec.=10^3 sec)
(a) Storage capacity = 19*220*138.23 Kbits = 577.8 Mbits = 72.225 Mbytes.
(b) Data transfer rate = 138.23 kbits/16.67 msec = 8.2938 Mbits/sec
This is the maximum data transfer rate excluding seek time and rotational latency.

Problem 4.2 A hard disk with one platter rotates at 15,000 rpm and has 1024 tracks, each with 2048 sectors. Disk read-write head starts at track 0. (Tracks are numbered from 0 to 1023) The disk receives a request to access a random sector on a random track. If the seek time of the disk head is 1 ms for every 100 tracks it crosses.
(a) What is the average seek time?
(b) What is the average rotational latency?
(c) What is the transfer time for a sector?

Solution

(a) Since, the disk receives a request to access a track at random. Thus, the head may have to move either direction. On an average, the head will have to move 1024/2 = 511.5 tracks.
Since the seek time of the head is 1ms per 100 tracks, the average seek time = 511.5/100 = 5.115 ms.
(b) Since, the platter rotates at 15,000 rpm, each rotation takes 1/15000 min. = (60 * 10^3)/15000 = 4 ms.
The average rotational latency is half the rotation time, which is = 2 ms.
(c) Each rotation takes 4 ms.
Number of sectors per track = 2048.
Therefore, each sector has read-write head over it = 4/2048 ms = 0.002 ms (apprx.)
Therefore, the transfer time is = 0.002 ms (apprx.)

4.6.3 Optical Media

Although optical media have slower seek times and transfer rates than magnetic media, they do have the greatest storage capacities. Although multimedia has been available with computers, it was not practical until the development of large volume portable storage devices. Depending on the type of optical media, typical disc capacities are 650–680 MB and 4.7–17 GB.

CD CD represents Compact Disc (note: when referring to optical media, disc ends in "c" not "k" as with magnetic disks). CD-ROM means Compact Disk — Read Only Memory, and it is the same media as that used in a home or car CD player. Because the same formats are used for music and data, audio CDs can be played on a computer sound system that has the proper hardware and software. Although data CDs can be read by a computer, they cannot be played in a stereo system, and if they could, what would they sound like? Just a few years ago, software applications were installed from several floppy disks, but now they are usually installed from a single CD-ROM. CDs were the optical discs referred to earlier with a storage capacity of 650–680 MB, which is the equivalent of about 470 standard 3.5" floppy disks. Yet some applications are so large that they may require more than one disk. For example, it requires four discs to install all the components of the Premium version of Office2000. A photographic top view of a CD is shown in Fig. 4.22.

Figure 4.22 *Top view of a CD*

Unlike magnetic disks that place data in concentric circles *(tracks)*, optical discs mimic the layout of a phonograph record and place data in a single spiral *(track)*. However, the spiral track on a CD is only on one side and spirals from inside out. Digital data—binary 0s and 1s—are represented by pits, which scatter a low power laser beam, and lands (flat spots), which reflect it. This is illustrated in the Fig. 4.23 that depicts the side view of a CD.

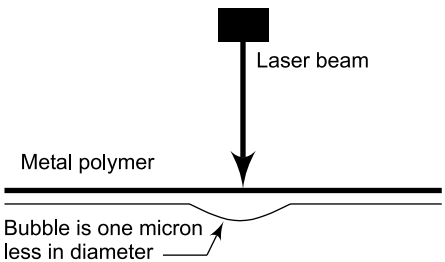


Figure 4.23 *Side view of a very small portion of CD*

By holding one of the CDs up to the light, the pits can be seen because they are less than a millionth of a meter in diameter.

CD-R CD-R means Compact Disc–Recordable. With the proper hardware (a burner), blank CD-R discs, and appropriate software, we can create our own data or audio CDs. Unlike CD-ROM disks where the pits are pressed into the surface in a process similar to that used to make phonograph records, CD-R discs contain a dye layer composed of photosensitive organic compounds. By using a higher energy setting, the dye is heated by the writing laser and becomes opaque through a chemical reaction. So the sections of the disk that have not been burned act as lands, while the opaque sections act as the non-reflective pits. In addition to being able to create new CDs, CD-R drives can also read CD-ROMs and play audio CDs. Furthermore, newly created audio CDs can also be played in a home or car sound system. Unfortunately, CD-R discs can only be recorded once. That is, the same tracks cannot be erased and then rerecorded over. Some of the newer burners are multi-session; this allows users to keep adding data to a CD-ROM over time. This is important if we want to use the CD-R drive to create backup CD-ROMs instead of the traditional tape backups. Are CDs reusable? In other words, can we erase and rerecord them? The answer is yes and it is nothing but the CD-RW.

CD-RW The final type of CDs is CD-RW, which means Compact Disc—ReWriteable. With the proper hardware (also called a burner), blank CD-RW discs, and appropriate software, we can create our own data or audio CDs. CD-RW technology uses a different kind of data-bearing layer from that used in an ordinary CD-R. This technology uses a phase change process to alter its state from a reflective state to a light absorbing state, and it can be reversed to make the area erasable and

reusable. When attempting to copy an audio CD to a blank CD-RW disc, the warning message as shown in Fig 4.24 appears.

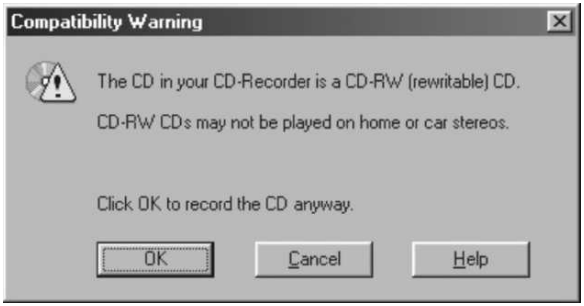


Figure 4.24 *A warning message for using CD-RW to copy audio CD*

CD-RW discs will not play in a home or car sound system, and, furthermore, older CD-ROM players cannot read them.

DVD The newest type of optical storage and the one with the greatest storage capacity is DVD, which means Digital Versatile Disc (originally digital *video* disc). CD players use an infrared laser to read the information stored on the disc, whereas DVD players use red lasers, which allow the bits to be packed closer together. DVDs also have a more efficient encoding, which helps to increase the capacity of the discs. Unlike CDs, DVDs can be dual layered and double sided. DVDs have a storage capacity of 4.7-17 GB, which is the equivalent of up to 15 CDs. Furthermore, DVD players can read CDs, and it is not surprising that they are the standard, or at least the optional, disc reader in many newer computer systems.

With such an enormous storage capacity, what are some practical uses for DVD-ROMs? As indicated by their early name, the most popular one is for the playback of movies. Many videotape rental stores are now including an increasing number of movies titles on DVD, which can be played in a computer DVD player or in one attached to a home entertainment system. Because of the high storage capacity, not only can a full-length movie be recorded on a single disc, but so can multiple language tracks and subtitles written in different languages. By using an audio DVD, a box set of CDs could be released on a single disc instead of multiple CDs. Recall that the Premier version of Office2000 requires four CDs. A single DVD could be used in place of these. Phonebooks of the entire population of our country are available on approximately 4–6 regional CDs containing all the listed phone numbers. Can computer users create their own DVDs? As we may expect, the answer is yes. Comparable to the record-once CD-R burners are the DVD-R burners, and a combination DVD-R/CD-RW burner can be found in modern-day computers. However, there are two competing and incompatible standards for rewriteable DVDs — DVD+RW and DVD-RAM.

4.7 ASSOCIATIVE MEMORY

Several data-processing applications require the search of data in a block or record stored in memory. The normal procedure of searching a block is to store all data where they can be addressed in

sequence. The normal search method is selecting addresses in sequence, reading the content of memory at each address and comparing the information read with the data being searched until a match occurs.

The searching time for desired data stored in memory can be reduced largely if stored data can be searched only by the data value itself rather than by an address. The memory accessed by the data content is known as associative memory or content addressable memory (CAM). When a data is stored in this memory, no address is stored. At any first empty location, the data is stored. When a data word is to be read from the memory, only the data word or part of data, called *key*, is provided. The memory is sequentially searched thoroughly for match with the specified key and set them for reading next.

The *advantage* of using this memory is that it is suitable for parallel searches due to its organization. The searching in this memory is fast. Since each cell must have storage capability as well as logic circuits for matching, the associative memory is more expensive than a RAM memory. For this reason, this memory is used in applications, where search time is very critical and must be very short. An associative memory organization is shown in Fig. 4.25.

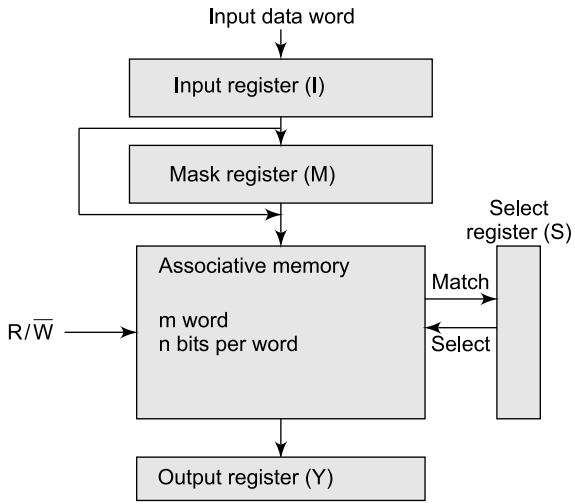


Figure 4.25 *Block diagram of associative memory*

Hardware Structure The associative memory consists of a memory array and logic for m words with n bits per word. In this organization, several registers are used, functions of which are described next.

1. Input register (I) The input register I is used to hold the data to be written into the associative memory or the data to be searched for. At a time it holds one word of data of the memory, i.e. its length is n-bit.

2. Mask register (M) The mask register M provides a mask for choosing a particular field or key in the input register's word. The maximum length of this register is n-bit, because the M register can hold a portion of the word or all bits of the word to be searched. Suppose, a student database file containing several fields such as name, class roll number, address, etc. is stored in the memory. From this file, say only 'name' field is required for searching. Then the M register will hold this 'name' field only and the searching in the memory will be only with respect to this field without bothering about other fields. Thus, only those bits in the input register that have 1s in their corresponding position of the mask register are compared. The entire argument is compared with each memory word if the mask register contains all 1s. Each word in memory is matched with the input data in the I-register.

To illustrate the matching technique, let the input register I and mask register M have the following information:

I = 1001 1011
M = 1111 0000
Word1 = 0011 1100 no match
Word2 = 1001 0011 match

Only four leftmost bits of I are compared with stored memory words because M has 1s in these positions. There is a match for word2, but not with word1.

3. Select register (S) The select register S has m bits, one for each memory word. If matches found after comparing input data in I register with key field in M register, then the corresponding bits in select register (S) are set.

4. Output register (Y) This register contains the match data word retrieved from the associative memory.

The relation between the memory array and four external registers in an associative memory system is shown in Fig. 4.26.

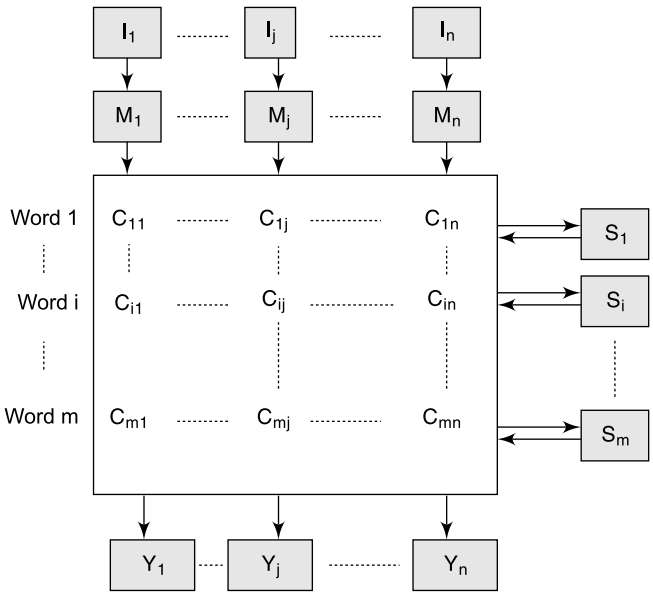


Figure 4.26 *Associative memory of size m × n*

The internal organization of a cell C_{ij} is shown in Fig. 4.27. It consists of a flip-flop storage element A_{ij} and the circuits for matching, selecting, reading and writing the cell. By a write operation, the input bit I_j is stored into the cell. By a read operation, the stored bit is read from the cell. The match and select logic compares the content of the storage cell with the corresponding unmasked bit of the input register and provides an output for the decision logic that sets the bit in S_i.

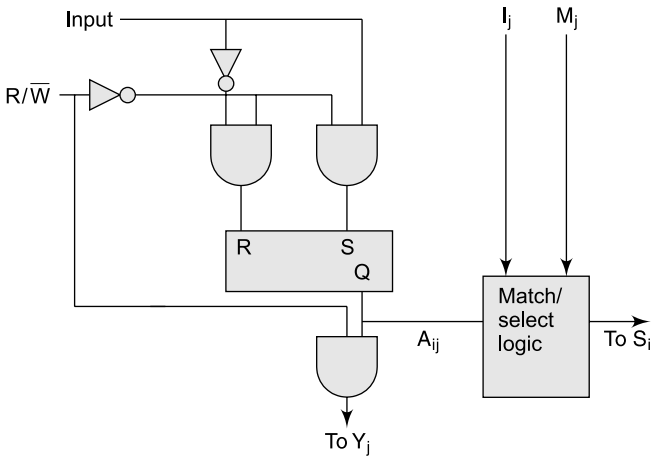


Figure 4.27 *One cell of associative memory*

4.8 CACHE MEMORY

Cache memory is small and fast memory used to increase the instruction-processing rate. Its operation is based on the property called "locality of reference" inherent in programs. Analysis of a large number of typical programs shows that the CPU references to main memory during some time period tend to be confined within a few localized areas in memory. In other words, few instructions in the localized areas in memory are executed repeatedly for some time duration and other instructions are accessed infrequently. This phenomenon is known as the property of locality of reference. This property may be understood considering that when a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitutes the loop. Thus loop tends to localize the references to memory for fetching the instructions.

There are two dimensions of the locality of reference property: *temporal and spatial.*

Temporal Locality Recently referenced instructions are likely to be referenced again in the near future. This is often caused by special program constructs such as iterative loops or subroutines. Once a loop is entered or a subroutine is called, a small code segment will be referenced repeatedly many times. Thus temporal locality tends to cluster the access in the recently used areas.

Spatial Locality This refers to the tendency of a program to access instructions whose addresses are near one another. For example, operations on tables or arrays involve accesses of a certain clustered area in the address space. Program segments, such as routines and macros, tend to be stored in the same neighbourhood of the memory space.

If active segments of a program are placed in a fast small cache memory, the average memory access time can be reduced, thus reducing the total execution time of the program. This memory is logically placed between the CPU and main memory as shown in Fig. 4.3. Because we know that the cache memory's speed is almost same as that of CPU. The main idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will be almost same as access time of cache.

The operation of the cache is conceptually very easy and is as follows: First the cache memory is accessed, when the CPU needs to access memory for a word. If the word is found in the cache, the CPU reads it from the fast cache memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed next to find the word. Due to the property of locality of reference, a *block* of words containing the one just accessed is then brought into the cache memory from main memory. The block size may vary from machine to machine. Another term often used to refer to a cache block is *cache line.*

4.8.1 Performance of Cache Memory

The performance of the cache memory is measured in terms of a quantity called hit ratio. When the CPU refers to memory and finds the word in cache, it is said that a hit occurred. If the word is not found in cache, then the CPU refers to the main memory for the desired word and it is referred to as a miss to cache. The hit ratio (h) is defined below:

$$\text{Hit ratio (h)} = \frac{\text{number of hits}}{\text{total CPU references to memory}}$$

$$= \frac{\text{number of hits}}{\text{number of hits + number of misses}}$$

Thus, the hit ratio is nothing but a probability of getting hits out of some number of memory references made by CPU. So its range is $0 \leq h \leq 1$.

Now we will observe the *average access time* of a memory system consisting of two levels of memories: main memory and cache memory.

Let, t_c, h and t_m denote the cache access time, hit ratio in cache and the main memory access time, respectively. Then the average access time can be formulated as:

$$t_{av} = h^* t_c + (1- h) *(t_c+ t_m) = t_c + (1 - h) * t_m \quad ...(4.1)$$

This equation is derived from the fact that when there is a cache hit, the main memory is not be accessed and in the case of cache miss, both main memory and cache memory are accessed.

4.8.2 Cache Mapping

The main characteristic of cache memory is its fast access time. Therefore, the waiting time for the CPU is very small or nil when searching for words in the cache. The transfer of data as a block from main memory to cache memory is referred to as a mapping process. Three types of cache mapping have been used.

1. Associative mapping.
2. Direct mapping.
3. Set-associative mapping.

To discuss these three mapping procedures we will use a specific example of memory organization as shown in Fig. 4.28.

The cache can store 256 words (each of 8 bits) out of 64K words in main memory at any given time. There is a duplicate copy in main memory for each word stored in cache. The CPU communicates with both memories. The CPU first sends a 16-bit (because $64K = 2^{16}$) address to cache memory. If there is a hit, the CPU accepts the 8-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

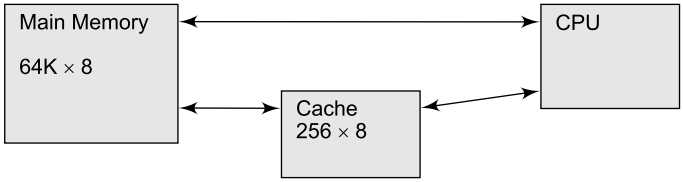


Figure 4.28 *Example of cache memory*

Here, in all three methods, we will be using hexadecimal (HEX) numbers for both address and data for simplicity. Then address of 16 bits is shown in four-digit HEX number and similarly 8 bits data is shown in 2-digit HEX number.

Associative Mapping The associative cache memory uses the fastest and most flexible mapping method, in which both address and data of the memory word are stored. This organization is sometimes referred to as a *fully associative cache* and is illustrated in Fig. 4.29. The cache memory can store 256 words out of 64K words from main memory. This method allows any location in cache to store any word from main memory. The CPU first sends a 16-bit address for a desired word to the input register and the associative cache memory is then searched for a matching the address sequentially. If the address of the desired word is found, then the 8-bit data word is read and sent to the CPU. If no match occurs, then main memory is searched for the address of the word. Then address-data pair is brought into the cache memory from main memory and placed onto an old address-data pair, if the cache is full. In this situation, a replacement algorithm is needed for selecting an old address-data pair to make place for the newly coming address-data pair. For this, different type replacement algorithms such as First-in First-out (FIFO) or Least Recently Used (LRU) are used, which have been discussed later in this section.

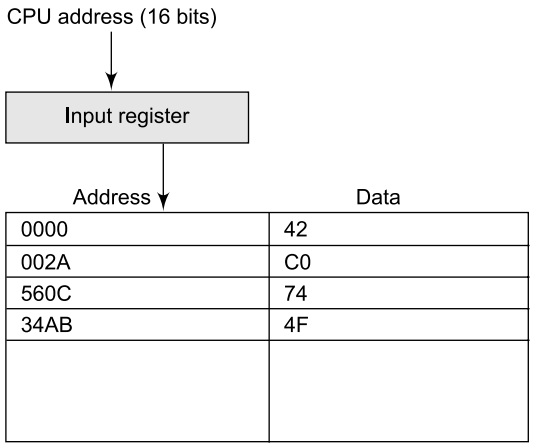


Figure 4.29 *Associative mapping cache (all numbers in HEX system)*

Merits of Associative Mapping This memory is easy to implement and it is also very fast.

Demerits of Associative Mapping This memory is expensive compared to random-access memories because of additional storage of addresses with data in the cache memory. Here, in our example, we are storing 16 + 8 = 24 bits for a single word of 8-bit.

Direct Mapping Instead of storing total address information with data in cache, only part of address bits is stored along with data in the direct cache mapping. Let us assume that cache memory can hold 2^m words and main memory can hold 2^n words. This means that the CPU will generate n-bit memory address. This n-bit address is divided into two fields: lower-order m bits for the index field and the remaining higher-order (n-m) bits for the tag field. The direct mapping cache organization uses the n-bit address to access the main memory and the m-bit index to access the cache. So, for our example, the index and tag fields are shown as follows:

8 bits	8 bits
Tag	Index

The internal organization of the words in the cache memory is shown in Fig. 4.30.

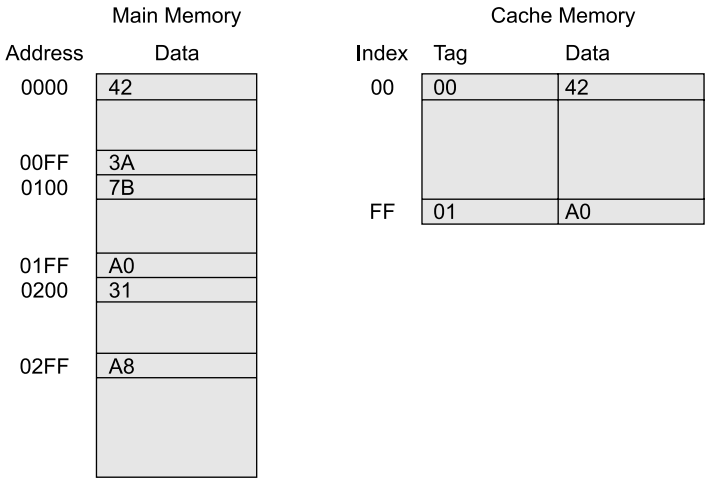


Figure 4.30 *Direct mapping cache organization*

The cache memory stores maximum 256 data words and their associated tags. Due to a miss in the cache, a new word is brought into the cache; the tag bits are stored alongside the data bits. The index field is used as the address to access the cache, when the CPU generates a word's address. The tag field of the CPU address is matched sequentially with all tag values stored in cache. If the CPU's tag-address matches with any cache tag, i.e. there is a hit and the desired data word is read by the CPU from cache. If there is no match in cache, then there is a miss and the required word is read from main memory, which is then stored in the cache together with its new tag, replacing the previous tag-data pair value. Thus, the new tag-data pair is placed in same indexed location in cache as CPU's current index of the address for which miss has occurred. But, here it can be noted that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

The cache is divided into cache *blocks,* also called cache *lines.* A block contains a set of contiguous address words of same size. Each block is typically 32 bytes. We know that data is transferred from main memory to cache memory as block. The direct mapping example just described above uses a block size of one word. The direct mapping cache organization using block size 8 words is shown in Fig. 4.31.

The index field is now divided into two parts: the block field and the word field. Since each block size is 8 words, the 256-word cache can hold 32 blocks. The block number is specified with a 5-bit field, since there are 32 blocks and a word within a block is specified by 3 bits, since each block contains 8 words. The tag field for all stored words within a block is same, since a block contains consecutive 8 words of data. When a miss occurs in cache, an entire block must be brought into cache memory from main memory.

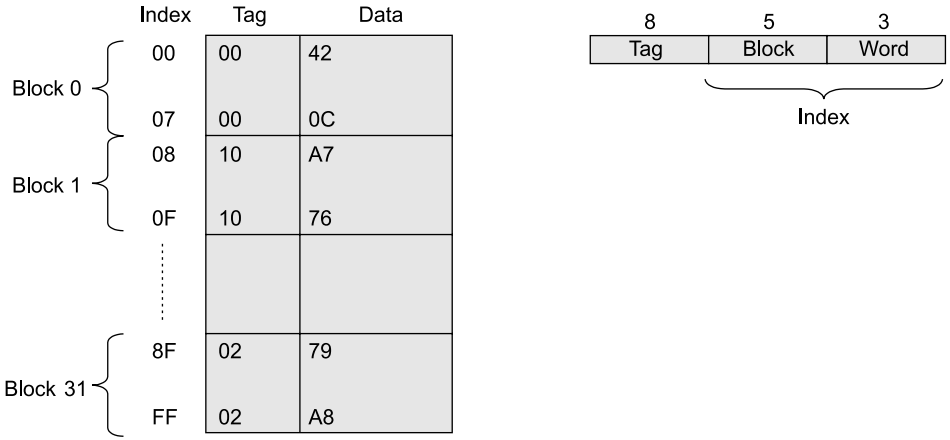


Figure 4.31 *Direct mapping cache with block size of 8 words*

Merits of direct mapped cache

(a) This is the simplest type of cache mapping, since only tag field is required to match. That's why it is one of the fastest cache.

(b) Also, it is less expensive cache relative to the associative cache. Because instead of storing all 16 bits of the address, only tag value of 8 bits is stored along with the data word.

Demerits of direct mapped cache The hit ratio is not good. It needs frequent replacement for data-tag value. Because there is a fixed cache location for any given block. For example, if a program happens to reference words repeatedly from two different blocks that map into the same cache block, then the blocks need to be swapped in continuously making hit ratio to drop.

Set-Associative Mapping The direct mapped cache can hold maximum 256 words, according to our example. The set-associative cache is an improved version of direct mapped cache organization, where multiple of 256 words can be stored, but with increased cost. In set-associative cache memory, two or more words can be stored under the same index address. Each data word is stored together with its tag. The number of tag-data words under an index is said to form a set. If k number of words with their associated tags (i.e. set size = k) are stored under an index of cache, then the cache memory is called *k-way set-associative*. A 2-way set-associative memory is shown in Fig. 4.32, where two data words together with two tag values are stored in a single index address.

For 2-way set-associative cache, the word length is $2(8 + 8) = 32$ bits, since each tag requires 8 bits and each data word requires 8 bits. So, the cache memory size is now converted to 256×32. This means that the cache memory can hold 512 words of the main memory.

When the CPU generates a memory address of 16-bit, the index value of 8-bit is used to access the cache. Then the tag field of CPU address is compared with both tags under the selected index of the 2-way set associative cache memory for hits. The comparison of tags in the set of cache memory is done using the associative search technique, which is why the mapping technique is called set-associative mapping. In this case, since multiple number of words is stored under a common index value, the hit ratio improves compared to previous two techniques.

If the set is full and a miss occurs in a set-associative cache, then one of the stored tag-data pairs must be replaced with a new pair value from the main memory. Some important replacement algorithms are discussed next.

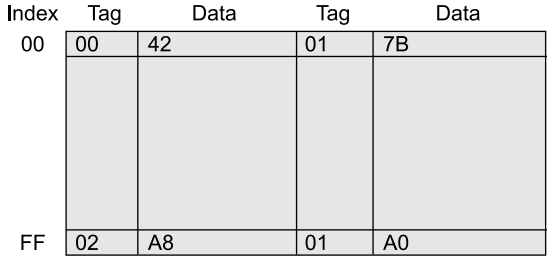


Figure 4.32 *2-way set-associative cache memory*

Merits of Set-Associative Cache This cache memory has highest hit ratio compared to other two cache memories.

Demerits of Set-Associative Cache This is the most expensive memory. The cost increases as set size increases.

Replacement Methods In case a miss occurs in cache memory, then a new data from main memory needs to be placed over old data in the selected location of cache memory. In case of direct mapping cache, we have no choice and thus no replacement algorithm is required. The new data has to be stored only in a specified cache location as per the mapping rule for the direct mapping cache. For associative mapping and set-associative mapping, we need a replacement algorithm since we have multiple choices for locations. We outline below some most used replacement algorithms.

First-in first-out (FIFO) algorithm This algorithm chooses the word that has been in the cache for a long time. In other words, the word which entered the cache first, gets pushed out first.

Least recently used (LRU) algorithm This algorithm chooses the item for replacement that has been used by the CPU minimum number of times in the recent past.

Cache Writing Methods Generally only two operations: read and write are performed on a memory. The read operation does not change the content of memory, since a copy of data has been retrieved from the memory in the course of read operation. However, the write operation changes the content of memory. So, the write operation should be performed carefully. There are two methods in writing into cache memory: Write-through and Write-back policies.

Write-through policy This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus the main memory always contains the same data as the cache. But it is a slow process, since each time main memory needs to be accessed.

Write-back policy In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified or dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set. The philosophy of this method is based on the fact that during a write operation, the word residing in cache may be accessed several times (temporal locality of reference). This method reduces the number of references to main memory. However, this method may encounter the problem of inconsistency due to two different copies of the same data, one in cache and other in main memory.

Cache Types

Caches are distinguished by the kinds of information they store.

Instruction cache vs Data cache: *Instruction or I-cache* stores instructions only, while a *data* or *D-cache* stores data only. Separating the stored data in this way recognizes the different access behavior patterns of instructions and data. For example, programs tend to involve few write accesses, and they often exhibit more temporal and spatial locality than the data they process.

Unified cache vs Split cache: A cache that stores both instructions and data is referred to as *unified cache*. A *split cache,* on the other hand, consists of two associated but largely independent units: an I-cache for instructions and a D-cache for data. While a unified cache is simpler, a split cache makes it possible to access programs and data concurrently. A split cache can also be designed to manage its I- and D-cache components differently.

Caches are also classified by the level they occupy in the memory hierarchy. Early computers employed a single, multichip cache that occupied one level of the hierarchy between the CPU and main memory. Two developments made it desirable to introduce two or more cache levels in high-performance systems: the feasibility of including part of the real memory space on a microprocessor chip and growth in the size (but not the speed) of main memory in typical computers. A level 1 (L1) or primary cache is an efficient way to implement an on-chip memory. An additional memory level can be introduced via an off-chip, level 2 (L2) or secondary cache. The desirability of an L2 cache increases with the size of main memory, assuming that the size of the on-chip, L1 cache is fixed. As main-memory size increases further, even more cache levels may be desirable.

4.8.3 Techniques to Improve the Cache Memory Performance

Referring Eq. (4.1), a better measure of memory hierarchy performance is the average memory access time:

$$\textit{Average memory access time} = \textit{Hit time} + \textit{Miss rate} \times \textit{Miss penalty}$$

where *Hit time* is the time to hit the cache, *Miss rate* is the fraction of accesses that are not in the cache (i.e., (1 - *h*)) and *Miss penalty* is the additional clock cycles to service the miss. In other words, the extra time needed to bring the desired information into the cache from main memory in case of a miss in cache is called the miss penalty.

Hence, in order to improve the cache performance, i.e., to minimize the average access time, different efficient techniques must be implemented in the memory system to reduce the hit time, miss rate and miss penalty.

Techniques to Reduce the Miss Rate

Before going into details of different techniques, we have to gain better insights into the causes of misses. There are three types of misses (they are known as “three C’s”):

Compulsory The very first access to a block cannot be in the cache, so the block must be brought into the cache. These are also called cold start misses.

Capacity If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur because of blocks being discarded and later retrieved.

Conflict If the block placement strategy is set-associative or direct-mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called collision misses.

Compulsory miss rates are usually small. There is little we can do about these misses other than pre-fetching. We can eliminate all conflict misses if we use a fully associative cache. But fully associative caches are expensive in terms of hardware and slower which lengthen the clock cycle, reducing overall performance. Little can be done for capacity misses other than having larger caches but we will find other things we can adjust to improve on capacity misses.

Some techniques to reduce the cache miss rate are described below.

- ***Large Block Size*** The simplest way to reduce miss rate is to increase block size. Large block sizes will reduce compulsory misses. Large blocks may increase conflict misses and even capacity misses if cache is small. So, it is the task of the cache designer to choose the block sizes in such a way that all types of cache miss rates are minimized.
- ***Higher Associativity*** Increased associativity of a set associative cache will reduce the cache miss rate. That means that an 8-way set associative cache will experience less number of cache misses than that of 4-way or 2-way set associative cache. But higher way set associative cache will increase the cost of the memory.
- ***Use of Victim Cache*** To reduce the conflict misses without impairing clock rate, one small fully associative cache called victim cache is placed between a cache and its refill path. This victim cache contains only blocks (victims) that are discarded recently from a cache because of a miss and are checked on a miss to see if they have the desired data before going to the next lower-level memory. If it is found there, the victim block and cache block are swapped.
- ***Large Cache*** A larger cache will reduce capacity miss rates since the cache has a larger capacity. Also it will reduce conflict miss rates because the larger cache allows more refill lines (i.e. more cache blocks).This is an obvious solution and has no seeming performance drawbacks. However, you must be careful where you put this larger cache. A larger on-chip cache might take space away from other hardware that could provide performance increases (registers, more functional units, logic for multiple-issue of instructions, etc.). Also, more cache means a greater expense for the machine.
- ***Compiler Optimizations*** Till now our techniques to reduce misses have required changes to or additions to the hardware: larger blocks, higher associativity, use of victim cache and large cache. This final technique reduces miss rates without any hardware changes. Specific techniques include:
  - ❑ *Merging Arrays* This first technique reduces misses by improving spatial locality. Some programs reference multiple arrays in the same dimension with the same indices at the same time. The danger is that these accesses will interfere with each other, leading to conflict misses. This danger is removed by combining these independent matrices into a single compound array so that a single cache block can contain the desired elements.

```
/* Before */ int val[SIZE]; int key[SIZE];
/* After */ struct merge { int val; int key;
                          };
struct merge merged_array[SIZE];
```

An interesting characteristic of this example is that the proper coding practice of using an array of records would achieve the same benefits as this optimization.

- *Loop Interchange* Some programs have nested loops that access data in memory in non-sequential order. Simply exchanging the nesting of the loops can make the code access the data in the order it is stored. Like the prior example, this technique reduces misses by improving spatial locality. Reordering maximizes use of data in a cache block before it is discarded.

```
/* Before */
for (j =0; j < 100; j = j+1)
    for (i=0; i < 5000; i = i+1)
    x[i][j] = 2 * x[i] [j];
/* After */
for (i = 0; i < 5000; i = i+1)
for (j = 0; j < 100; j = j+1)
    x[i][j] = 2 * x[i] [j] ;
```

The original code would skip through memory in strides of 100 words, while the revised version accesses all the words in the cache block before going to the next one. This optimization improves cache performance without affecting the number of instructions executed, unlike the prior example.

- *Loop Fusion* Some programs have separate sections of code that access the same arrays with the same loops, performing different computations on the common data. By "fusing" the code into a single loop, the data that are fetched into the cache can be used repeatedly before being swapped out. Hence, in contrast to our first two techniques, the target of this optimization is reducing misses via improved temporal locality.

```
/* Before */
for (i=0; i < N; i = i+1)
    for (j=0; j < N; j = j+1)
        a[i] [j] = 1/ b[i][j] * c[i][j];
for (i=0; i < N; i = i+1)
for (j=0; j < N; j = j+1)
    d[i] [j] = a[i] [j] + c[i][j];
/* After */
for (i=0; i < N; i = i+1)
    for (j=0; j < N; j = j+1)
        {
            a[i][j] = 1/b[i][j] * c[i][j];
            d[i][j] = a[i] [j] + c[i][j];
            }
```

The original code would take all the misses to access arrays a and c twice, once in the first loop and then again in the second. In the fused loop, the second statement freeloads on the cache accesses of the first statement.

Techniques to Reduce the Hit Time

- ***Small and Simple Caches*** Associative cache access requires using the address part to find the appropriate line (block) in the cache. The set-associative cache uses index part of the address to find the appropriate block in cache and then comparing tags to see if the entry is the right one. Also these comparisons use more hardware to be done in parallel. It is also critical to keep the cache small so that it fits on the chip. One solution is to keep tags on the chip and data off the chip. This permits a faster comparison followed by accessing the data portion somewhat slower. In the end, this result is not appealing for reducing hit time. Thus a better approach is to use direct-mapped caches.
- ***Avoid Address Translation*** CPU generates an address and sends it to cache. But the address generated is a logical (virtual) address, not the physical address in memory. To obtain the physical address, the virtual address must first be translated. Translation requires accessing information stored in registers, TLB (Translation Look-aside Buffer) or main memory page table, followed by a concatenation. If we store virtual addresses in the cache, we can skip this translation. There are problems with this approach though
  – if a process is switched out of memory then the cache must be flushed.
  – the OS and user may share addresses in two separate virtual address spaces.
  Thus this may cause problems if we use the virtual addresses in the cache.
- ***Pipelining Writes*** Write operations will take longer than read operations because the tag must be checked before the write can begin. A read can commence and if the tag is wrong, the item read can be discarded. The write takes two steps, tag comparison first; followed by the write (a third step might be included in a write-back cache by combining items in a buffer). By pipelining writes, we can partially speed up the process. This works by overlapping the tag checking and writing portions. Although this only works with more than one consecutive write where all writes are cache hits.

Techniques to Reduce the Miss Penalty

- ***Multilevel Caches*** To improve the performance, we find that we would like
  – a faster cache to keep pace with memory
  – a larger cache to lower miss rate
  Both cases will offer a small but fast cache on the CPU chip and a larger but slower cache on the motherboard. The slower cache is still much faster than main memory. This gives us a new formula for average memory access time = Hit time L1 + miss rate L1 * miss penalty L1; where L1 is the first cache (called the first-level cache).
  Miss penalty L1 = hit time L2 + miss rate L2 * miss penalty L2; where L2 is the second cache (called the second-level cache).
  Thus, average memory access time = hit time L1 + miss rate L1 * (hit time L2 + miss rate L2 * miss penalty L2).
- ***Early Restart*** On a cache miss, memory system moves a block into cache. Transferring a full block will require many bus transfers. Rather than having the cache and CPU wait until the

entire block is available, move requested word from the block first to allow cache access as soon as the item is available and transfer rest of block in parallel with that access. This requires two ideas:

- *Early restart* The cache transmits the requested word to the CPU as soon as it arrives from memory.
- *Critical word first* Have memory return the requested word first and the remainder of the block afterward (this is also known as wrapped fetch).

- ***Priority of Reads over Writes*** This is based on the philosophy "Make the more common case fast". Reads occur with a much greater frequency than writes. In programs, instructions are read only and many operands are read but not written back. Writes are slower anyway because of the need to write to both cache and main memory. If we use a write buffer for both types of write policy:
  - Write-through cache writes to write buffer first and any read misses are given priority over writing the write buffer to memory.
  - Write-back cache writes to write buffer and the write buffer is only written to memory when we are assured of no conflict with a read miss.

  Thus, read misses have priority over write misses since read misses are more common, so we make the common case fast.

The techniques discussed so far to improve miss rate, miss penalty and hit time generally affect the other components of the average memory access equation as well as the complexity of the memory hierarchy. Generally a technique helps only one factor. For example, "large cache size" technique can reduce misses, but it hurts the hit time. Thus in summary, no technique helps more than one category.

Problem 4.3 A hierarchical cache-main memory subsystem has the following specifications:

(i) Cache access time of 50 nsec
(ii) main memory access time of 500 nsec
(iii) 80% of memory request are for read
(iv) hit ratio of 0.9 for read access and the write-through scheme is used.

Calculate the following:

(a) average access time of the memory system considering only memory read cycle.
(b) average access time of the system both for read and write requests.

Solution

Given,

Cache access time $t_c = 50$ nsec.

Main memory access time $t_m = 500$ nsec.

Probability of read $p_r = 0.8$

Hit ratio for read access $h_r = 0.9$

Writing scheme: write-through.

(a) Considering only memory read cycle,

The average access time $t_{av\text{-}r} = h_r * t_c + (1–h_r)* (t_c + t_m)$

$= 0.9*50 + (1–0.9)*550$

$= 100$ nsec.

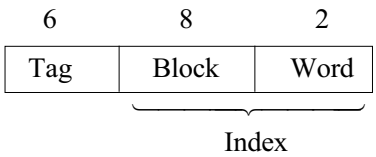
(b) For both read and write cycles,
The average access time = $p_r * t_{av\text{-}r} + (1–p_r)* t_m$ Since in write-through method, access time for write cycle will be the main memory access time.
$= 0.8* 100 + (1–0.8) * 500$
$= 180$ nsec.

Problem 4.4 For a cache memory of size 32 KB, how many cache lines (blocks) does the cache hold for block sizes of 32 or 64 bytes?

Solution
The number of blocks in cache = size of cache/block size.
Thus, for block size of 32 bytes, the number of blocks = (32 * 1024)/32 = 1024.
Similarly, for block size of 64 bytes, the number of blocks = (32 * 1024)/64 = 512.

Problem 4.5 A computer has a main memory of 64K × 16 and a cache memory of 1K words. The cache uses direct mapping with a block size of four words.
(a) How many bits are there in the tag, index, block and word fields of the address format?
(b) How many bits are there in each word of cache?
(c) How many blocks can the cache accommodate?

Solution
(a) The main memory size = 64K × 16
Therefore, the CPU must generate the address of 16-bit (since $64K = 2^{16}$)
The cache memory size = 1K
Therefore, the size of index field of cache = 10-bit ($1K = 2^{10}$)
The tag-field uses 16 – 10 = 6 bits
The size of each cache block = 4 words
Thus, the number of blocks in cache = 1024/4 = 256
Therefore the number of bits required to select each block = 8 (since $256 = 2^8$)
The number of bits required to select a word in a block = 2, because there are 4 words in each block.
Thus, the address format is as follows:



(b) The main memory size = 64K × 16
Therefore, the number of bits in each word in cache = 16
(c) From part (a); the number of blocks in cache = 256.

Problem 4.6 A cache has 64 KB capacity, 128-byte lines and is 4-way set-associative. The CPU generates 32-bit address for accessing data in the memory.
(a) How many lines and sets does the cache have?
(b) How many entries are required in the tag field?
(c) How many bits of tags are required in each entry in the tag array?

Solution

(a) The number of lines in cache = (64 * 1024)/128 = 512.
Since the cache is 4-way set-associative, the number of sets = 512/4 = 128.

(b) Since one tag array entry is required for each line, the tag array needs 512 entries.

(c) Since cache has 128 sets, the number bits required to select a set = 7.
Each line consists of 128 bytes.
Therefore, number of bits required to select a byte (word) = 7
Since the CPU generates 32-bit address to access a byte (word) in memory, the number of bits of tag required in each entry in the tag array = 32 – (7 + 7) = 18.

4.9 VIRTUAL MEMORY

Parts of programs and data are brought into main memory from secondary memory, as the CPU needs them. Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, though that may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual (logical) address is mapped to a physical address in main memory. This mapping is done during run-time and is performed by a hardware device called *memory-management unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

The virtual memory makes the task of programming much easier, because the programmer no longer needs to bother about the amount of physical memory available. For example, a program size is 18 MB and the available user part of the main memory is 15 MB (Other part of the main memory is occupied by the operating system). First, 15 MB of the program is loaded into main memory and then remaining 3 MB is still in the secondary memory. When the remaining 3 MB code is needed for execution, swap out the 3 MB code from main memory to secondary memory and swap in new 3 MB code from secondary memory to main memory.

The advantage of virtual memory is efficient utilization of main memory, because the larger size program is divided into blocks and partially each block is loaded in the main memory whenever it is required. Thus multiple programs can be executed simultaneously. The technique of virtual memory has other advantages of efficient CPU utilization and improved throughput.

Logical (Virtual) Address Space and Physical Address Space When a program needs to be executed, the CPU would generate addresses called *logical* addresses. The corresponding addresses in the physical memory, as occupied by the executing program, are called *physical* addresses. The set of all logical addresses generated by the CPU or program is called *logical-address space* and the set of all physical addresses corresponding to these logical addresses is called *physical-address space*. The memory-management unit (MMU) maps each logical address to a physical address during program execution. Figure 4.33 illustrates this mapping method, which uses a special register called base register or relocation register. The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

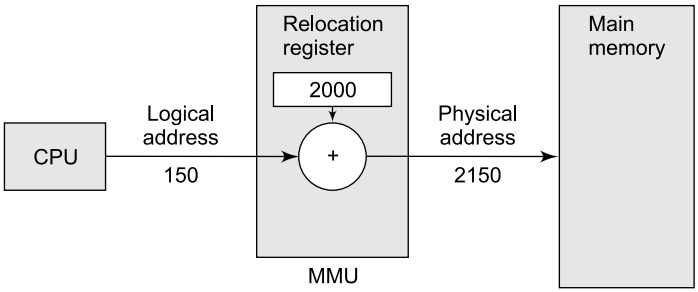


Figure 4.33 *A simple memory-management scheme*

A virtual memory system may be configured in one of the following ways:

1. Paging technique
2. Segmentation technique.

Paging Paging is a non-contiguous memory allocation method. In other words, the program is divided into small blocks in paging and these blocks are loaded elsewhere in main memory. In paging, the virtual address space is divided into equal size blocks called *pages* and the physical (main) memory is divided into equal size blocks called *frames*. The size of a page and size of a frame are equal. The size of a page or a frame is dependent on the operating system and is generally 4 KB.

In paging, operating system maintains a data structure called *page table*, which is used for mapping from logical address to physical address. The page table generally contains two fields, one is page number and other is frame number. The table specifies the information that which page would be mapped to which frame. Each operating system has its own way of maintaining the page tables; most allocate a page table for each program.

Each address generated by the CPU (i.e. virtual address) is divided into two parts: *page number (p)* and *offset or displacement (d)*. The page number p is used as index in the page table and the offset d is the word number within the page p. The structure of paging method is shown in Fig. 4.34.

In order to illustrate the paging, let us consider the following example:

There are two programs of sizes 16 KB and 24 KB in the virtual memory (secondary memory). The available physical (main) memory is 72 KB and size of each page is 4 KB.

For first program of size 16 KB, the number of pages is 16 KB/4 KB = 4 and similarly, for second program of size 24 KB the number of pages is 6. Since the size of physical memory is 72 KB, the number of frames is 72 KB/4 KB = 18. For each program, a page table is maintained. The page tables for programs and their mappings are shown in Fig. 4.35. Page tables are created by the operating system. In this example, total 10 pages (4 for program 1 and 6 for program 2) are loaded in different parts of physical memory. Since the physical memory has 18 frames, therefore remaining 8 free frames can be used for some other programs.

Advantages

1. The paging supports time-sharing system.
2. It utilizes the memory efficiently.
3. It supports non-contiguous memory allocation.
4. It is quite easy to implement.

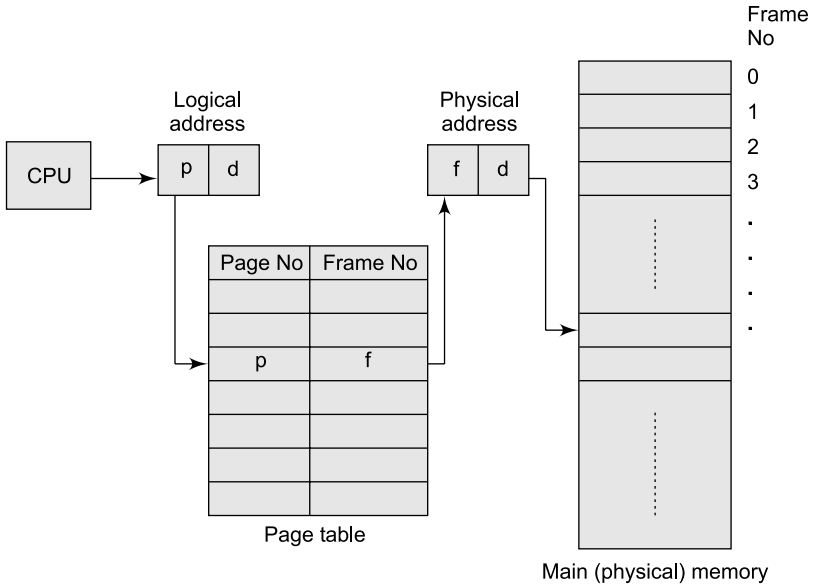


Figure 4.34 *Paging structure*

Disadvantages

1. The paging may encounter a problem called *page break*. For example, the virtual address space for a program is 18 KB and the page size is 4 KB. Thus the number of frames required by this program is 5. However, the used space in last (fifth) frame of the physical memory is only 2 KB and remaining 2 KB of the frame is wasted. This is referred to as *page break*.
2. When the number of pages in a virtual memory is large, it is quite difficult to maintain the page tables.

Page Replacement When a program starts execution, one or more pages are brought to the main memory and the page table is responsible to indicate their positions. When the CPU needs a particular page for execution and that page is not in main (physical) memory (still in the secondary memory), this situation is called *page fault*. When the page fault occurs, the execution of the present program is suspended until the required page is brought into main memory from secondary memory. The required page replaces an existing page in the main memory, when it is brought into main memory. Thus, when a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several replacement algorithms such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement* algorithm available.

The *FIFO algorithm* is simplest and its criterion is "select a page for replacement that has been in the main memory for longest period of time".

The *LRU algorithm* states that "select a page for replacement, if the page has not been used often in the past". The LRU algorithm is difficult to implement, because it requires a counter for each page to keep the information about the usage of page.

The *optimal algorithm* generally gives the lowest page faults of all algorithms and its criterion is "replace a page that will not be used for the longest period of time". This algorithm is also difficult to implement, because it requires future knowledge about page references.

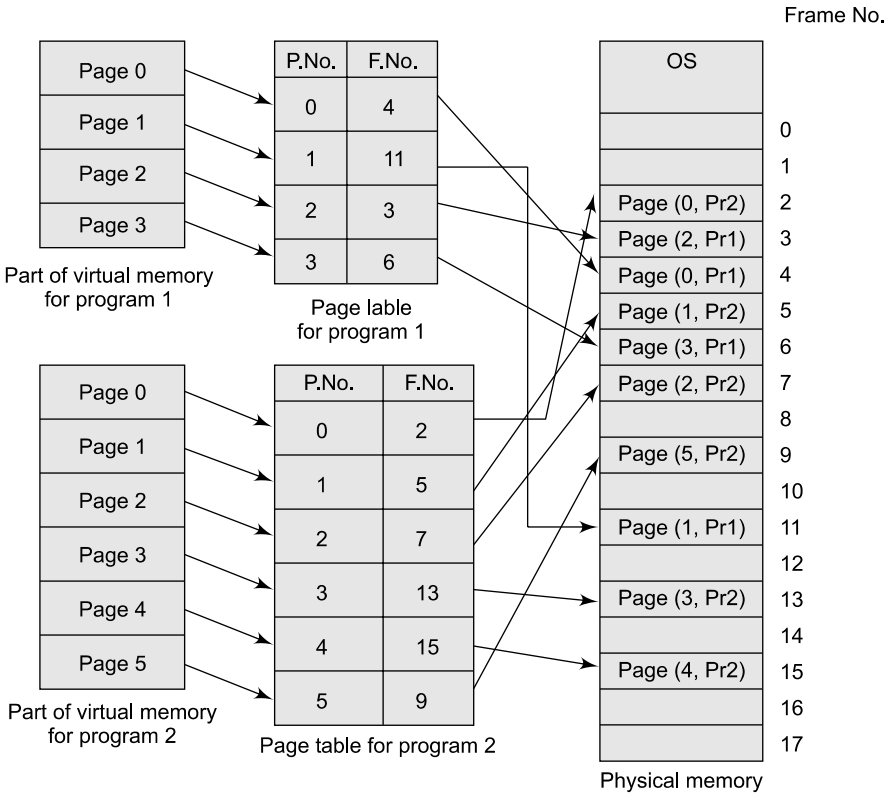


Figure 4.35 *Example of paging*

An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a *reference string*. We can generate reference strings randomly or we can trace a given system and record the address of each memory reference. For example, if we trace a particular executing program and obtain the following address sequence:

0202, 0103, 0232, 0324, 0123, 0344, 0106, 0287, 0345, 0654, 0102, 0203, 0234, 0205, 0104, 0134, 0123, 0145, 0156, 0167

If size of each page is 100 bytes, the above address sequence is reduced to the reference string:

2, 1, 2, 3, 1, 3, 1, 2, 3, 6, 1, 2, 1

To determine the number of page faults for a particular reference string and page-replacement algorithm, we also need to know the number of page frames available. This is obvious that, if number of available-frames increases, the number of page faults decreases.

To illustrate the page replacement algorithms, we shall use the following reference string:

0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 3, 2, 4, 1

for a memory with 3 frames. The algorithm having least page fault is considered the best one.

Example *for FIFO*

Reference string / Frame	0	1	2	3	0	1	2	3	0	1	3	2	4	1
0	0	0	0	3	3	3	2	2	2	1		1	1	
1		1	1	1	0	0	0	3	3	3		2	2	
2			2	2	2	1	1	1	0	0		0	4	
Page fault	y	y	y	y	y	y	y	y	y	y	n	y	y	n

If a required page is already in main memory, page replacement is not required, which is indicated in the table by 'n'. In that situation, no page fault occurs.

From this table, we observe that 12 page faults occur, out of 14 references. Thus, the page fault rate is = no. of page faults/no. of page references in the string = 12/14 = 85%.

Example *of LRU*:

Reference string / Frame	0	1	2	3	0	1	2	3	0	1	3	2	4	1
0	0	0	0	3	3	3	2	2	2	1		1	4	4
1		1	1	1	0	0	0	3	3	3		3	3	1
2			2	2	2	1	1	1	0	0		2	2	2
Page fault	y	y	y	y	y	y	y	y	y	y	n	y	y	y

From this table, we observe that 13 page faults occur, out of 14 references. Thus, the page fault rate is = 13/14 = 93%.

Example *of Optimal replacement algorithm:*

Reference string / Frame	0	1	2	3	0	1	2	3	0	1	3	2	4	1
0	0	0	0	0			0			1			1	
1		1	1	1			2			2			2	
2			2	3			3			3			4	
Page fault	y	y	y	y	n	n	y	n	n	y	n	n	y	n

From this table, we observe that 7 page faults occur, out of 14 references. Thus, the page fault rate is = 7/14 = 50%.

Segmentation Segmentation is a memory management scheme that supports the user view of memory. A logical-address space of a program is a collection of segments. A *segment* is defined as a logical grouping of instructions, such as subroutine, array or data area. Each segment has a name and a length. The address of the segment specifies both segment name and offset within the segment. For simplicity of implementation, segments are referred to by a segment number rather than by a segment name. Thus, a logical address consists of two tuples: (*segment number (s), offset (d)*).

The mapping of logical address to corresponding physical address is done using *segment table*. Each entry of the segment table has a segment *base* and a segment *limit*. The segment base indicates the starting physical address where the segment resides in main memory and the segment limit

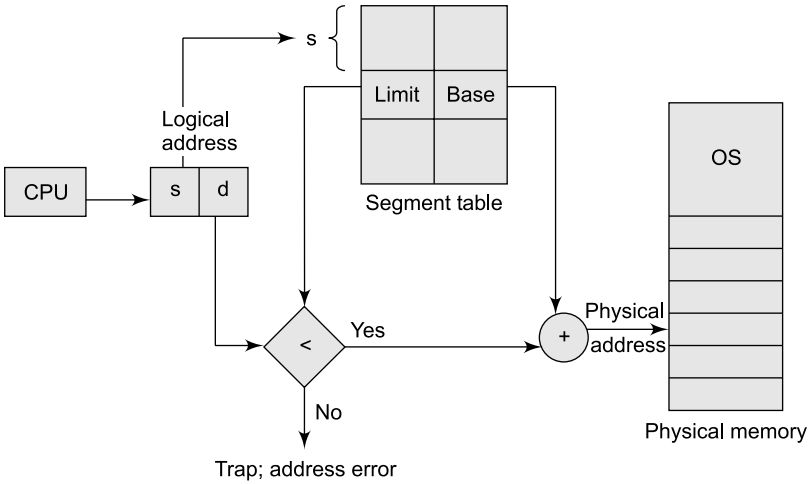


Figure 4.36 *Segmentation hardware*

specifies the length of the segment. The hardware implementation of segmentation is shown in Fig. 4.36.

A logical address consists of two fields: (segment number (s), offset (d)). The segment number s is used as an index in the segment table and the offset d is word number within the segment s. The offset d must be between 0 and the segment limit. If offset is beyond that range, the operating system generates an error signal (trap), which means that the offset is not valid. If it is valid, it is added with the segment base to produce the address in the physical memory for the desired word.

To illustrate the segmentation technique, we consider the example in Fig. 4.37. The logical address space is divided into 3 segments. Each segment has an entry in the segment table. The base specifies the starting address of the segment and the limit specifies the size of the segment. For example, first (i.e. 0^{th}) segment is loaded in the main memory from 1000 to 1500. Thus, the base is 1000 and limit is 1500 – 1000 = 500. A reference to word (byte) 20 of segment 0 is mapped onto address 1000 + 20 = 1020. Thus, the logical address (0 (segment no.), 20 (offset)) is mapped to the corresponding physical address 1020. Similarly, a reference to word 50 of segment 2 is mapped onto address 2500 + 50 = 2550. Thus, the logical address (2, 50) has the corresponding physical address 2550.

The *advantages* of segmentation are:

1. It supports efficient utilization of physical memory. Unlike paging, there is no space wastage within a segment.
2. It supports user view of memory more efficiently. It can handle dynamically growing segments.
3. Protection and sharing can be done easily.

Examples: The processors such as IBM 360/67 and VAX 11/780 use paging. Intel's microprocessor 80286 supports only segmentation; whereas microprocessor 80386 uses both segmentation and paging schemes.

Problem 4.7 Consider a logical address space of 8 pages of 1024 words each, mapped onto a physical memory of 32 frames.

(a) How many bits are there in the logical address?

(b) How many bits are there in the physical address?

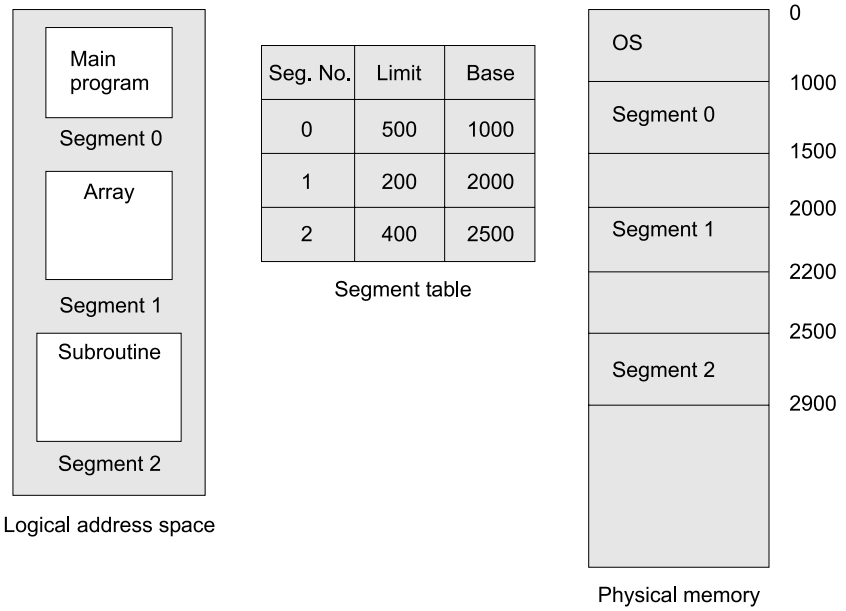


Figure 4.37 *Example of segmentation*

Solution

(a) Logical address space consists of 8 pages, each of 1024 words.
To select each page, 3 bits are required. (Because $8 = 2^3$)
To select each word in a page, 10 bits are required. (Because$1024 = 2^{10}$)
Therefore, the logical address consists of 3 + 10 = 13 bits.

(b) The physical memory consists of 32 frames.
So, to select each frame, 5 bits are required.
Since in paging, size of page and size of frame are equal.
Therefore, the physical address consists of 5 + 10 = 15 bits.

Problem 4.8 Consider the following segment table:

Segment	*Base*	*Length*
0	215	500
1	2000	160
2	1200	40

What are the physical addresses for the following logical addresses?
(a) 0, 430
(b) 1, 234
(c) 2, 13

Solution

(a) The physical address corresponding to 0, 430 is 215 (base) + 430 (offset) = 645.
(b) In logical address 1, 234; offset value 234 is greater than the length of segment 1 (i.e. 160). So, there is an error.
(c) The physical address corresponding to 2, 13 is 1200 + 13 = 1213.

Translation Look-aside Buffer (TLB)

In paging scheme, the main memory is accessed two times to retrieve data, one for accessing a page table and another for accessing data itself. Since the access time of the main memory is large, one new technique is adopted to speed up the data retrieval. A fast associative memory called *translation look-aside buffer (TLB)* is used to hold most recently used page table entries. When the CPU needs to access a particular page, the TLB is accessed first. If desired page table entry is present in the TLB, it is called *TLB hit* and then the frame number is retrieved from the table to get the physical address in main memory. If the desired page table entry is not present in the TLB, it is called *TLB miss* and then the CPU searches the original page table in main memory for the desired page table entry and this information (entry) is also loaded in the TLB. This ensures that translation information pertaining to a future reference is confined to the TLB. The organization of address translation scheme that includes a TLB is shown in Fig. 4.38.

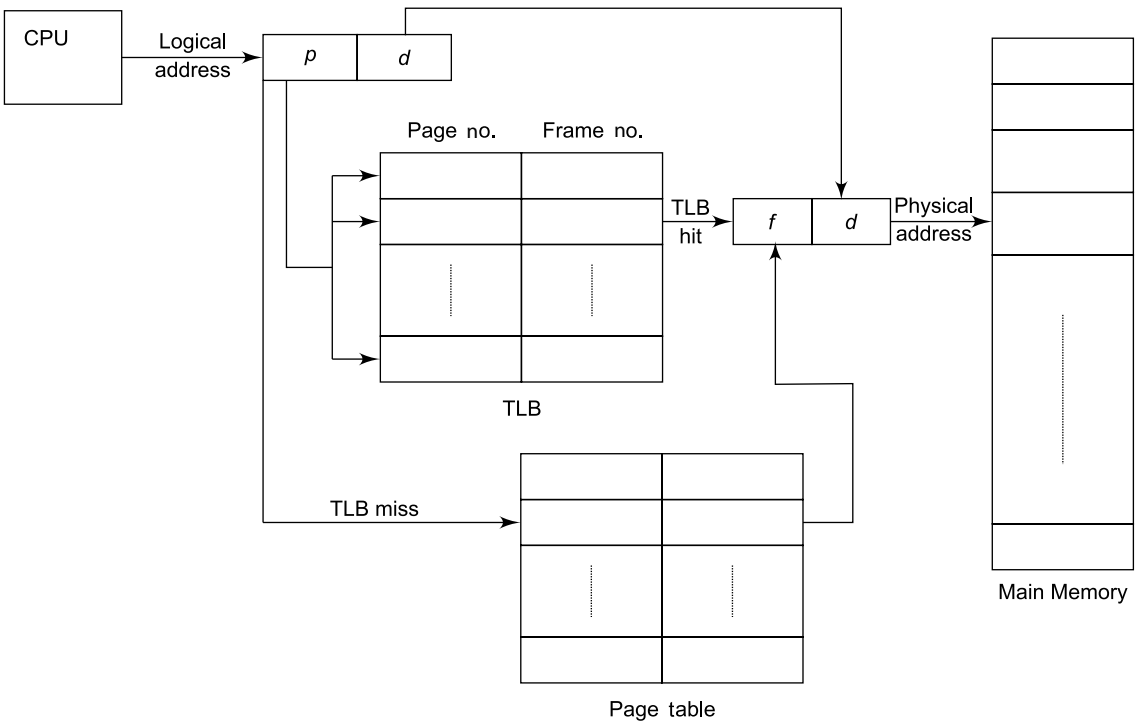


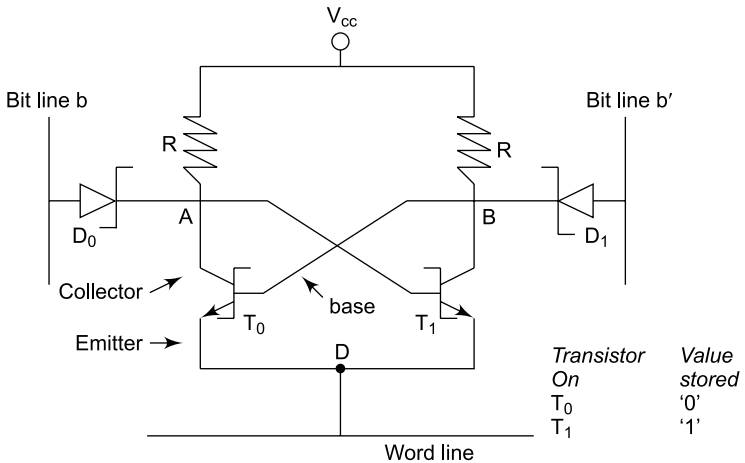
Figure 4.38 *Address translation using TLB*

This concept is used in many mainframes, including the IBM 370/168 computer. In the IBM 370/168, the TLB can hold 128 entries. With the advent of IC technology, this technique is also gaining popularity in the microprocessor world.

SOLVED PROBLEMS

1. *Describe the storage structure of a bipolar storage cell and explain the reading and writing operations on the cell. Give a suitable diagram.*

Answer



Two junction transistors T_0 and T_1 are connected in such a way that they form a flip-flop. Assuming point D at ground voltage (i.e. word line is disabled), when T_0 is on, the flip-flop stores logic '0' while current flows to ground through T_0 making point A at '0' level. This in turns holds T_1 off. As a result, the point B is held at voltage equal to the base-emitter voltage of T_0 (i.e. B at logic '1') whereby T_0 is held on. Similarly, when T_1 is on, the point B is at '0' level while the point A is held at the base-emitter voltage of T_1 (i.e. A at logic '1') whereby T_1 is held on. Depending on 0 (or 1) stored, T_0 (or T_1) is on and both the diodes D_0 and D_1 are reverse biased. As a result both the bit lines b and b′ are isolated from the flip-flop cell.

Read operation: On selecting the word line, the voltage at D goes down at ground level, which selects the cell. Depending on the T_0 (or T_1) conducting, the point A (or B) gets the voltage close to ground level. As a result D_0 (or D_1) gets forward biased. Depending on D_0 (or D_1) being forward biased, the appropriate logic value is read through bit line b (or b′).

Write operation: On selecting the word line, the voltage at D goes down at ground level to select the cell. Depending on the logic '1' (or '0') to be written, the bit line b (or b′) is held high whereby D_0 (or D_1) gets forward biased and consequently T_1 (or T_0) switch is on resulting T_0 (or T_1) off.

2. *How do the following influence the performance of a virtual memory system?*
   (i) *Size of a page* (ii) *Replacement policy*

Answer

(i) *Page size:* If page size is large, the page fault rate will be less. But, in that case, transfer time of the page will increase.
If page size is small, the memory is better utilized, but number of pages and hence the size of page table will be large.

(ii) *Replacement policy:* When a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several replacement

policies such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement algorithm* available. The performance of virtual memory is degraded if too many page faults occur, because that lead to bring the new required pages to the physical memory. That's why the algorithm which gives lowest page faults is considered as best algorithm.

3. *A computer has direct mapped cache with 16 one-word blocks. The cache is initially empty. What is the observed hit ratio when the CPU generates the following word address sequence: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17?*

Answer

The direct mapping is expressed as

I = J mod K

Where

I = cache block number
J = main memory block number
K = number of blocks in cache.

The processor generates the addresses for words in main memory. In our problem, K = 16 with one word per block and main memory block sequence is: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.

Thus, the corresponding cache block sequence and its word is as (block no., word address): (1, 1), (4,4), (8,8), (5,5), (4,20), (1,17), (3,19), (8,56), (9,9), (11, 11), (4,4), (11,43), (5,5), (6,6), (9,9), (1,17). Initially, cache is empty.

(Cache block no., word address)	(1,1)	(4,4)	(8,8)	(5,5)	(4,20)	(1,17)	(3,19)	(8,56)	(9,9)
Hit(H)/Miss (M)	M	M	M	M	M and replace	M and replace	M	M and replace	M

(Cache block no., word address)	(11,11)	(4,4)	(11,43)	(5,5)	(6,6)	(9,9)	(1,17)
Hit(H)/Miss(M)	M	M	M and replace	H	M	H	H

Therefore, hit ratio = 3/16.

4. *What is the bandwidth of a memory system that transfers 128-bit of data per reference, has a speed 20 ns per operation?*

Answer

Given the speed of 20 ns, one memory reference can initiate in every 20 ns and each memory reference fetches 128-bit (i.e. 16 bytes) of data. Therefore, the bandwidth of the memory system is 16 bytes/20 ns = $(16 \times 10^9)/20$ bytes per second = 8×10^8 bytes per second.

5. *What will be the maximum capacity of a memory, which uses an address bus of size 12-bit?*

Answer

The maximum capacity of memory will be 2^{12} words i.e. 4096 words.

6. *Why is the memory system of a computer organized as a hierarchy?*

Answer

Ideally, we would like to have the memory which would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three requirements simultaneously. If we increase the speed and capacity, then cost will increase. We can achieve these goals at optimum level by using several types of memories, which collectively give a memory hierarchy.

The lower levels of memory hierarchy, which are implemented using slow and cheap memory technologies, contain most of programs and data. The higher levels of memory hierarchy, which are

implemented using fast and expensive memory technologies, contain smaller amount of programs and data. The processor, being very high speed device, references data in the fast higher levels of memory hierarchy. If referred data is not available there, it is moved from lower levels of the hierarchy so that the higher levels handle most references. If most references are handled by the higher levels, the memory system gives an average access time almost same as the fastest level of the memory hierarchy, with a cost per bit same as that of the lowest level of the hierarchy.

7. *What are destructive read out memory and non-destructive read out memory? Give examples.*

Answer

In some memories, reading the memory word destroys the stored word, this fact is known as *destructive readout* and memory is known as *destructive readout memory*. In these memories, each read operation must be followed by a write operation that restores the memory's original state. Example includes dynamic RAM.

In some memories, the reading the memory word does not destroy the stored word, this fact is known as *non-destructive readout* and memory is known as *non-destructive readout memory*. Examples include static RAM and magnetic memory.

8. *Why do the DRAMs generally have large capacities than SRAMs constructed by the same fabrication technology?*

Answer

Each DRAM cell contains two devices – one capacitor and one transistor, while each SRAM cell consists of six transistors. This means a DRAM cell is much smaller than a SRAM cell, allowing the DRAM to store more data in the same size chip space.

9. *Why is refreshing required in Dynamic RAM?*

Answer

Information is stored in a dynamic RAM memory cell in the form of a charge on a capacitor. Due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitor drops below some threshold value. Thus, this charge in capacitor needs to be periodically recharged or refreshed.

10. *Suppose a DRAM memory has 4 K rows in its array of bit cells, its refreshing period is 64 ms and 4 clock cycles are needed to access each row. What is the time needed to refresh the memory if clock rate is 133 MHz? What fraction of the memory's time is spent performing refreshes?*

Answer

In DRAM memory, no. of rows of cells in memory is 4K = 4096 and 4 clock cycles are needed to access each row.

Therefore, no. of cycles needed to refresh all rows = $4096 \times 4 = 16384$ cycles.

Since clock rate is 133 MHz,

The time needed to refresh all rows = $16384 / (133 \times 10^6)$ seconds

$= 123 \times 10^{-6}$ seconds

$= 0.123$ ms. [1 ms = 10^{-3} sec.]

Thus, the refreshing process occupies 0.123 ms in each 64 ms time interval.

Therefore, refresh overhead is 0.123/64 = 0.002.

Hence, only 0.2 % of the memory's time is spent performing refreshes.

11. *How many 256 × 4 RAM chips are needed to provide a memory capacity of 2048 bytes? Show also the corresponding interconnection diagram.*

Answer

The given RAM memory size is 256 × 4. This memory chip requires 8 (because $256 = 2^8$) address lines and 4 data lines.

Size of memory to be constructed is 2048 bytes, which is equivalent to 2048 × 8. Thus, it requires 11 (because $2048 = 2^{11}$) address lines and 8 data lines.

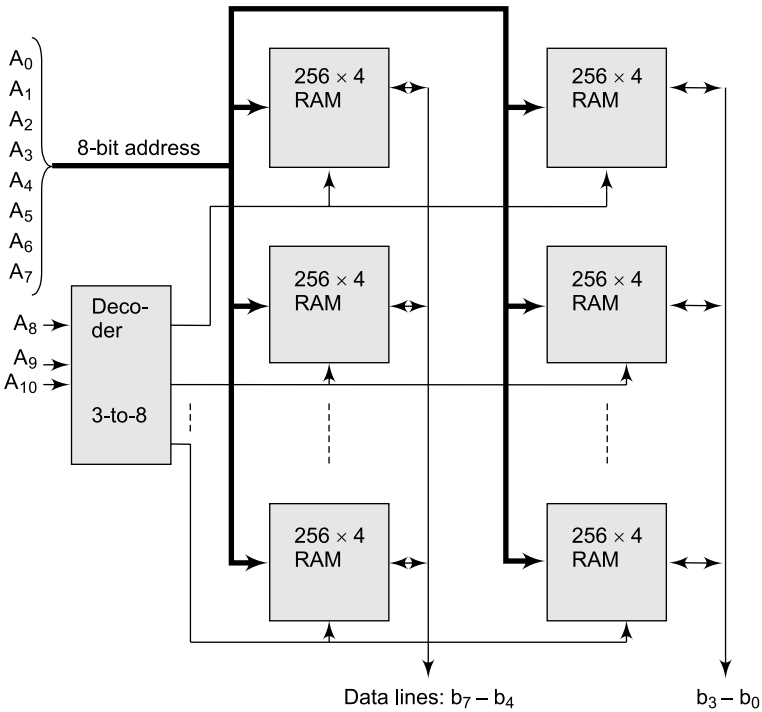
In the interconnection diagram:

The number of rows required = 2048/256 = 8.

The number of columns required = 8/4 = 2.

Thus, total number of RAMs each of size 256 × 4 required = 8 * 2 = 16.

The interconnection diagram is shown in the following figure.



12. *Explain how a RAM of capacity 2 K bytes can be mapped into the address space $(1000)_H$ to $(17FF)_H$ of a CPU having a 16 bit address lines. Show how the address lines are decoded to generate the chip select condition for the RAM.*

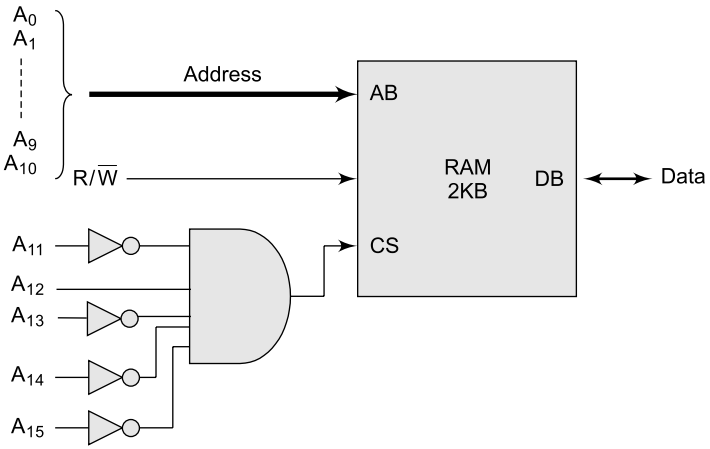
Answer

Since the capacity of RAM memory is 2048 bytes = 2 KB, the memory uses 11 (2 KB = 2^{11}) address lines, say namely $A_{10} - A_0$, to select one word. Thus, memory's internal address decoder uses 11 lines $A_{10} - A_0$ to select one word.

To select this memory module, remaining 5 (i.e. 16 – 11) address lines $A_{15} - A_{11}$ are used. Thus, an external decoding scheme is employed on these higher-order five address bits of processor's address. The address space of the memory is 1000_H and $17FF_H$.
Therefore, the starting address $(1000)_H$ in memory is as:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Based on the higher-order five bits (00010), external decoding scheme performs a logical AND operation on address values: $\overline{A_{15}}$, $\overline{A_{14}}$, $\overline{A_{13}}$, A_{12} and $\overline{A_{11}}$. The output of AND gate acts as chip select (CS) line. The address decoding scheme is shown in the following figure.



13. *A high speed tape system accommodates 1200 ft. reel of standard 9-track tape. The tape is moved past the recording head at a rate of 140 inches per second. What must be the linear tape recording density in the order to achieve a data transfer rate of 10^5 bits per second?*

Answer
Given,
Tape length = 1200 ft.
Tape speed = 140 inches/sec.
Data transfer rate = 10^5 bits/sec.
We have, data transfer rate = tape density × tape speed
Therefore, the tape density = $10^5/140$ bits/inch
= 714 bits/inch

14. *Suppose a 30 GB hard-disk is to be manufactured. If the technology used to manufacture the disks allows 1024-byte sectors, 2048-sector tracks and 4096-track platters. How many platters are required?*

Answer
The total capacity of each platter = size of each sector × no. of sectors per track × no. of tracks per platter

$= 1024 \times 2048 \times 4096$ bytes
$= 8 \times 2^{30}$ bytes
$= 8$ GB

Therefore, no. of platters required $= \lceil$capacity_of_disk/capacity_of_each_platter$\rceil$
$= \lceil 30/8 \rceil$
$= 4.$

15. *What is the average time to read or write a 512-byte sector from a disk rotating at 5,400 RPMs, if the seek time is 12 ms, the transfer rate is 4 MB per second and the controller overhead is 8 ms?*

Answer

Disk access time = seek time + rotational latency + transfer time + control overhead
Rotational latency = (60 * 1000)/5400 = 11.11 ms
Transfer time = 512/4MB = 0.122 ms
Disk access time = 12 + 11.11 + 0.122 + 8 ms = 31.232 ms

16. *A hierarchical cache-main memory subsystem has the following specifications:* (i) *Cache access time of 160 ns* (ii) *main memory access time of 960 n* (iii) *hit ratio of cache memory is 0.9. Calculate the following:*
    (a) *Average access time of the memory system.*
    (b) *Efficiency of the memory system.*

Answer

Given,
Cache access time, $t_c = 160$ ns
Main memory access time, $t_m = 960$ ns
Hit ratio, $h = 0.9$

(a) The average access time of the system, $t_{av} = h \times t_c + (1 - h) \times (t_c + t_m)$
$= 0.9 \times 160 + 0.1 \times (160 + 960)$
$= 256$ ns

(b) The efficiency of the memory system $= t_c / t_{av}$
$= 160/256$
$= 0.625$

17. *A three level memory system having cache access time of 15 ns and disk access time of 80 ns has a cache hit ratio of 0.96 and main memory hit ratio of 0.9. What should be the main memory access time to achieve effective access time of 25 ns?*

Answer

Given,
Cache access time, $t_c = 15$ ns
Disk (secondary) memory access time, $t_s = 80$ ns
Hit ratio for cache, $h_c = 0.96$
Hit ratio for main memory, $h_m = 0.9$

The average access time, t_{av} = 25 ns

Let, the main memory access time is t_m unit.

Now, we know, the average access time of the memory system,

$$t_{av} = h_c \times t_c + h_m \times (1–h_c) \times (t_c + t_m) + (1–h_c) \times (1–h_m) \times (t_c + t_m + t_s)$$

That is, $25 = 0.96 \times 15 + 0.9 \times 0.04 \times (15 + t_m) + 0.04 \times 0.1 \times (15 + t_m + 80)$

By simplifying, we get, $t_m = 27$

Hence, the main memory access time must be 27 ns to achieve the effective access time of 25 ns.

18. *Explain how cache memory increases the performance of a computer system.*

Answer

Due the locality of reference property of programs, some blocks like program loop, subroutine and data array in the programs are referenced frequently. Since the cache memory's speed is almost same as that of CPU. When these program blocks are placed in fast cache memory, the average memory access time is reduced, thus reducing the total execution time of the program.

19. *How does the size of cache block (i.e. line) affect the hit ratio?*

Answer

Generally, increasing the block size of a cache increases the hit ratio because of the property of locality of reference, i.e. the addresses close to an address that has just been referenced are likely to be referenced soon. Increasing the block size increases the amount of data near the address that caused the miss that is brought into the cache on a cache miss. Since most of this data is likely to be referenced soon, bringing it into the cache eliminates the cache misses that would have occurred when data was referenced.

However, sometimes large block can reduce the performance of the memory system. Since the larger blocks reduce the number of blocks in the cache, conflict misses can arise giving the reduced performance of the system.

20. *Given the following, determine size of the sub-fields (in bits) in the address for direct mapping, associative and set associative mapping cache schemes: We have 256 MB main memory and 1MB cache memory. The address space of this processor is 256 MB. The block size is 128 bytes. There are 8 blocks in a cache set.*

Answer

Given,

The capacity of main memory = 256 MB

The capacity of cache memory = 1 MB

Block size = 128 bytes.

A set contains 8 blocks.

Since, the address space of the processor is 256 MB.

The processor generates address of 28-bit to access a byte (word).

The number of blocks main memory contains = 256 MB/128 bytes = 2^{21}.

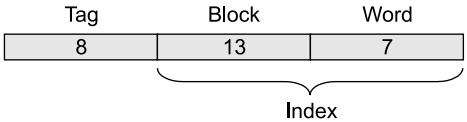
Therefore, no. of bits required to specify one block in main memory = 21.

Since the block size is 128 bytes.

The no. of bits required to access each word (byte) = 7.
For associative cache, the address format is:

Tag	Word
21	7

The number of blocks cache memory contains = 1 MB / 128 bytes = 2^{13}.
Therefore, no. of bits required to specify one block in cache memory = 13.
The tag field of address = 28 – (13 + 7) = 8-bit.
For direct cache, the address format is:



In case of set-associative cache:
A set contains 8 blocks.
Therefore, the number of sets in cache = 2^{13} / 8 = 2^{10}.
Thus, the number of bits required to specify each set = 10.
The tag field of address = 28 – (10 + 7) = 11-bit.
For set-associative cache, the address format is:

Tag	Set	Word
11	10	7

21. *Why do virtual page and physical frame have same size, in paging?*

Answer

Virtual address that is generated by processor is divided into two fields; page number that identifies the page containing an address word and offset that identifies the location of the address (word) within the page. Similarly, physical address is divided into two fields; frame number and offset. If the page size and frame size are same, address translation can be done easily. The offset from the virtual address can be concatenated with the frame number that corresponds to the virtual page containing the address to produce the physical address that corresponds to a virtual address. If these two were different, a complicated means would be required for address translation.

22. *In a system with 64-bit virtual addresses and 43-bit physical addresses, how many bits are required for the virtual page number and physical frame number if the pages are 8 KB in size? How big is each page table entry? How many page table entries are required for this system?*

Answer

Since the page size is 8 KB, 13 bits are required for the offset field of both the virtual and physical address.

Therefore, bits required for the virtual page number = 64 – 13 = 51 and

Bits required for the physical frame number = 43 – 13 = 30.

Each page table entry contains frame number and a valid/invalid bit.

So, a total of (30 + 1) i.e. 31 bits is used to store each page table entry.

Since each virtual page number contains 51 bits,

The virtual address space can hold 2^{51} pages, which requires 2^{51} page table entries.

23. *A virtual memory system has the following specifications:*
    - *Size of the virtual address space is 64 KB*
    - *Size of physical address space is 4 KB*
    - *Page size is 512 byte*

    From the following page table, what are the physical addresses corresponding to the virtual addresses:

    (a) *3494* (b) *12350* (c) *30123*

Page number	*Frame number*
0	*0*
3	*1*
7	*2*
4	*3*
10	*4*
12	*5*
24	*6*
30	*7*

Answer

Since page size is 512 bytes

Lower order 9 bits are used for offset within the page.

Size of the virtual address space is 64 KB

Therefore, each virtual address consists of 16-bit.

Thus, higher order 16–9 i.e. 5 bits specify the virtual page number.

(a) For virtual address 3494; lower (494) number is the offset and higher (3) is the page number. Now, looking in the table, we find frame number (1) corresponding to the page number (3). By concatenating the offset within the page with the physical frame number, we get physical address 1494 corresponding to the virtual address 3494.

(b) For virtual address 12350, the physical address is 5350.

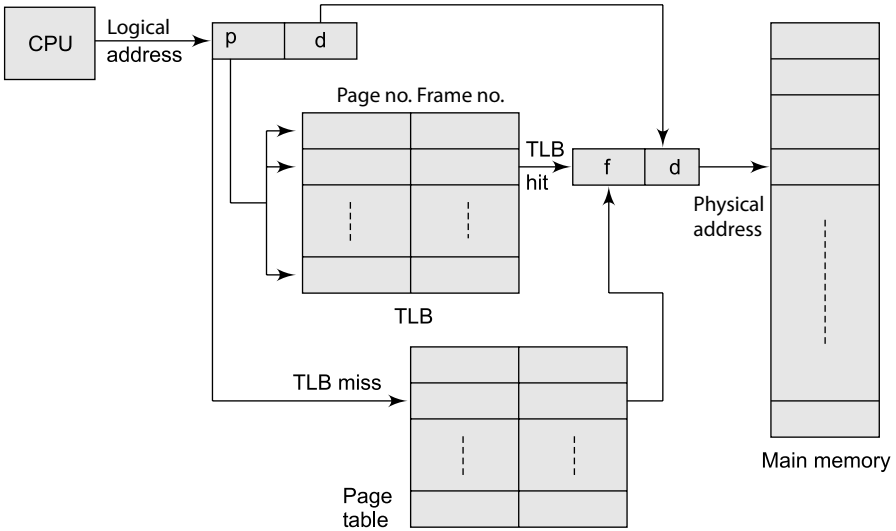
(c) For virtual address 30123, the physical address is 7123.

24. *What is a translation look-aside buffer (TLB)?*

Answer

In paging scheme, the main memory is accessed two times to retrieve data, one for accessing page table and another for accessing data itself. Since the access time of main memory is large, one new technique is adopted to speed up the data retrieval. A fast associative memory called *translation look-aside buffer (TLB)* is used to hold most recently used page table entries. When CPU needs to access a

particular page, the TLB is accessed first. If desired page table entry is present in the TLB, it is called *TLB hit* and then the frame number is retrieved from the table to get the physical address in main memory. If the desired page table entry is not present in the TLB, it is called *TLB miss* and then CPU searches the original page table in main memory for the desired page table entry. The organization of address translation scheme that includes a TLB is shown in the following figure.



25. *Suppose a processor's TLB has hit ratio 80% and it takes 20 ns to search the TLB and 100 ns to access main memory. What will be the effective access time?*

Answer

When the referred page number is found in the TLB, then a mapped memory access takes (20 + 100) i.e. 120 ns. If it is not found in the TLB, then we must first access memory for the page table and frame number, and then access the desired word in memory, for a total of (20 + 100 + 100) i.e. 220 ns.

Therefore, the effective access time = $(TLB_{hit} \times TIME_{hit}) + (TLB_{miss} \times TIME_{miss})$
$= 0.8 \times 120 + 0.2 \times 220$ ns
$= 140$ ns

26. *Why does the virtual memory prevent programs from accessing each other's data?*

Answer

Each program has its own virtual address space. The virtual memory system translates different programs' virtual addresses to different physical addresses so that no two programs' virtual addresses map onto the same physical addresses, thus preventing the programs from accessing each other's data.

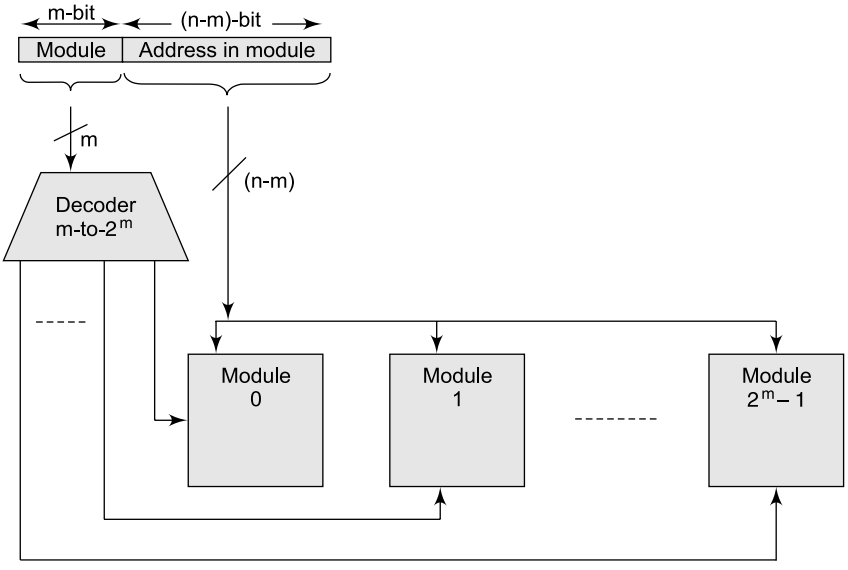
27. *What is memory interleaving? What are the varieties of it? Discuss.*

Answer

The main memory is partitioned into several independent memory modules (chips) and addresses distributed across these modules. This scheme, called interleaving, allows concurrent accesses to more than one module. The interleaving of addresses among M modules is called M-way interleaving.

There are two basic methods, *higher-order* and *lower-order interleaving*, of distributing the addresses among the memory modules. Assume that there are a total of $N = 2^n$ words in main memory. Then the physical address for a word in memory consists of n bits, $a_{n-1}\ a_{n-2} \ldots a_1 a_0$. One method, *high-order interleaving*, distributes the addresses in $M = 2^m$ modules so that each module i, for $0 \le i \le M-1$, contains consecutive addresses $i2^{n-m}$ to $(i+1)2^{n-m} - 1$, inclusive. The high-order m bits are used to select the module while the remaining n-m bits select the address within the module, as shown in next diagram.

The second method, called *low-order interleaving*, distributes the addresses so that consecutive addresses are located within consecutive modules. The low-order m bits of the address select the module, while the remaining n-m bits select the address within the module. Hence, an address A is located in module A mod M.



M-way higher-order interleaving

28. *The following measurements are given for a computer system that uses a paged memory system with a TLB:*
    - *Time taken to search in the TLB: 20 ns*
    - *Main memory access time: 100 ns.*

    Determine the average access time assuming a TLB hit ratio of 0.8.

Answer

In the event of a TLB hit, the time needed to retrieve the data is

t_1 = TLB search time + time for one memory access
= 20 + 100 ns
= 120 ns

However, when a TLB miss occurs, the main memory is accessed twice (first access for page table and frame number and then second access for the desired data in memory) to retrieve the data.

Therefore, the retrieval time t_2 in this case is

t_2 = TLB search time + 2 (time for one memory access)
= 20 + 200 ns
= 220 ns

The average access time t_{av} is

$t_{av} = h\ t_1 + (1 - h)\ t_2$ where h is the TLB hit ratio.
= 0.8 * 120 + 0.2 * 220 ns
= 140 ns

29. *Consider three machines with different cache organizations, and measured instruction and data miss rates as:*

Instruction Miss Rate	*Data Miss Rate*		*Organization*
Cache 1:	4%	8%	Direct mapped with one-word blocks.
Cache 2:	2%	5%	Direct mapped with four-word blocks.
Cache 3:	2%	4%	Two-way set associative with four-word blocks.

Assume each instruction takes one reference to memory for the instruction, and one-third of the instructions take one reference to memory for data (the other two-thirds have no data references). The cache miss penalty is 6 + W, where W is the number of words in a cache block. Compute the aggregate miss rate and the average memory access time for each machine. Find which machine spends the least and most cycles on cache misses.

Answer

Each instruction takes one reference to memory, and 1/3 of instructions also require a data reference; thus, for every four memory references, 3 are instructions and 1 is data, or 3/4 of memory references are for instructions, 1/4 for data. If the instruction miss rate is X and the data miss rate is Y, the aggregate miss rate is simply 0.75X + 0.25Y.

For Cache 1: 0.75*4% + 0.25*8% = 5%
For Cache 2: 0.75*2% + 0.25*5% = 2.75%
For Cache 3: 0.75*2% + 0.25*4% = 2.5%

The average memory access time is computed as hit-time + miss-rate*miss-penalty. The cache miss penalty is 6 + W, where W is the number of words in a cache block. We use the aggregate miss-rate here, and assume hit-time is one.

Average access time of Cache 1: 1 + 0.05*7 = 1.35
Average access time of Cache 2: 1 + 0.0275*10 = 1.275
Average access time of Cache 3: 1 + 0.025*10 = 1.25

Note that Cache 3 has a miss rate 1/2 that of Cache 1, but the average memory access time is very close.

Every 7500 instructions will include 2500 data references, for 10000 memory references. Cache 1 will have 500 misses in those references, using 3500 cycles to fill the cache. Cache 2 will have 275 misses, using 2750 cycles to fill the cache line. Cache 3 will have 250 misses, using 2500 cycles to fill the cache line. Thus, Cache 3 uses the least cycles for cache fills.

30. *Which machine gives better performance between the following two?*
    (a) A machine with a base CPI of 1, overall cache miss rate of 3%, cache penalty of 10 cycles, and clock cycle time of C (frequency = 1/C), or
    (b) A machine with base CPI of 1, overall cache miss rate of 5%, cache penalty of 16 cycles, and clock cycle time of 0.6C (frequency = 5/(3C)).
    Assume each instruction takes one reference to memory for the instruction, and one-third of the instructions take one reference to memory for data (the other two-thirds have no data references).

Answer

For IC (instruction count) instructions, the execution time is IC*CPI*C, where CPI is clock cycles per instruction and C is the cycle time.

We have one instruction reference for each instruction and one data reference for every 3 instructions. The cache miss rate will cause an extra MP (miss penalty) cycles for every MR (miss rate) memory references, which occur at a rate of MRI (memory references per instruction).

Therefore, the execution time formula is IC*(CPI+MRI*MR*MP)*C.

Execution time of machine 1: IC*(1 + 4/3*0.03*10)*C = 1.4 *IC*C

Execution time of machine 2: IC*(1 + 4/3*0.05*16)*0.6C = 1.24 *IC*C

Comparing these two, the IC and C terms drop out.

So, machine 2 gives better performance, because it takes less time.

31. *Consider the performance of a main memory organization, when a cache miss has occurred, as*
    - *4 clock cycles to send the address*
    - *24 clock cycles for the access time per word*
    - *4 clock cycles to send a word of data.*

    Estimate:
    (i) *The miss penalty for a cache block of 4 words.*
    (ii) *The main memory bandwidth.*

Answer

Given, a cache block of 4 words.

(i) Therefore, the miss penalty is 4 × (4 + 24 + 4) or 128 clock cycles.

(ii) The memory bandwidth of 4/128 = 1/32 words per clock cycle.

32. *Show the memory map with a CPU having 8 bit data bus and 16 bit address bus requring four RAM chips of size 256 x 8 bit each and a ROM chip of 512 x 8 bit size. Explain the meory map.*

Answer

The addressing of memory can be designed by means of a table, known as memroy address map, which specifies the meory address space assigned to each chip. The address map table for the memory connection to the CPU shown in Figure 8 is constructed in Table.

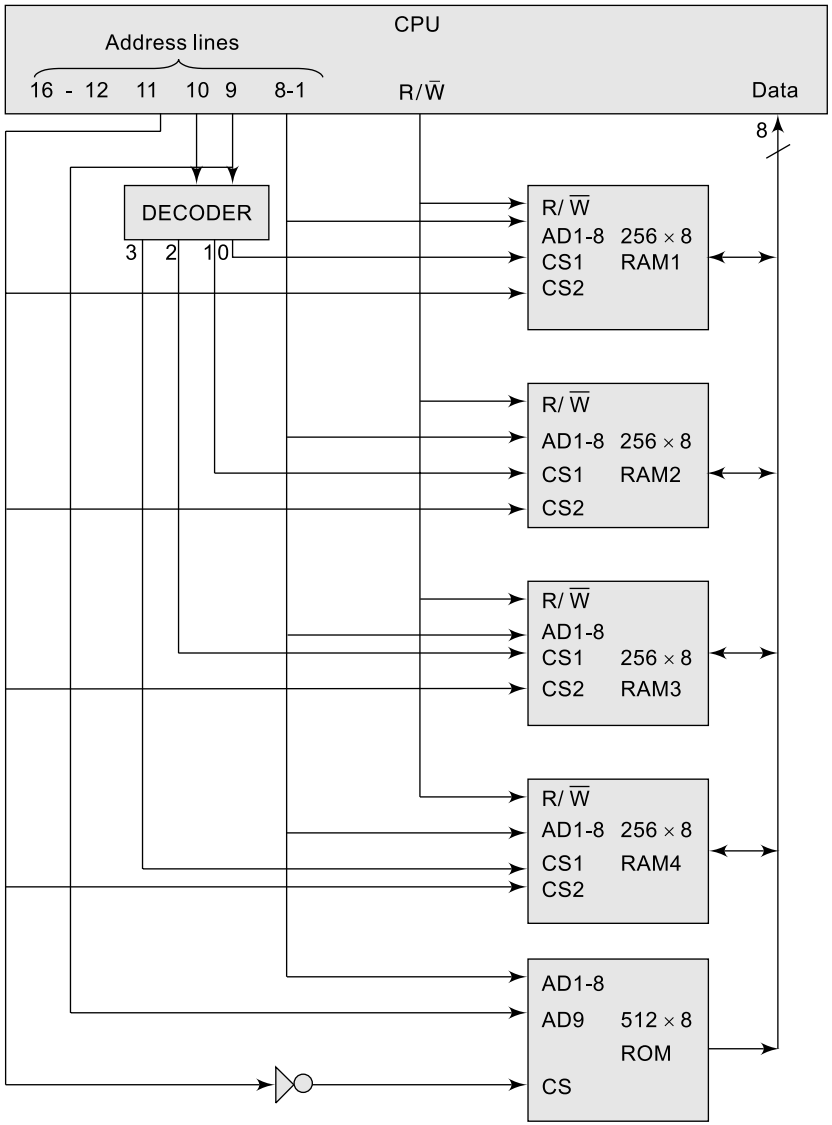


Figure 8 *Memory connection with 16-bit CPU*

The CPU generates 16-bit address for memory read or write operation. The address lines 1 to 8 are connected to each memory and address line 9 is used in dual purposes. In case of a RAM selection out of four RAMs, the line no. 9 and line no. 10 are used through a 2-to-4 decoder. The line no. 9 is also connected to the ROM as address line along with lines 1 to 8 giving a total of 9 address lines in the ROM, since the ROM has 512 locations. The CPU address line number 11 is used for separation between RAM and ROM. The other 12 to 16 lines of CPU are unused and for simplicity we assume that they carry 0s as address signals. For ROM, 10^{th} line is unused and thus it can be assumed that this line carries signal 0.

Table *Memory address map table*

Chip selected	*Address space (in HEX)*	*Address bus*										
		11	10	9	8	7	6	5	4	3	2	1
RAM1	0400 – 04FF	1	0	0	x	x	x	x	x	x	x	x
RAM2	0500 – 05FF	1	0	1	x	x	x	x	x	x	x	x
RAM3	0600 – 06FF	1	1	0	x	x	x	x	x	x	x	x
RAM4	0700 – 07FF	1	1	1	x	x	x	x	x	x	x	x
ROM	0000 – 01FF	0	0	x	x	x	x	x	x	x	x	x

33. *What is Blu-ray Disc (BD)?*

Answer

After DVD, *Blu-ray Disc (BD)* (not Blue-ray) is the name of a new optical disc format jointly developed by the Blu-ray Disc Association (BDA), a group of the world's leading consumer electronics, personal computer and media manufacturers, in the year 2006. The format was developed to enable recording, rewriting and playback of high-definition video (HD), as well as storing large amounts of data. The format offers more than five times the storage capacity of traditional DVDs and can hold up to 25GB on a single-layer disc and 50GB on a dual-layer disc.

While current optical disc technologies such as DVD, DVD±R, DVD±RW, and DVD-RAM rely on a red laser to read and write data, the new BD format uses a blue-violet laser instead, hence the name Blu-ray. Despite the different type of lasers used, Blu-ray products can easily be made backwards compatible with CDs and DVDs through the use of a BD/DVD/CD compatible optical pickup unit. The benefit of using a blue-violet laser (405nm) is that it has a shorter wavelength than a red laser (650nm), which makes it possible to focus the laser spot with even greater precision. This allows data to be packed more tightly and stored in less space, so it is possible to fit more data on the disc even though it is the same size as a CD/DVD. This together with the change of numerical aperture to 0.85 is what enables Blu-ray Discs to hold 25GB/50GB. Blu-ray disc also has a higher data transfer rate - 36 Mbps (megabits per second) - than today's DVDs, which transfer at 10 Mbps. A Blu-ray disc can record 25 GB of material in just over an hour and a half.

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:
   (i) How many memory locations can be addressed by a 32-bit computer?
   (a) 64 KB (b) 32 KB (c) 4 GB (d) 4 MB
   (ii) The access time of memory is the time duration
   (a) from receiving read/write signal to the completion of read/write operation
   (b) from receiving an address value to the completion of read/write operation
   (c) from receiving a chip enable signal to the completion of read/write operation
   (d) for all memory operations starting from enable signal.

(iii) The bandwidth of memory accesses is in ascending order for
(a) cache, DRAM, SDRAM and RDRAM
(b) SRAM, DRAM, SDRAM and cache
(c) DRAM, SDRAM, RDRAM and cache
(d) RDRAM, DRAM, SDRAM and cache.

(iv) The memory hierarchy system in respect of increasing speed consists of
(a) secondary, main, cache and internal
(b) internal, main, cache and secondary
(c) internal, secondary, main and cache
(d) cache, main, secondary and internal.

(v) The memory hierarchy system in respect of increasing cost consists of
(a) secondary, main, cache and internal
(b) internal, main, cache and secondary
(c) internal, secondary, main and cache
(d) cache, main, secondary and internal.

(vi) The semi-random access mechanism is followed in
(a) RAM (b) ROM (c) magnetic (d) register.

(vii) The associative access mechanism is followed in
(a) main (b) cache (c) magnetic (d) both (a) and (b).

(viii) The initial bootstrap program is generally stored in
(a) RAM (b) ROM (c) magnetic (d) cache.

(ix) If the size of a RAM is 1 GB (assuming it is byte-addressable), it means that it has number of address lines and number of data lines as
(a) 20 and 8 (b) 20 and 16 (c) 30 and 8 (d) 10 and 32.

(x) To construct a RAM memory of capacity 512 words each of size 12 bits using RAM chips each of size 128×4, the number of smaller size RAM chips required is
(a) 4 (b) 8 (c) 12 (d) 16.

(xi) A computer's memory is composed of 8 K words of 32 bits each. How many bits are required for memory address if the smallest addressable memory unit is a word?
(a) 13 (b) 8 (c) 10 (d) 6

(xii) A computer's memory is composed of 4 K words of 32 bits each. How many total bits are there in the memory?
(a) 12800 (b) 1280000 (c) 1310720 (d) 131072

(xiii) A computer's memory is composed of 8 K words of 32 bits each, and a byte is 8 bits. How many bytes does this memory contain?
(a) 8 K (b) 32 K (c) 16 K (d) 4 K

(xiv) A computer's memory is composed of 8 K words of 32 bits each, and the smallest addressable memory unit is an 8-bit byte. How many bits will be required for the memory address?
(a) 12 (b) 15 (c) 13 (d) 10

(xv) The number of transistors required in SRAM, DRAM and ROM are
(a) 4, 2, 2 (b) 1, 1, 4 (c) 1, 6, 1 (d) 6, 1, 1; respectively.

(xvi) Flash memory is
(a) SRAM (b) DDR SDRAM (c) PROM (d) EEPROM.

(xvii) What characteristic of RAM memory makes it not suitable for permanent storage?
(a) Too slow (b) Unreliable (c) It is volatile (d) Too bulky

(xviii) Part of the operating system is usually stored in ROM so that it can be used to boot up the computer. ROM is used rather than RAM because
(a) ROM chips are faster than RAM
(b) ROM chips are not volatile
(c) ROM chips are cheaper than RAM chips
(d) None of the above.

(xix) A given memory chip has 12 address pins and 4 data pins. It has the following number of locations:
(a) 2^4 (b) 2^{12} (c) 2^{48} (d) 2^{16}.

(xx) RAM is called DRAM (Dynamic RAM) when
(a) it is always moving around data
(b) it requires periodic refreshing
(c) it can do several things simultaneously
(d) none of the above.

(xxi) Which of the following is non-volatile memory?
(a) EEPROM (b) SRAM (c) DRAM (d) None of the above

(xxii) The cylinder in a disk pack is
(a) collection of all tracks in a surface
(b) logical view of same radius tracks on different surfaces of disks
(c) collection of all sectors in a track
(d) collection of all disks in the pack.

(xxiii) The amount of time required to read a block of data from a disk into memory is composed of seek time, rotational latency, and transfer time. Rotational latency refers to
(a) the time it takes for the platter to make a full rotation
(b) the time it takes for the read-write head to move into position over the appropriate track
(c) the time it takes for the platter to rotate the correct sector under the head
(d) none of the above.

(xxiv) If a magnetic disc has 100 cylinders, each containing 10 tracks of 10 sectors, and each sector can contain 128 bytes, what is the maximum capacity of the disk in bytes?
(a) 128,000 (b) 12,800,000 (c) 12,800 (d) 1,280,000

(xxv) According to the specifications of a particular hard disk a seek takes 3 msecs (thousandths of a second) between adjacent tracks. If the disk has 100 cylinders, how long will it take for the head to move from the innermost cylinder to the outermost cylinder?
(a) 30 ms (b) 300 ms (c) 3000 ms (d) 3 ms

(xxvi) The locality of reference property justifies the use of
(a) secondary memory (b) main memory
(c) cache memory (d) register.

(xxvii) The order of CPU references in memories is as
(a) secondary, main and cache (b) cache, main and secondary
(c) main, cache and secondary (d) cache, secondary and main.

(xxviii) The hit ratio for cache memories is in ascending order for
(a) associative, direct and set-associative (b) direct, associative and set-associative
(c) set-associative, direct, associative (d) set-associative, associative, direct.

(xxix) Paging is
(a) non-contiguous memory allocation method
(b) implementation of virtual memory
(c) contiguous memory allocation method
(d) both (a) and (b).

(xxx) Size of virtual memory is equivalent to the size of
(a) main memory (b) secondary memory
(c) cache memory (d) totality of (a) and (b).

(xxxi) Code sharing is possible in
(a) paging (b) segmentation (c) both (a) and (b) (d) none.

(xxxii) A page fault
(a) occurs when a program accesses a main memory
(b) is an error in a specific page
(c) is an access to a page not currently residing in main memory
(d) is a reference to a page residing in another page.

(xxxiii) The user view of memory is supported by
(a) paging (b) segmentation (c) both (d) none.

(xxxiv) Associative memory is
(a) very cheap memory (b) pointer addressable memory
(c) content addressable memory (d) slow memory.

(xxxv) Cache memory is made up of
(a) CMOS RAM (b) bipolar RAM (c) magnetic disc (d) optical disc.

(xxxvi) A 20-bit address bus allows access to a memory of capacity
(a) 1 MB (b) 2 MB (c) 32 MB (d) 64 MB.

(xxxvii) The largest delay in accessing data on disk is due to
(a) seek time (b) rotation time
(c) data transfer time (d) none.

(xxxviii) Cache memory refers to
(a) cheap memory that can be plugged into the mother board to expand main memory
(b) fast memory present on the processor chip that is used to store recently accessed data
(c) a reserved portion of main memory used to save important data
(d) a special area of memory on the chip that is used to save frequently used constants

(xxxix) A major advantage of direct mapping of a cache is its simplicity. The main disadvantage of this organization is that
(a) it does not allow simultaneous access to the intended data and its tag
(b) it is more expensive than other types of cache organizations
(c) the cache hit ratio is degraded if two or more blocks used alternately map onto the same block frame in the cache
(d) its access time is greater than that of other cache organizations
(e) the number of blocks required for the cache increases linearly with the size of the main memory

(xxxx) If a cache access requires one clock cycle and handling cache misses or stalls the processor for an additional five cycles, which of the following cache hit rates comes closest to achieving an average memory access of 2 cycles?
(a) 75 (b) 80 (c) 83 (d) 86 (e) 98

(xxxxi) A computer that is advertised as having a 96 K byte DRAM memory and a 2.1 gigabyte hard drive has
(a) 96 K bytes of primary memory and 2.1 gigabytes of secondary memory
(b) 2.1 gigabytes of primary memory and 96 K bytes of secondary memory
(c) 96 bytes of cache, 2.1 gigabytes of primary memory
(d) 96 K bytes of cache, 96 K bytes of primary memory, and 2.1 gigabytes of secondary memory

(xxxxii) A memory management technique used to improve computer performance is
(a) selecting memory chips based on their cost
(b) storing as much data as possible on disk
(c) using the cache to store data that will most likely be needed soon
(d) preventing data from being moved from the cache to primary memory.

(xxxxiii) Access time is
(a) time taken to request and complete the requested data
(b) same as the latency time
(c) both (a) and (b) are correct
(d) both (a) and (b) are incorrect.

(xxxxiv) If a page fault occurs then the contents of cache memory search the information in
(a) secondary memory (b) main memory
(c) cache memory (d) virtual memory.

(xxxxv) Partition of memory in fixed size is
(a) segmentation (b) paging
(c) both segmentation and paging (d) neither segmentation nor paging.

Group B

2. What do you mean by capacity, access time, cycle time and bandwidth of memory?
3. Why is the memory system of a computer organized as a hierarchy? What are the basic elements of a memory hierarchy?
4. Describe different access methods of the memory system.
5. What are two major elements of the main memory? Explain each of them.
6. Compare and contrast SRAM and DRAM.
7. What will be the maximum capacity of a memory, which uses an address bus of size 8-bit?
8. Show the bus connections with a CPU to connect 4 RAM chips of size 256×8 bit each and a ROM chip of 512×8 bit size. Assume the CPU has 8 bit data bus and 16 bit address bus. Clearly specify generation of chip select signals. Give the memory address map for the system.
9. Suppose we are given RAM chips each of size 256×4. Design a $2\text{ K} \times 8$ RAM system using this chip as the building block. Draw a neat logic diagram of your implementation.
10. Briefly describe the storage structure of a CMOS SRAM storage cell and explain the read and write operations on the cell. Give the suitable diagram.
11. Draw the cell structure of DRAM and explain the read and write operations on it.
12. Briefly describe different types of RAMs.
13. Draw and describe the storage structure of a ROM storage cell and explain the read and write operations on the cell.
14. Briefly describe different types of ROMs.
15. Explain the need of auxiliary (secondary) memory devices. How are they different from main memory?

16. What are the different types of secondary memories?
17. Give at least two differences between magnetic tape and magnetic disk.
18. Discuss in brief the internal structure of magnetic tape using diagram. Explain how read and write operations are performed on it.
19. Draw and describe the internal structure of a disk-pack.
20. Define:
    (a) Sector
    (b) Track
    (c) Cylinder
    (d) Rotational latency
    (e) Seek time
    (f) Access time in magnetic disk.
21. Why the formatting of disk is necessary before working on that disk?
22. What is the transfer rate of an 8-track magnetic tape whose speed is 120 inches per second and whose density is 1600 bits per inch?
23. A disk pack has 20 recording surfaces and has a total of 4000 cylinders. There is an average of 300 sectors per track. Each sector contains 512 bytes of data.
    (a) What is the maximum number of bytes that can be stored in this pack?
    (b) What is the data transfer rate in bytes per second at a rotational speed of 3600 rpm?
    (c) Using a 32-bit word, suggest a suitable scheme for specifying the disk address, assuming that there are 256 bytes per sector.
24. A disk pack has the following parameters:
    (a) average time to position the magnetic head over a track is 20 ns
    (b) rotational speed of 2400 rpm.
    (c) number of bits per track is 20000
    (d) number of bits per sector is 1500

    Calculate the average time to read one sector.
25. What is CAM? What is its main characteristic? What is its main advantage? Draw and describe hardware structure of associative memory or CAM. Also draw its one bit cell structure.
26. The cache memory is designed based on one property of programs stored memory. What is that property? Explain. What are different varieties of this property?
27. What is cache memory? How does cache memory increase the performance of a computer? What is hit ratio?
28. Describe the operation of cache memory. What is meant by "the cache memory has hit ratio of 0.8"?
29. Derive an expression for effective (average) access time for an n-level memory system having hit ratios $h_1, h_2, \ldots h_n$ and access times $t_1, t_2, \ldots t_n$,where $t_1 < t_2 < \ldots t_n$.
30. A three level memory system having cache access time of 10 nsec and disk access time of 40 nsec has a cache hit ratio of 0.96 and main memory hit ratio of 0.9. What should be the main memory access time to achieve effective access time of 10 nsec?
31. What is meant by cache mapping? What are different types of mapping? Describe different mapping techniques with suitable example.
32. Compare different cache mapping techniques.
33. What are different replacement methods in cache memories? Explain.
34. Name different cache memory writing methods. Explain them with advantages and disadvantages.
35. Name the categories of cache misses and explain the reasons behind them.
36. Write three methods for reducing cache miss rates.

37. What is virtual memory? What are the reasons for using it?
38. What do you mean by logical address space and physical address space?
39. Explain the paging and segmentation techniques used for implementing virtual memory.
40. A digital computer has a memory unit of 64 K × 16 and a cache memory of 1 K words. The cache uses direct mapping with a block size of 4 words. How many bits are there in the tag, index, block and word fields of the address format?
41. What is the limitation of direct-mapped cache? Explain with an example how it can be improved in set-associative cache.
42. Compare paging and segmentation methods.
43. What is page fault? What are different page replacement algorithms? Describe briefly.
44. Consider a cache (M_1) and (M_2) hierarchy with the following characteristics;
    M_1: 16 k words, 50 ns access time;
    M_2: 1 M words, 400 ns access time;
    Assume 8 word cache blocks and a set size of 256 words with set-associative mapping.
    (a) Show the mapping between M_2 and M_1.
    (b) Calculate the effective memory-access time with a cache-hit ratio of h = 0.95.
45. Consider the design of 3-level memory hierarchy with following specification for memory hierarchy.

Memory levels	Access time	Capacity	Cost/KB
Cache	25ns	512 KB	$1.25
Main memory	unknown	32 MB	$0.2
Disk unit	4 ms	unknown	$0.0002

    The design goal is to achieve effective memory access time t = 10.04 microsecond with cache-hit ratio h1 = 0.98 and h2 = 0.9 in the main memory unit. Also the total cost of the memory hierarchy is upper-bounded by $1500. Find out the unknown terms in the above table.
46. A system has 48 bit virtual address, 36 bit physical address and 128 MB of main memory address. The system has 4096 bytes of pages. How many virtual and physical pages can the address support? How many page frames of main memory are there?
47. What do you mean by page reference string? Suppose a process accesses the following addresses at a particular time interval: 0100,0432,0101,0612,0102,0103,0104,0101,0611,0102,0103,0104,0101, 0601,0101,0102,0609,0102,0105. Assume a page size=100 bytes.
    What will be the reference string? Using this reference string, calculate the page fault rate for the following algorithms:
    (a) FIFO replacement.
    (b) LRU replacement.
    (c) Optimal replacement.
    Assume that number of frames = 3.
48. If the size of cache block is increased, discuss possible merits and demerits.
49. Which locality of reference is exploited if the size of cache block increases?
50. In order to exploit temporal locality of reference, what should be the policy of writing program?
51. How does the size of cache block affect the hit ratio?
52. If the size of MAR and MDR are 32-bit and 16-bit respectively, what is the size of main memory?

CHAPTER

5
Computer Instruction Set

5.1 INTRODUCTION

Computer architecture is defined as the study of the components and their interconnections that form a computer system. An instruction set design for a machine is the primary architectural consideration. The complete collection of instructions that are understood by a machine is called the *instruction set* for the machine. Much of a computer system's architecture is hidden from a high level language programmer. In the abstract sense, the programmer should not really care about what is the underlying architecture. The instruction set is the boundary where the computer designer and the computer programmer can view the same machine. Any weakness of the instruction set design will drastically affect the machine language programmer and the compiler. Hence, a well-designed, pre-planned instruction set enables the compilers to create a compact machine language program. This chapter discusses the various factors that influence the instruction set design.

5.2 INSTRUCTION SET DESIGN

The complete collection of instructions that are understood by a machine (processor) is called the *instruction set* for the machine. As pointed out earlier, a well-designed, pre-planned instruction set enables the compilers to create a compact machine language program. Thus, it is a very important task for computer architects to design a good instruction set for the concerned machine.

Before finalizing the instruction set for a computer, computer architects have to consider the following aspects.

Flexibility to the programmer A programmer wishes to have as many instructions as possible so that the appropriate operations are carried out by the respective instructions. While designing the instruction set, it should be noted that too many instructions in the instruction set results in a complex control unit design. Instruction decoding requires huge circuitry and time.

Number of addressing modes If all possible addressing modes are present in the architecture, it will give a lot of options for programming a particular program. However, it will again require complex control unit.

Number of general purpose registers (GPRs) If the CPU has a large number of GPRs, the execution will be faster. But, use of large number of registers increases cost of the CPU.

System performance The system performance can be enhanced, if less number of instructions is used in a program. For short programs, instructions should be powerful. Thus, a single instruction must be able to perform several microoperations (i.e, large length instructions are used). So, reduced size program is desired. But, this increases the complexity in the control unit design and instruction execution time.

Applications An instruction set for a particular computer is designed in aiming the certain application area. For example, a scientific computer must have strong floating-point arithmetic without which the precision would be heavily degraded. Whereas, an entertainment computer must have multimedia operations.

5.3 INSTRUCTION FORMATS

In broad sense, the superiority of a computer is decided on the basis of its instruction set. Since, the total number of instructions and their powerfulness has contributed to the efficiency of the computer, these two factors are given highest priority. An efficient program is the one which is short, hence fast execution and occupies less memory space. The size of a program depends largely on the formats of instructions used.

A computer usually has a variety of instruction formats. It is the task of the control unit within CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction. The most common format followed by instructions is depicted in the Fig. 5.1.

Figure 5.1 *Different fields of instructions*

The bits of the instruction are divided into groups called fields. The commonly used fields found in instruction formats are:

1. *Operation Code* (or, simply *Op-code*): This field states the operation to be performed. This field defines various processor operations, such as add, subtract, complement, etc.
2. *Address:* An address field designates a memory address or a processor register or an operand value.
3. *Mode*: This field specifies the method to get the operand or effective address of operand. In some computers' instruction set, the op-code itself explicitly specifies the addressing mode used in the instruction. A computer has various addressing modes, which are presented in the Section 5.8.

For example, in the instruction ADD R1, R0; ADD is the op-code to indicate the addition operation and R1, R0 are the address fields for operands.

In certain situations, other special fields are sometimes used. For example, a field that gives the number of shifts in a shift-type micro-operation or, a label field is used to process unconditional branch instruction.

Since the number of address fields is the primary concern of an instruction format, we check the effect of including multiple address fields in an instruction in the Section 5.5.

The memory or processor registers store the operand values on which operation codes specified by computer instructions are executed. Memory addresses are used to specify operands stored in memory. A register address specifies an operand stored in processor register. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 32 processor registers R0 to R31 has a register address field of 5 bits. For example, processor register R7 is specified by the binary number 00111. The internal organization of processor registers determines the number of address fields in the instruction.

5.4 CPU ORGANIZATION

The design of an instruction set for a computer depends on the way in which the CPU is organized. Generally, there are three different CPU organizations with certain specific instructions:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

5.4.1 Single Accumulator Based CPU Organization

In early days of computer history, computers had accumulator based CPUs. It is a simple CPU, in which the accumulator register is used implicitly for processing all instructions of a program and intermediate results are stored into this register. The instruction format in this computer uses one address field. For this the CPU is known as one address machine. For example, the instruction arithmetic multiplication defined by an assembly language instruction uses one address field and is written as

MULT X

where X is the address of the operand. The MULT instruction in this example performs the operation AC ← AC * M[X]. AC is the accumulator register and M[X] denotes the memory word (operand) located at location X. The organization of an accumulator-based processor is shown in Figure 5.2.

The program counter (PC) register holds address of the next instruction to be executed. The memory address register (MAR) holds the address of the data to be accessed in memory. The stack pointer (SP) is a special register used to hold the address of the top element of the stack. The instruction register (IR) holds the instruction currently being executed. The status register (SR) indicates different status flags such as zero flag (Z-flag) and carry flag (C-flag).

This type of CPU organization is first used in PDP-8 processor and is used for process control and laboratory applications. This type of CPU organization has been totally replaced by the introduction of the new general register based CPU.

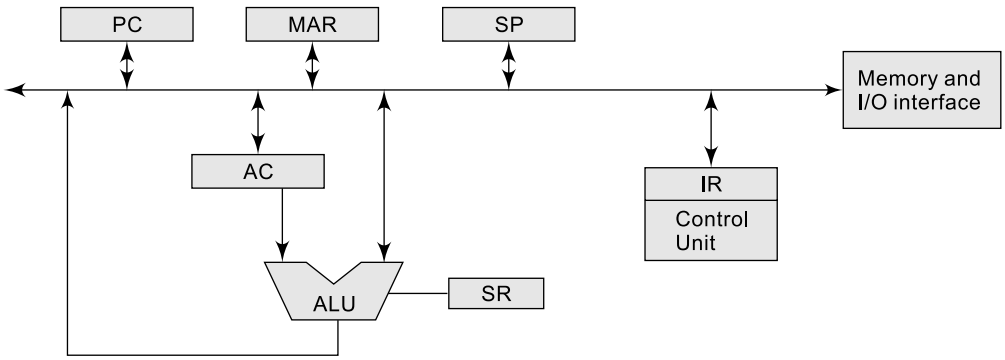


Figure 5.2 *Accumulator-based CPU organization*

Advantages

1. One of the operands is always held by the accumulator register. This results in short instructions and less memory space.
2. Instruction cycle takes less time because it saves time in instruction fetching from memory.

Disadvantages

1. When complex expressions are computed, program size increases due to the usage of many short instructions to execute it. Thus memory size increases.
2. As the number of instructions increases for a program, the execution time increases.

5.4.2 General Register Based CPU Organization

Instead of a single accumulator register, multiple general registers are used in this type of CPU organization. This type computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word. For example, an arithmetic multiplication written in an assembly language uses three address fields and is written as

MULT R1, R2, R3

The meaning of the operation is R1 ← R2 * R3. This instruction also can be written in the following way, where the destination register is the same as one of the source registers.

MULT R1, R2

This means the operation R1 ← R1 * R2, which uses two address fields. The use of large number of registers results in short programs with limited instructions. The organization of a general register-based processor is shown in Figure 5.3. The organization of a general register-based IBM 360 and PDP-11 are some of the typical examples.

The *advantages* of this organization:

1. Since large number of registers is used in this organization, the efficiency of the CPU increases.
2. Less memory space is used to store the program since the instructions are used in more compact way.

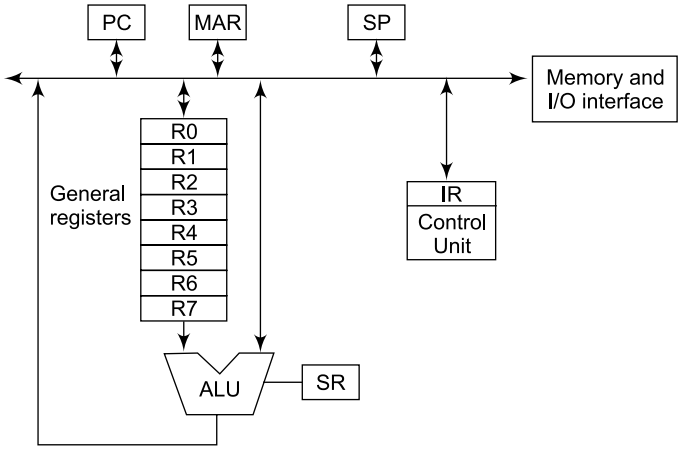


Figure 5.3 *Typical general-register based CPU organization*

The *disadvantages* of this organization:

1. Care should be taken to avoid unnecessary usage of registers. Thus compilers need to be more intelligent in this aspect.
2. This organization involves more cost, since large number of registers is used.

5.4.3 Stack Based CPU Organization

Stack based computer operates instructions, based on a data structure called stack. A stack is a list of data words with a Last-In, First-Out (LIFO) access method that is included in the CPU of most computers. A portion of memory unit used to store operands in successive locations can be considered as a stack in computers. The register that holds the address for the top most operand in the stack is called a stack pointer (SP). The two operations performed on the operands stored in a stack are the *PUSH* and *POP*. From one end only, operands are pushed or popped. The PUSH operation results in inserting one operand at the top of stack and it decreases the stack pointer register. The POP operation results in deleting one operand from the top of stack and it increases the stack pointer register.

For example, Fig. 5.4 shows a stack of four data words in the memory. PUSH and POP instructions require an address field. The PUSH instruction has the format:

PUSH <memory address>

The PUSH instruction inserts the data word at specified address to the top of the stack. The POP instruction has the format:

POP <memory address>

The POP instruction deletes the data word at the top of the stack to the specified address. The stack pointer is updated automatically in either case. The PUSH operation can be implemented as

SP ← SP – 1 ; decrement the SP by 1
SP ← <memory address> ; store the content of specified memory address into SP, i.e. at top of stack

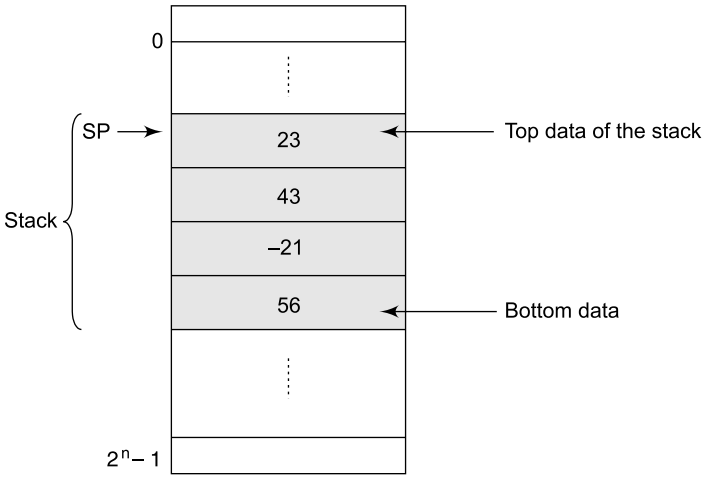


Figure 5.4 *A stack of words in memory*

The POP operation can be implemented as

<memory address> ← SP ; transfer the content of SP (i.e. top most data)into specified memory location

SP ← SP + 1 ; increment the SP by 1

The Fig. 5.5 shows the effects of these two operations on the stack in Fig. 5.4.

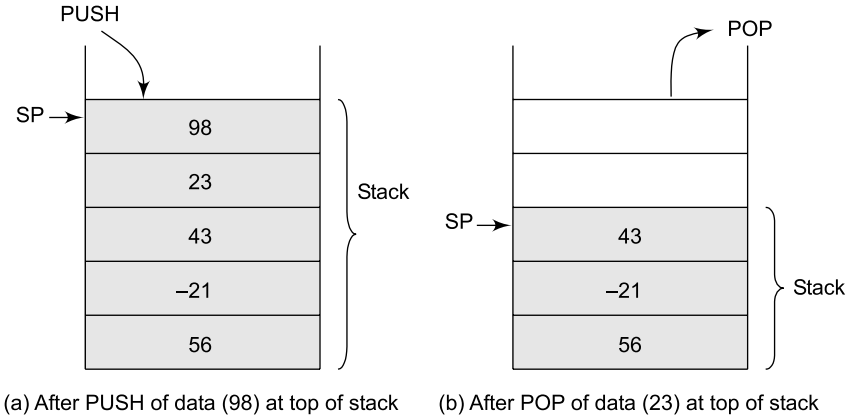


Figure 5.5 *Effects of stack operations on the stack in Fig. 5.4*

Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two operands that are on top of the stack. For example, the instruction

SUB

in a stack computer consists of an operation code only with no address field. This operation pops the two top data from the stack, subtracting the data, and pushing the result into the stack at the top. The organization of a stack-based machine is shown in Fig. 5.6.

PDP-11, Intel's 8085 and HP 3000 are some of the examples of stack organized computers.

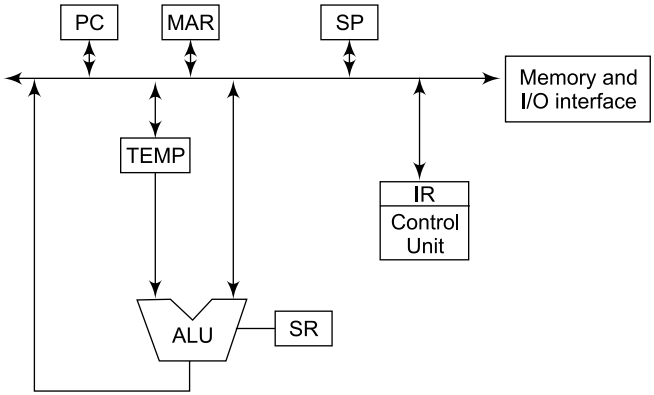


Figure 5.6 *Typical stack-based CPU organization*

Advantages

1. Efficient computation of complex arithmetic expressions.
2. Execution of instructions is fast, because operand data are stored in consecutive memory locations.
3. Since instructions do not have address field, the length of instructions is short.

Disadvantage

1. Program size lengthens.

One of the three types of organizations that have just been described has been implemented in most contemporary computers. Though, some computers have been built with features from more than one camp. For example, the Intel 8080 microprocessor has seven general-purpose registers, one of which is an accumulator register. Thus, some of the characteristics of a general register organization and some of the characteristics of an accumulator organization are followed by the processor 8080.

5.4.4 Arithmetic Expression Evaluation

The stack organization is very effective for evaluating arithmetic expressions. Expressions are usually represented in what is known as *infix notation*, in which each operator is written between two operands (e.g., $A + B$). With this notation, we must distinguish between $(A + B) * C$ and $A + (B * C)$ by using either parentheses or some operator-precedence convention. Thus, the order of the operators and operands in an arithmetic expression does not uniquely determine the order in which operations are to be performed.

Polish notation (also known as prefix notation), named after the Polish mathematician Jan Lukasiewicz, refers to the notation in which the operator is placed before its two operands (e.g., $+ AB$). Regardless of the complexity of an expression, no parentheses are required when using Polish notation.

Reverse Polish notation (RPN) (also known as postfix notation) refers to the analogous notation in which the operator is placed after its two operands (e.g., $AB +$). Again, regardless of the complexity of an expression, no parentheses are required when using reverse Polish notation.

Stack-organized computers are better suited to postfix (RPN) notation than traditional infix notation. Thus the infix notation must be converted to postfix notation (RPN). The conversion from infix

notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infix notation. Conventionally, three levels of precedence for the usual five binary operators as:

Highest: Exponentiation (^)
Next highest: Multiplication (*) and division (/)
Lowest: Addition (+) and subtraction (–)

Consider the expression

$$(A - B)*[C/(D + E) + F]$$

To evaluate the expression we must first perform the arithmetic inside the parentheses $(A - B)$ and $(D + E)$. Next we must calculate the expression inside the square brackets. The division of $C/(D + E)$ must be done prior to the addition of F since division has precedence over addition. The last operation is the multiplication of the two terms between the parentheses and brackets. The expression can be converted to reverse Polish notation, without the use of parentheses, by taking into consideration the operation hierarchy. The converted expression is

$$AB - CDE + /F+*$$

Now we want to calculate the value of an arithmetic expression by using a stack. The procedure consists of first converting the arithmetic expression into its equivalent reverse Polish notation. The operands are pushed into the stack in the order in which they appear. The following microoperations are executed with the stack when an operation is encountered: (1) the topmost operands in the stack are used for the operation, and (2) the stack is popped and the result of the operation replaces the lower operand. By pushing the operands into the stack continuously and performing the operations as defined above, the expression is evaluated in the proper order and the final result remains on top of the stack.

For illustration, consider the following arithmetic expression:

$$(2 + 4) * (4 + 6)$$

In reverse Polish notation, it is expressed as

$$2\ 4 + 4\ 6 + *$$

The stack operations are shown in Fig. 5.7 for this expression evaluation.

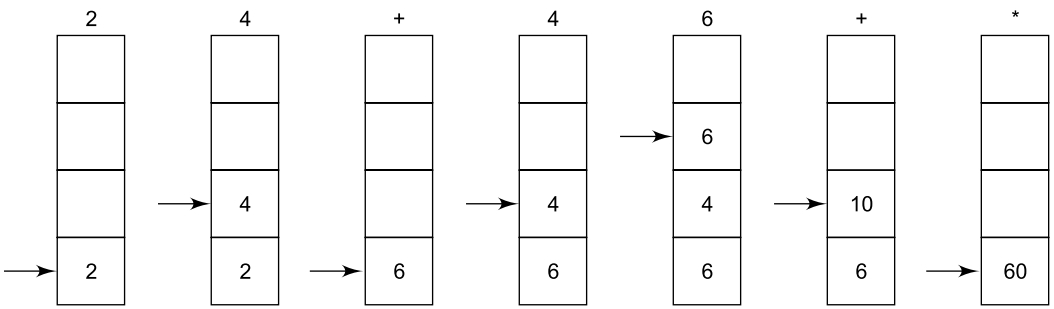


Figure 5.7 *Stack operations to evaluate (2 + 4) * (4 + 6)*

5.4.5 Subroutines

A subroutine is a program segment for carrying out repeatedly needed tasks such as searching, and sorting. A subroutine may be written and tested separately. A subroutine can be linked with a user program so that the latter can call the former as many times as needed. Thus the use of subroutines

can save a programmer's time as well as the memory space needed by an application program. A large program can be thought of as a collection of independent program modules, where each module may be a subroutine or a set of subroutines. This is the key feature of the modern software approach called *modular programming.*

Subroutine calls and returns from subroutines are usually handled by two special instructions, CALL and RET, respectively. The CALL instruction is of the form CALL SUB, where the parameter SUB refers to the address of the first instruction of the subroutine. When this instruction is executed, the current contents of the PC (Program Counter) register are saved in the stack, and the PC is loaded with <SUB> The current content of the PC provides the address of the instruction that immediately follows the CALL instruction. This address is also called the *return address* because this is the point where execution of the calling program will take place after exiting from the subroutine. The CALL instruction is functionally equivalent to the following instruction sequence:

```
PUSH PC        ; save the return address in the stack (SP)
JP SUB         ; branch to the subroutine
```

The RET instruction is usually the last instruction of the subroutine. When this instruction is executed, the return address previously saved in the stack is retrieved and loaded into the PC. The control is then transferred to the calling program. A RET instruction is functionally equivalent to:

```
POP PC         ; PC ← (SP)
```

Here we might have the natural question: why the return address is not saved in a CPU register rather than the stack. This arrangement fails to work if nested subroutine calls are to be implemented. Subroutine nesting refers to one subroutine calling another.

For example, consider the main program M and two subroutines SUB_1 and SUB_2 shown in Fig. 5.8 (a) The main program calls subroutine SUB_1, and this subroutine in turn calls subroutine SUB_2. Parameters RM and $RSUB_1$ refer to the return addresses of the main program M and the subroutine SUB_1 respectively. When the main program calls subroutine SUB_1, the return address RM is pushed into the stack (see Figure 5.8(b)) and the control is transferred to subroutine SUB_1. Similarly, when subroutine SUB_1 calls subroutine SUB_2, the return address $RSUB_1$ is pushed into the stack (see Figure 5.8(c)), and the control is transferred to subroutine SUB_2. When subroutine SUB_2 completes its execution, the return address is retrieved from the stack and loaded into the PC. Since the return address is $RSUB_1$, the execution of subroutine SUB_1 is resumed. Similarly, when subroutine SUB_1 terminates, the return address RM (see Figure 5.8(d)) is retrieved from the stack and loaded into the PC. The execution of the main program is then resumed.

To implement subroutine nesting, the return addresses must be retrieved exactly in the reverse order in which they are saved. Since a stack is a LIFO data structure, its use is a natural solution to this problem. Suppose a CPU register is used to save the return address. The return address $RSUB_1$ will overwrite the return address RM, and control will not be transferred back to the main program at all.

5.5 INSTRUCTION LENGTH

Length of an instruction basically depends on the number of address fields used in it.

The advantages and disadvantages of using number of addresses in an instruction are summarized on next page:

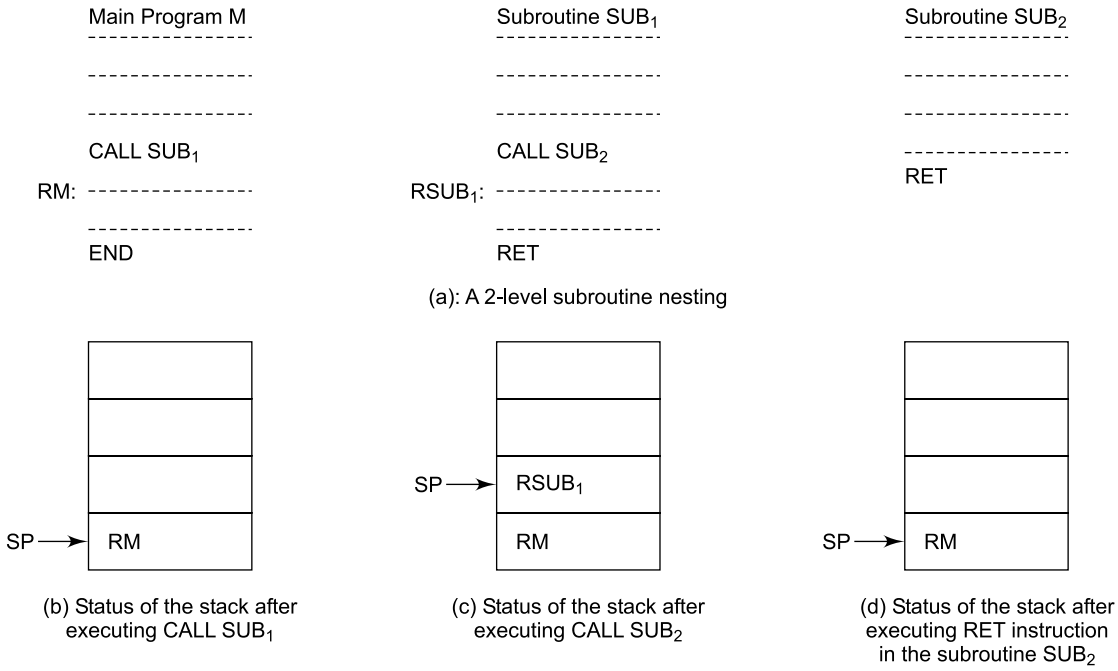


Figure 5.8 *Implementation of a 2-level subroutine nesting*

- The fewer the addresses, the shorter the instruction. Long instructions with multiple addresses usually require more complex decoding and processing circuits.
- Limiting the number of addresses also limits the range of functions each instruction can perform.
- Fewer addresses means more primitive instructions, and longer programs are needed.
- Storage requirements of shorter instructions and longer programs tend to balance; larger programs require longer execution time.

The length of an instruction can be affected by and affects:

- Memory size
- Memory organization
- Bus structure
- CPU complexity
- CPU speed

To show how the number of addresses affects a computer program, we will evaluate the arithmetic statement

$$X = (A + B) - (C + D)$$

using zero, one, two or three address instructions. For this, LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and SUB are used for the arithmetic operations addition and subtraction respectively. Assume that the respective operands are in memory addresses A, B, C and D and the result must be stored in the memory at address X.

Three-address Instructions The general register organized computers use three-address instructions. Each address field may specify either a processor register or a memory operand. The program to evaluate X = (A + B) – (C + D) in assembly language is shown below, together with comments that give explanation of each instruction.

```
ADD R1, A, B        ; R1← M[A] + M[B]
ADD R2, C, D        ; R2 ← M[C] + M[D]
SUB X, R1, R2       ; X ← R1 – R2
```

The advantage of three-address format is that it generates short programs. The disadvantage is that it uses long instructions.

Two-address Instructions The most popular instructions in commercial computers are two-address instructions. The general register organized computers use two-address instructions as well. Like three-address instructions, each address field may specify either a processor register or a memory operand. The assembly program using two-address instructions to evaluate X = (A + B) – (C + D) is as follows:

```
LOAD R1, A          ; R1← M[A]
ADD R1, B           ; R1 ← R1 + M[B]
LOAD R2, C          ; R2 ← M[C]
ADD R2, D           ; R2 ← R2 + M[D]
SUB R1, R2          ; R1 ← R1 – R2
STORE X, R1         ; X ← R1
```

One-address Instructions The single accumulator based computers use one-address instructions. Here, all instructions use an implied accumulator (AC) register. The program to evaluate X = (A + B) – (C + D) using one-address instructions is as follows:

```
LOAD C              ; AC ← M[C]
ADD D               ; AC ← AC + M[D]
STORE T             ; T ← AC
LOAD A              ; AC ← M[A]
ADD B               ; AC ← AC + M[B]
SUB T               ; AC ← AC – M[T]
STORE X             ; X ← AC
```

T is the temporary memory location required for storing the intermediate result.

Zero-address Instructions Zero-address instructions are used by stack-organized computers, which do not use any address field for the operation-type instructions. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions. However, two basic instructions in stack PUSH and POP require an address field to specify the destination or source of operand. The assembly language program using zero-address instructions is written next. In the comment field, the symbol TOS is used, which means the top of stack.

```
PUSH A              ; TOS ← A
```

```
PUSH B          ; TOS ← B
ADD             ; TOS ← (A + B)
PUSH C          ; TOS ← C
PUSH D          ; TOS ← D
ADD             ; TOS ← (C + D)
SUB             ; TOS ← (A + B) – (C + D)
POP X           ; X ← TOS
```

5.6 DATA ORDERING AND ADDRESSING STANDARDS

There are two different schemes followed for positioning of data words in memory and addressing: *Big-endian* assignment and *Little-endian* assignment. Suppose we have 32-bit data word 642CD09A $_{HEX}$ to be stored in memory from address 0 onwards. Since there are 4 bytes in each word, the word occupies addresses 0 to 3, if the memory is byte-addressable (i.e. successive addresses refer to successive byte locations in the memory). In big-endian assignment, the most significant byte is stored in lower address and least significant byte is stored in higher address. The little-endian assignment is used for opposite ordering. That means, the least significant byte is stored in lower address and the most significant byte is stored in higher address in little-endian scheme. The methods are depicted in the Fig. 5.9 (assuming word length of the machine = 32 bits). In figure, the number in each box indicates the byte address of the data word. Thus, the byte arrangements and memory addresses are as follows:

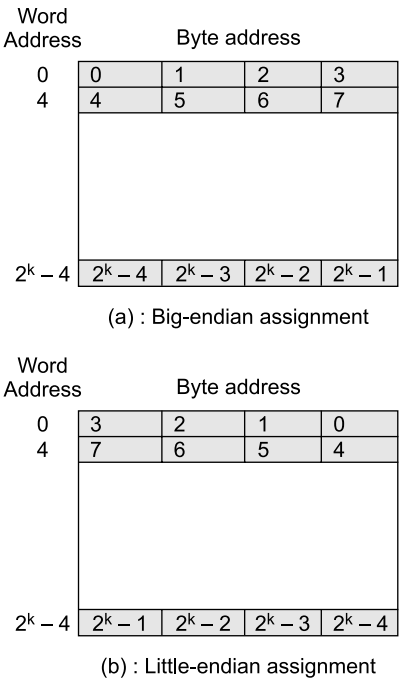


Figure 5.9 *Byte and word addressing (assuming word length = 32 bits)*

In big-endian (address [data]): 0 [64], 1 [2C], 2 [D0], 3 [9A].

In little-endian (address [data]): 0 [9A], 1 [D0], 2 [2C], 3 [64].

Some computers use only one method, whereas some commercial computers use both.

5.7 INSTRUCTION CYCLE

As pointed out in Chapter 1, the processing required for each instruction of a program is called *instruction cycle*. The control unit's task is to go through an instruction cycle (see Figure 5.10) that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

Step 1 is basically performed using a special register in the CPU called *program counter* (PC) that holds the address of the next instruction to be executed. If the current instruction is simple arithmetic/logic or load/store type the PC is automatically incremented. Otherwise, PC is loaded with the address dictated by the currently executing instruction. The decoding done in Step 2 determines the operation to be performed and the addressing mode of the instruction for calculation of address of operands. After getting the information about the addresses of operands, the CPU fetches the operands in Step 3 from memory or registers and stores them its registers. In step 4, the ALU of processor executes the instruction on the stored operands in registers. After the execution of instruction, in phase 5 the result is stored back in memory or register and returns to step 1 to fetch the next instruction in sequence. All these sub-operations are controlled and synchronized by the control unit.

5.7.1 Instruction Fetch

An instruction is generally fetched from a memory location specified by a register called program counter (PC). It keeps track of the address of instruction in memory that has to be executed next. The instruction is executed sequentially unless an instruction changes the content of the PC. The instruction register holds the instruction until it is decoded. The instruction decoder analyzes the bit pattern and then determines the next action to be performed by the control unit. The PC is incremented every time, an instruction is fetched. But when a jump (branch) instruction is executed, i.e. program calls another program known as *subroutine*, then the content of the PC is replaced by the address given in jump location. In this operation, the processor is required to remember the contents of the program counter at the time of jump enabling processor to resume the task of execution of main program when it has finished the last instruction of the subroutine. When the processor receives the call instruction of the subroutine, it increments the content of the PC and stores the content in a specified reserved memory area known as *stack***.** The address of the instruction of the main program to be executed after completion of the subroutine is stored in the stack. Many processors reserve stack area inside the processor itself. This type of stack is known as *internal stack*. Other processors have stack area somewhere in the memory and have a pointer pointed to the stack top. The stack pointer (SP) register holds the address of the stack-top. The stack is a sequence of memory locations defined by the programmer. The stack is used to save the content of a register during the execution of a program.

5.7.2 Instruction Decode and Execution

The content of the PC is transferred to the special register known as address buffer or memory address register (MAR). The content of the MAR is transferred to the memory through the address bus. By sending certain control signals to the memory the CPU also indicates that it wants to read the contents of the memory. The decoder circuitry in the memory is activated and the memory understands what is to be done. Then the memory sends op-code to the CPU through the data bus. The op-code first comes in a data buffer or data register. The operation code is then placed in the

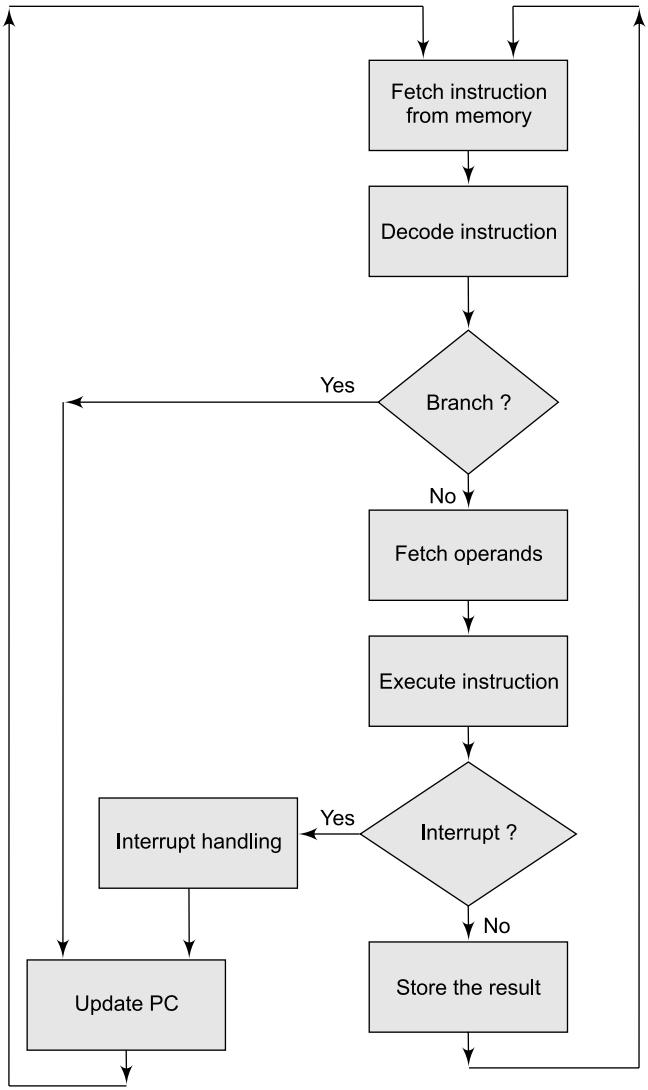


Figure 5.10 *Instruction cycle*

instruction register (IR). The instruction decoder decodes the instruction and resolves the addressing mode used in the instruction for knowing the addresses of operands. Then the instruction is executed by the ALU of CPU. Finally, the content of the PC is incremented. The execution of an instruction requires the flow of data in most of the instructions. A data is received either from the memory or input devices. The data word flows to the processor through the data bus and is placed in accumulator (AC) or any other general-purpose register depending upon the instruction. After the execution of an instruction, the data is placed in a register or a memory location. After the execution of a program, the result (data) is placed in the memory or sent to an output device. When a data word is written into the memory, it is also held in data buffer until the write operation is completed.

5.7.3 Interrupt Cycle

To process interrupts, an interrupt cycle is added to the instruction cycle (which consists of two major cycles: fetch and execute, here), as shown in Figure 5.11. In the interrupt cycle, the CPU checks to verify if any interrupts have occurred, indicated by the presence of an interrupt signal. If no interrupts are pending, the CPU proceeds to the fetch cycle and fetches the next instruction of the current program as usual. If an interrupt is pending, the CPU does the following:

1. It suspends execution of the current program being executed.
2. It saves the context of the current program being executed. This means saving the address of the next instruction to be executed and any other data relevant to current activity of the CPU.
3. It sets PC (program counter) to start address of interrupt handler (i.e. interrupt service routine) to service the interrupt.
4. Then process interrupt.
5. After the completion of interrupt handler, the CPU resumes execution of the interrupted program.

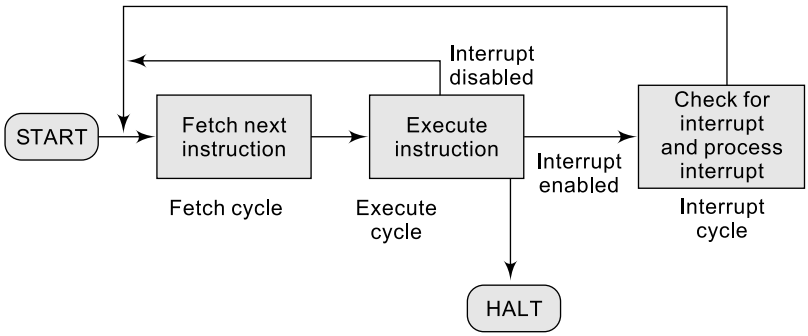


Figure 5.11 *Instruction cycle with interrupts*

Interrupts are not always handled immediately, the CPU has authority to disable all or selected interrupt signals and subsequently enable them. A disabled interrupt simply means that the CPU can and will ignore that interrupt request signal.

For example, it is generally desirable to finish the processing of one interrupt before taking on another. Thus, interrupts are often disabled while the CPU is processing an interrupt. If an interrupt occurs during this time, it generally remains pending and will be checked by the CPU after the CPU has enabled interrupt. A simple flowchart of interrupt cycle is shown in Figure 5.12.

5.8 ADDRESSING MODES

The datapath (specifically ALU) of the CPU executes the instructions as dictated by the op-code field of instructions. The instructions are executed on some data stored in registers or memory. The different ways in which the location of an operand is specified in an instruction are referred to as *addressing modes*. A computer uses variety of addressing modes.

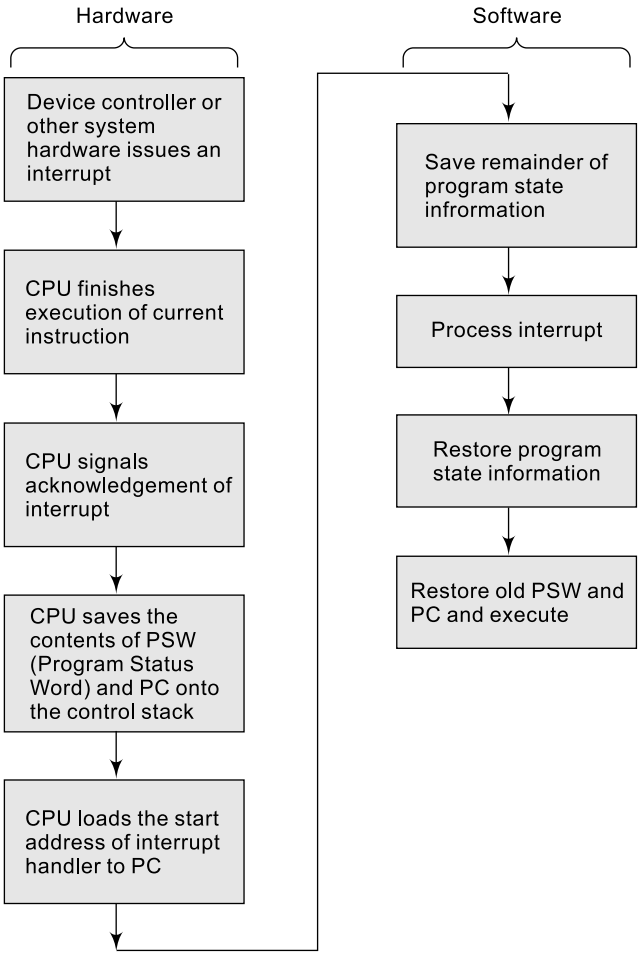


Figure 5.12 *Flowchart of a simple interrupt cycle*

The advantages of having different addressing modes:
Computers use different addressing modes for the following purposes:

1. It gives programming versatility or flexibility to the programmers with respect to the number of instructions and execution time by providing various addressing modes.
2. To reduce the length of the instructions or the size of programs. Because these two parameters are associated with the capacity of memory.

A computer generally has variety of addressing modes. Sometimes, two or more addressing modes are combined in one mode. The popular addressing modes are discussed next.

Class I Here, no address field is used in instruction.

1. Implied (or Inherent) mode In this mode the operands are indicated implicitly by the instruction. The accumulator register is generally used to hold the operand and after the instruction execution the result is stored in the same register. For example,

(a) RAL; Rotates the content of the accumulator left through carry.
(b) CMA; Takes complement of the content of the accumulator.
This mode is very popular with 8-bit micro-processors such as the Intel's 8085.

2. Immediate mode In this mode the operand is mentioned explicitly in the instruction. In other words, an immediate-mode instruction contains an operand value rather than an address of it in the address field. To initialize register to a constant value, this mode of instructions is useful. For example:

(a) MVI A, 06; Loads equivalent binary value of 06 to the accumulator.
(b) ADI 05; Adds the equivalent binary value of 05 to the content of AC.

3. Stack addressing mode Stack-organized computers use stack addressed instructions. In this addressing mode, all the operands for an instruction are taken from the top of the stack. The instruction does not have any operand field. For example, the instruction

SUB

uses only one op-code (SUB) field, no address field. Both the operands are in the topmost two positions in the stack, in consecutive locations. When the SUB instruction is executed, two operands are popped out automatically from the stack one-by-one. After subtraction, the result is pushed onto the stack. Since no address field is used, the instruction is short.

Class II Here, address field is register address.

4. Register (direct) mode In this mode the processor registers hold the operands. In other words, the address field is now register field, which contains the operands required for the instruction. A particular register is selected from a register field in the instruction. Out of 2^k registers in the CPU, one register is selected using k-bit field. This mode is useful to a long program in storing the intermediate results in the registers rather than in memory. This will result in fast execution since register accessing is much faster than memory accessing.

For example:

ADD R1, R2; Adds contents of registers R1 and R2 and stores the result in R1.

5. Register indirect mode In this mode the instruction specifies an address of CPU register that holds the address of the operand in memory. In other words, address field is a register which contains the memory address of operand (see Fig. 5.13). This mode is very useful for rapid access of the main memory location such as an array. The advantage of using this mode is that using small number of bits in the address field of the instruction a memory location is accessed rather than directly using large bits.

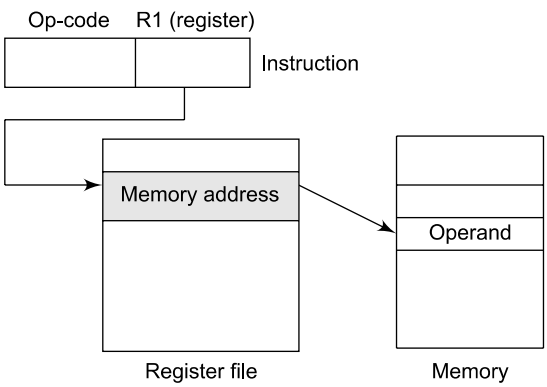


Figure 5.13 *Register indirect mode*

6. Auto-increment or auto-decrement mode This is similar to the register indirect mode except that after or before register's content is used

to access memory it is incremented or decremented. It is necessary to increment or decrement the register after every access to an array of data in memory, if the address stored in the register refers to the array. This can be easily achieved by this mode.

Class III Here, address field is a memory address.

Before discussing different addressing modes in this category, we need to know about the effective address of the operand.

Sometimes the instruction directly gives the address of the operand in its format. Sometimes the instruction does not give the operand or its address explicitly. Instead of that, it specifies the information from which the memory address of the operand can be determined. This address is referred to as *effective address*.

7. Direct (or Absolute) address mode In this mode the instruction contains the memory address of the operand explicitly. Thus, the address part of the instruction is the effective address (Fig. 5.14). Since the operand address is directly available in the instruction, there is no need for the effective address calculation step. Hence the instruction cycle time is reduced. Examples of direct addressing are:

(a) STA 2500H ; Stores the content of the accumulator in the memory location 2500H.

(b) LDA 2500H ; Loads the accumulator with the content of the memory location 2500H.

All branch-type instructions use direct addressing modes, because the address field of these specifies the actual branch address.

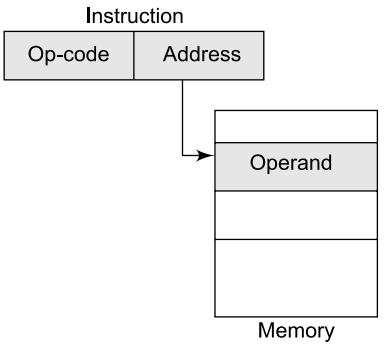


Figure 5.14 *Direct addressing mode*

8. Indirect address mode In this mode the instruction gives a memory address in its address field which holds the address of the operand. Thus, the address field of the instruction gives the address where the effective address is stored in memory. The Fig. 5.15 gives the idea of the mode. The following example illustrates the indirect addressing mode:

MOV R1, (X); Content of the location whose address is given in X is loaded into register R1.

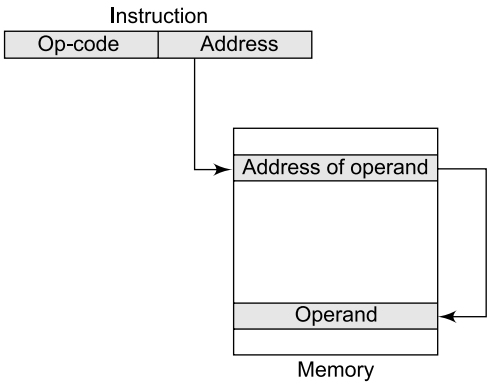


Figure 5.15 *Indirect addressing mode*

Since in this addressing, there is the scope of changing the address during run-time of program without changing the instruction content. This type of addressing modes gives flexibility in programming. It is very useful for implementing pointers in C language. However, instruction cycle time increases as there are two memory accesses.

Class IV Here, address field does not contain an effective address. The effective address is calculated from the following relation:

Effective address = address part of instruction + content of a CPU special register.

9. Relative address mode or PC-relative address mode In this mode the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. The instruction specifies the memory address of operand as the relative position of the current instruction address (Fig. 5.16). Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address. For example, the assembly language instruction

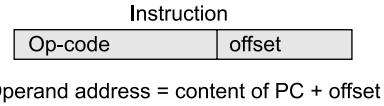


Figure 5.16 *Relative addressing mode*

JR 20; Branch to a location relative to the value 20 (offset)

The branch location is computed by adding the offset value 20 with the current value of the PC. This instruction (JR 20) requires 2 bytes: one for the op-code (JR) and another for its offset value 20. Consider that the instruction is stored in memory as shown in Fig. 5.17.

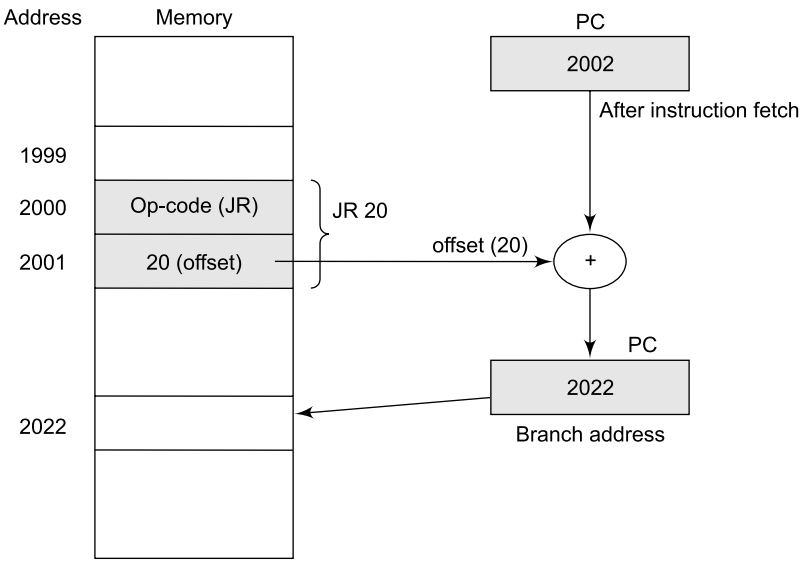


Figure 5.17 *Example of relative addressing*

Since the instruction is two-byte, the content of PC is 2000 + 2 = 2002 after the instruction fetch. The branch address is calculated by adding the content of PC with address part of instruction (offset), which gives 2022. Thus, after the instruction execution, the program branches to 2022 address in memory.

10. Indexed address mode In this mode the effective address is determined by adding the content of index register (XR) with the address part of the instruction (Fig. 5.18). This mode is useful in accessing operand array. The address part of the instruction gives the starting address of an operand array in memory. The index register is a special CPU register that contains an index value for the operand. The index value for operand is the distance between the starting address and the address of the operand. Any operand in the array can be accessed with the same instruction provided that the index register contains the

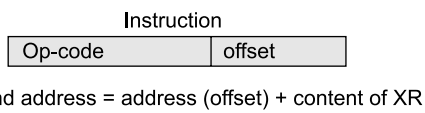


Figure 5.18 *Indexed addressing mode*

correct index value. For example, an operand array starts at memory address 1000 and assume that the index register XR contains the value 0002. Now consider load instruction

LDA 1000

The effective address of the operand is calculated as:

Effective address = 1000 + content of XR
= 1002.

11. Base register address mode This mode is used for relocation of the programs in the memory. *Relocation* is a technique of moving program or data segments from one part of memory to another part of memory. Relocation is an important feature of multi-programming systems. In this mode the content of the base register (BR) is added to the address part of the instruction to obtain the effective address (Fig. 5.19). This mode is similar to the indexed addressing mode, but exception is in the way they are used. A base register holds the starting address of a memory array of operands and the address part of the instruction gives a displacement or offset relative to this starting address. The base register addressing mode has the advantage over index addressing mode with respect to the size of instructions, because size of instructions in first case is smaller than that of second case.

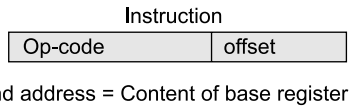


Figure 5.19 *Base register addressing mode*

Numerical Example

To illustrate various addressing modes, we will see the effect of the addressing modes on the instruction defined in Fig. 5.20. Suppose, the 2-word instruction stored at addresses 2000 and 2001 is an instruction:

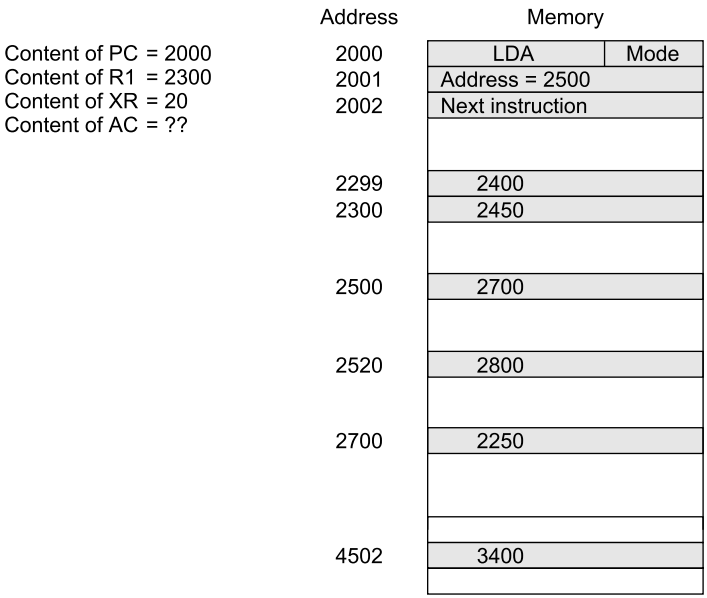


Figure 5.20 *Example of addressing modes*

LDA 2500; Load the accumulator (AC) register with the value as indicated by 2500

The address field value 2500 may be an operand value or memory address or register address of operand, which depends on the mode of instruction.

Table 5.1 shows the effective address of operand and content of AC register after the instruction execution for different addressing modes, if applicable.

In case of relative addressing, the effective address of the operand is calculated as:

$$\begin{aligned}\text{Effective address} &= \text{Content of PC} + \text{Address part of instruction}\\ &= 2002 + 2500\\ &= 4502\end{aligned}$$

The content of PC which holds the address of the next instruction to be executed, is 2002 since the current instruction being executed is stored in the locations 2000 and 2001. The operand address for other addressing modes can easily be determined.

Table 5.1 *Example of Addressing Modes*

Addressing Mode	*Effective Address*	*Content of AC*
Immediate	2001	2500
Register	—	2300
Register Indirect	2300	2450
Auto-decrement	2299	2400
Direct	2500	2700
Indirect	2700	2250
Relative	4502	3400
Indexed	2520	2800

5.9 INSTRUCTION SET

An instruction set of a processor is a collection that defines all the instructions. A complete instruction set is often referred to as the *instruction set architecture (ISA)* of the processor. An instruction from the set alone can be used in a program that runs on the processor. An instruction set of a processor must have the following characteristics:

1. Completeness We should be able to construct a machine language program to evaluate any function that is computable using a reasonable amount of memory space.

2. Efficiency Frequently required functions can be performed rapidly using relatively few instructions.

3. Regularity The instruction set should contain expected op-codes and addressing modes. For example, if there is left shift op-code, there should be right shift op-code.

4. Compatibility To reduce hardware and software design costs, the instructions may be required to be compatible with those of existing machines.

5.9.1 Instruction Types

The instructions in an instruction set are classified into different types on the basis of the following factors:

1. Op-code: Type of operation performed by the instruction.
2. Data: Type of data, i.e., binary, decimal, etc.
3. Operand location: Memory, register, etc.
4. Operand addressing: Method of specifying the operand location.
5. Instruction length: one byte, two byte, etc.
6. Number of address fields: 0 address, 1 address, 2 addresses, 3 addresses.

No two computers have same instruction set. But the actual operations available in the instruction set are not very different from one computer to another. Almost every computer has some unique instructions which attract the programmers. Computer architects give considerable attention to the framing of the instruction set since it involves both the programmer and the computer machine. Taking into account some important instructions of several popular computers, the instructions can be classified into following five types:

1. *Data transfer* instructions, which copy information from one location to another either in the processor's internal register set or in the external main memory.
2. *Arithmetic* instructions, which perform operations on numerical data.
3. *Logical* instructions, which include Boolean and other non-numerical operations.
4. *Program-control* instructions, such as branch instructions, which change the sequence in which programs are executed.
5. *Input-output* (I/O) instructions, which cause information to be transferred between the processor or its main memory and external I/O devices.

These types are not mutually exclusive. For example, the arithmetic instruction A = B + C implements the data transfer A ← B when C is set to zero.

Table 5.2 lists some sample instructions for each type of instructions. Since different computer manufacturers follow different types of symbolic names to instructions in assembly language notation, even for the same instruction; a simple mnemonics is adopted in this table for better comprehension.

Table 5.2 *List of Common Instruction Types*

Type	*Operation name*	*Description*
Data transfer	MOVE	Copy word or block from source to destination.
	LOAD	Copy word from memory to processor register.
	STORE	Copy word from processor register to memory.
	XCHG	Swap contents of source and destination.
	CLEAR	Transfer word of 0s to destination.
	SET	Transfer word of 1s to destination.
	PUSH	Transfer word from source to top of stack.
	POP	Transfer word from top of stack to destination.
Arithmetic	ADD	Compute sum of two operands.
	ADC	Compute sum of two operands and a carry bit.
	SUB	Compute difference of two operands.
	MULT	Compute product of two operands.

(*Contd.*)

(*Contd.*)

	DIV	Compute quotient and remainder of two operands.
	MUADD	Compute product of two operands and add it to a third operand.
	ABS	Replace operand by its absolute value.
	NEG	Change sign of operand.
	INCR	Add 1 to operand.
	DECR	Subtract 1 from operand.
	ASH(L/R)	Shift operand left (right) with sign bit.
Logical	AND	Perform bit-wise logical AND of operands.
	OR	Perform bit-wise logical OR of operands.
	XOR	Perform bit-wise logical exclusive-OR of operands.
	NOT	Complement the operand.
	SHIFT	(Logical) Shift operand left (right) introducing 0s at the end.
	ROTATE	Left (right) shift operand around closed path.
	CONVERT	Change data format, for example from binary to decimal.
Program control	JUMP	Unconditional transfer; load PC with specified address.
	JUMPC	Test specified conditions; if true, load PC with specified address.
	JUMPSUB	Place current control information including PC in known location and then load PC with specified address.
	RET	Restore current program control information including PC from known location.
	INT	Create a software interrupt; save current program control information in a known location and load the address corresponding to the specified code into PC.
	TEST	Test operand for specified condition and affect relevant flags.
	COMPARE	Make logical or arithmetic comparison of two or more operands and set relevant flags.
	WAIT (HOLD)	Stop program execution; test a specified condition continuously; when the condition is satisfied, resume instruction execution.
	NOP	No operation is specified, but program execution continues.
	EXECUTE	Fetch operand from specified location and execute as instruction; note that PC is not modified.
Input-output	IN (READ)	Copy of data from specified IO port to specified or implied destination.
	OUT (WRITE)	Copy of data from specified or implied source to IO port.
	START IO	Transfer instructions to I/O processor (IOP) to initiate an IO operation.
	HALT IO	Transfer instructions to IOP to terminate an IO operation.
	TEST IO	Transfer status information from IO system to specified destination.

5.9.2 CISC vs RISC

The computer architectures have been categorized into following two, based on CPU design and instruction set:

1. CISC (Complex Instruction Set Computer)
2. RISC (Reduced Instruction Set Computer)

Earlier most computer programming was done in assembly language. The instruction set architecture (ISA) was considered the most important part of computer architecture, because it determined how difficult it was to obtain optimal performance from the system.

Nowadays instruction set architecture (ISA) has become less significant for several reasons. Firstly, most programming is now done in high-level languages, so the programmer rarely interacts with the instruction set. Secondly, there is convergence that CISC is dropping less common instructions and RISC is including more common instructions.

All relatively older systems (main frame, mini or micro) follow CISC technique. Today's systems have been designed by taking important features from both types. RISC systems are more popular today due to their performance level as compared to CISC systems. However, due to high cost, RISC systems are used for special applications where speed, reliability, etc are important.

CISC In early days of computer history (before 1980s), most computer families started with simple instruction set, due to high cost of hardware. Then the hardware cost has dropped and the software cost has gone up steadily in the past three decades. Also, the semantic gap between HLL (high-level language) and computer architecture has widened. As the result of these, more and more functions have been built into hardware, making the instruction set very large and complex. Due to the popularity of micro-programmed control unit, large instruction set results.

Major characteristics of CISC

- A large number of instruction types used – typically from 100 to 250 instructions.
- A large number of addressing modes used- typically from 5 to 15 different modes.
- Some instructions that perform specialized tasks are used infrequently.
- Variable-length instruction formats.
- Small number of general-purpose registers (GPRs) – typically 8-24 GPRs.
- Clock per instruction (CPI) lies between 2 and 15.
- Mostly micro-programmed control units.
- Most instructions manipulate operands in memory.

Some *examples* of CISC processors are given below:

VAX 11/780

Number of instructions: 303
Instruction size: 2 – 8 bytes
Instruction format: not fixed
Addressing modes: 22
Number of general purpose registers: 16

Intel's Pentium

Number of instructions: 235
Instruction size: 1 – 8 bytes
Instruction format: not fixed
Addressing modes: 11
Number of general purpose registers: 8

Demerits of CISC machines

- CPU complexity: The micro-programmed control unit design becomes complex since the instruction set is large.
- System size and cost: Due to complexity of the CPU, a lot of hardware circuitry is used in the system. Thus, the hardware cost of the system and the power consumption have increased.

- Clock per instruction (CPI): Due to increased hardware circuitry, the propagation delays are more and the number of clock periods needed for each instruction execution is large and hence the overall execution time is reduced. In other words, the CPI consists of some number of clock pulses.
- Reliability: As heavy hardware is prone to frequent failures, the reliability of the system degrades.
- Maintainability: Since there are a large number of huge circuits, troubleshooting and detecting a fault is tough task.

RISC We started with RISC instruction sets and gradually moved to CISC instruction sets during the 1980s. After two decades of using CISC machines, computer scientists realized that only 25% of instructions of CISC machines are frequently used about 95% of the time. This proves that about 75% of hardware-supported instructions often are not used at all. Gradually VLSI (Very Large Scale Integration) technology has been invented, which offers design of very small-size chips (processor on a chip) with reasonable cost. Thus, we can replace micro-store, which earlier occupied about 70% of chip area, with registers. There was increased difference between CPU and memory speeds and complex instructions were not used by new compilers in CISC machines. These lead to the new concept of load/store architecture called RISC.

Major characteristics of RISC

- Relatively few number of instruction types—typically less than100 instructions.
- Relatively few addressing modes—typically less than or equal to 5.
- Fixed-length, easily decoded instruction formats.
- Large number of general-purpose registers (GPRs)—typically 32-192 GPRs.
- Mostly split data cache and instructions cache.
- Clock per instruction (CPI) lies between 1 and 2.
- Mostly hardwired control units.
- Memory access limited to load and store instructions.
- All operations are executed within registers of the CPU.

RISC processor's *example* includes:

Sun SPARC

Number of instructions: 52
Instruction size: 4 bytes
Instruction format: fixed
Addressing modes: 2
Number of general purpose registers: up to 520

PowerPC

Number of instructions: 206
Instruction size: 4 bytes
Instruction format: not fixed (but small differences)
Addressing modes: 2
Number of general purpose registers: 32

Demerits of RISC machines

- Lacks some sophisticated instructions found in CISC processors.

- Several RISC instructions may be needed to replace one CISC instruction, which results in longer programs.
- Difficult to program at assembly level.
- No solution for floating point numbers.
- Performance is intimately tied to compiler optimisation.
  — aim is to make procedure call/return and parameter passing highly efficient.
- More error-prone and less flexible hardwired control units.

The *conclusion* about these two classes of computers has been summarized as:

- RISC is good in environments requiring
  — small size.
  — low power consumption.
  — low heat dissipation.
- On modern-day general-purpose machines, RISC and CISC have converged to an extent. For example, Intel's Pentium series, the VAX 9000 and Motorola 88100 are built with mixed features taken from both the RISC and CISC camps.
- Modern RISCs (ARM, Sun SPARC, HP PA-RISC) more complex than forebears.
- Modern CISCs incorporate many features learned from RISC.

5.10 RISC PROCESSORS: CASE STUDY

5.10.1 Sun SPARC

SPARC stands for *Scalable Processor ARChitecture*. It is an architecture defined by Sun Microsystems. Sun developed its own SPARC implementation but also licenses the architecture to other vendors to produce SPARC-compatible machines. The SPARC architecture is inspired by the Berkeley RISC I machine, and its instruction set and register organization is based closely on the Berkeley RISC model.

The instruction set of the SPARC architecture has a distinct RISC style. The ar-chitecture specifications describe a processor in which data and memory addresses are 64-bit long. Instructions are of equal length, and they are all 32-bit long. Both integer and floating-point instructions are provided.

There are two register files, one for integer data and one for floating-point data. Integer registers are 64-bit long. Their number is implementation dependent and can vary from 64 to 528. SPARC uses a scheme known as *register windows.* At any given time, an application program sees only 32 registers, R0 to R31. Of these, the first eight are global registers that are always accessible. The remaining 24 registers are local to the current context.

Floating-point registers are only 32-bit long because this is the length of single-precision floating-point numbers according to the IEEE Standard described in Chapter 2. The instruction set includes floating-point instructions for double- and quad-precision operations. Two sequentially numbered floating-point registers are used to hold a double-precision operand and four are used for quad-precision. There is a total of 64 registers, F0 to F63. Single precision operands can be stored in F0 to F31, double precision operands in F0, F2, F4, . . . , F62, and quad-precision in F0, F4, F8, . . . , F60.

Instruction Set

Table 5.3 lists the instruction set for the SPARC architecture.

Table 5.3 *SPARC Instrcution Set*

OP-CODE	*Description*	*OP-CODE*	*Description*
Load/Store Instructions		**Arithmetic Instructions**	
LDSB	Load signed byte	ADD	Add
LDSH	Load signed half word	ADDCC	Add, set icc
LDUB	Load unsigned byte	ADDX	Add with carry
LDUH	Load unsigned half word	ADDXCC	Add with carry, set icc
LD	Load word	SUB	Subtract
LDD	Load double word	SUBCC	Subtract, set icc
STB	Store byte	SUBX	Subtract with carry
STH	Store half word	SUBXCC	Subtract with carry, set icc
STD	Store word	MULSCC	Multiply step, set icc
STDD	Store double word	**Jump/Branch Instructions**	
Shift Instructions		BCC	Branch on condition
SLL	Shift left logical	FBCC	Branch on floating-point condition
SRL	Shift right logical	CBCC	Branch on coprocessor condition
SRA	Shift right arithmetic	CALL	Call procedure
Boolean Instructions		JMPL	Jump and link
AND	AND	TCC	Trap on condition
ANDCC	AND, set icc	SAVE	Advance register window
ANDN	NAND	RESTORE	Move windows backward
ANDNCC	NAND, set icc	RETT	Return from trap
OR	OR	**Miscellaneous Instructions**	
ORCC	OR, set icc	SETHI	Set high 22 bits
ORN	NOR	UNIMP	Unimplemented instruction(trap)
ORNCC	NOR, set icc	RD	Read a special register
XOR	XOR	WR	Write a special register
XORCC	XOR, set icc	IFLUSH	Instruction cache flush
XNOR	Exclusive NOR		
XNORCC	Exclusive NOR, set icc		

#icc: Integer condition code field of the PSR (Programme Status Register)

Instruction Format

SPARC uses a simple set of 32-bit instruction formats (Fig. 5.21)

Addressing Modes

Only simple load and store instructions reference memory. There are separate load and store instructions for word (32-bit), double word, half word and byte. For the latter two cases, there are instructions for loading these quantities as signed or unsigned numbers. Signed numbers are sign extended to fill out the 32-bit destination register. Unsigned numbers are padded with 0s.

The only available addressing mode, other than register, is a displacement mode. That is, the effective address (EA) of an operand consists of a displacement from an address contained in a register:

$$EA = (R_{S1}) + S2$$

or $$EA = (R_{S1}) + (R_{S2})$$

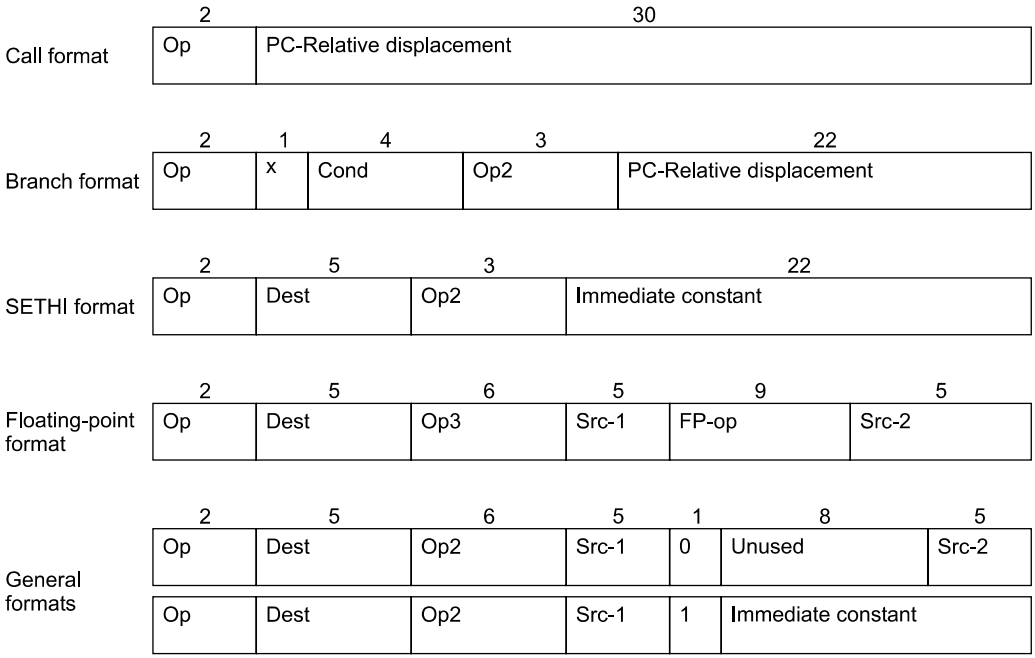


Figure 5.21 *SPARC Instruction formats*

depending on whether the second operand is immediate or a register reference. This single addressing mode is quite versatile and can be used to deduce other addressing modes, as shown in Table 5.4.

Table 5.4 *Deduced addressing modes with SPARC*

Mode	*Algorithm*	*SPARC Equivalent*	*Instruction Type*
Immediate	operand = A	S2	Register to register
Direct	EA = A	R_0 + S2	Load, store
Register	EA = R	R_{S1}, R_{S2}	Register to register
Register indirect	EA = (R)	R_{S1} + 0	Load, store
Displacement	EA = (R) + A	R_{S1} + S2	Load, store

5.10.2 PowerPC

In the early 1990s, IBM, Motorola and Apple collaborated on the development of a RISC-style processor family, the PowerPC for the personal computer and workstation markets. PowerPC processors produced by both IBM and Motorola have been used in IBM and Apple computers. In general, these processors have architectural features that have provided computing power similar to that of the Intel 32-bit processors over comparable time periods.

The following are the principal members of the PowerPC family (Table 5.5):

- **601:** The purpose of the 601 was to bring the PowerPC architecture to the marketplace as quickly as possible. The 601 is a 32-bit machine.

- **603:** This was intended for low-end desktop and portable computers. It is also a 32-bit machine, comparable in performance with the 601, but with lower cost and a more efficient implementation.
- **604:** This was also intended for desktop computers and low-end servers. Again, this is a 32-bit machine, but it uses much more advanced superscalar (discussed in chapter 8) design techniques to achieve greater performance.
- **620:** This was intended for high-end servers. This is the first member of the PowerPC family to implement a full 64-bit architecture, including 64-bit registers and data paths.
- **740/750:** This is also known as the G3 processor. This processor integrates two levels of cache in the main processor chip, providing significant performance improvement over a comparable machine with off-chip cache organization.
- **G4:** This processor increases the parallelism and internal speed of the processor chip.

Table 5.5 *PowerPC Processor family summary*

	601	*603*	*604*	*740/750 (G3)*	*G4*
First release year	1993	1994	1994	1997	1999
Clock speed (MHz)	50–120	100–300	166–350	200–366	500
L1 cache	–	16 Kbyte instr. 16 Kbyte data	32 Kbyte instr. 32 Kbyte data	32 Kbyte instr. 32 Kbyte data	32 Kbyte instr. 32 Kbyte data
Off-chip L2 cache support	–	–	–	256 Kbyte - 1Mbyte	256 Kbyte - 1 Mbyte
Number of transistors (millions)	2.8	1.6-2.6	3.6-5.1	6.35	–

Register Set

There are 32 general-purpose registers and 32 floating-point registers. The floating-point registers are 64 bits long. The IEEE standard is used for representation of floating-point numbers. The PowerPC architecture defines both 32-bit and 64-bit modes of operation. The size of the general-purpose registers is determined by which of these modes is implemented by a particular processor.

Data Types

The PowerPC can deal with data types of 8 (byte), 16 (half word), 32 (word), and 64 (double word) bits in length. Some instructions require that memory operands be aligned on a 32-bit boundary. In general, however, alignment is not required. One interesting feature of the PowerPC is that it can use either little-endian or big-endian style; that is, the least significant byte is stored in the lowest or highest address.

The byte, half word, word, and double word are general data types. The processor interprets the contents of a given item of data depending on the instruction. The fixed-point processor recognizes the following data types:

- ***Unsigned byte*** This can be used for logical or integer arithmetic operations. It is loaded from memory into a general register by zero extending on the left to the full register size.
- ***Unsigned half word*** This is same as for unsigned byte, but for 16-bit quantities.
- ***Signed half word*** This is used for arithmetic operations; loaded into memory by sign extending on the left to full register size (i.e. the sign bit is replicated in all vacant positions).
- ***Unsigned word*** It is used for logical operations and as an address pointer.

- ***Signed word*** This is used for arithmetic operations.
- ***Unsigned double word*** This is used as an address pointer.
- ***Byte string*** From 0 to 128 bytes in length.

In addition, the PowerPC supports the single- and double-precision floating-point data types defined in IEEE 754.

Instruction Set

Table 5.6 lists the instruction set for the PowerPC architecture.

Table 5.6 *PowerPC instruction set*

Instruction	*Description*
	Load/Store
lwzu	Load word and zero extend to left; update source register
ld	Load double word
lmw	Load multiple word; load consecutive words into contiguous registers from the target register through general-purpose register 31
lswx	Load a string of bytes into registers beginning with target register; 4 bytes per register; wrap around from register 31 to register 0
	Integer Arithmetic
add	Add contents of two registers and place in third register
subf	Subtract contents of two registers and place in third register
mullw	Multiply low-order 32-bit contents of two registers and place 64-bit product in third register
divd	Divide 64-bit contents of two registers and place in quotient in third register
	Branch Oriented
b	Unconditional branch
bl	Branch to target address and place effective address of instruction following the branch into the Link Register
bc	Branch conditional on Count Register and/or on bit in Condition Register
sc	System call to invoke an operating system service
trap	Compare two operands and invoke system trap handler if specified conditions are met
	Logical and Shift
cmp	Compare two operands and set four condition bits in the specified condition register field
crand	Condition register AND: two bits of the Condition Register are ANDed and the result placed in one of the two bit positions
and	ANDing contents of two registers and place in third register
cntlzd	Count number of consecutive 0 bits starting at bit zero in source register and place count in destination register
rldic	Rotate left double word register, AND with mask, and store in destination register
sld	Shift left bits in source register and store in destination register
	Floating-Point
lfs	Load 32-bit floating-point number from memory, convert to 64-bit format, and store in floating-point register
fadd	Add contents of two registers and place in third register
fmadd	Multiply contents of two registers, add the contents of a third, and place result in fourth register

(*Contd.*)

(*Contd.*)

Instruction	*Description*
fcmpu	Compare two floating-point operands and set condition bits
	Cache Management
dcbf	Data cache block flush; perform lookup in cache on specified target address and perform flushing operation
icbi	Instruction cache block invalidate

Instruction Format

All instructions in the PowerPC are 32-bit long and follow a regular format. The first 6 bits of an instruction specify the operation to be performed (i.e., Op-code). In some cases, there is an extension to the op-code elsewhere in the instruction that specifies a particular sub-case of an operation. In Fig. 5.22, op-code bits are represented in leftmost field of each format.

6-bit	5-bit	5-bit	16-bit		
Ld/st indirect	Dest. register	Base register	Displacement		
Ld/st indirect	Dest. register	Base register	Index register	Size, sign, update	/
Ld/st indirect	Dest. register	Base register	Displacement		XO

(a) Load/store instructions

Arithmetic	Dest. register	Src. register	Src. register	O	Add, sub, etc.	R
Add, sub, etc.	Dest. register	Src. register	Src. register	Signed immediate value		
Logical	Src. register	Dest. register	Src. register	And, Or, Xor, etc.		R
And, Or, etc.	Src. register	Dest. register	Unsigned immediate value			
Rotate	Src. register	Dest. register	Shift amnt.	Mask begin	Mask end	R
Rotate or shift	Src. register	Dest. register	Src. register	Shift type or mask		R
Rotate	Src. register	Dest. register	Shift amnt.	Mask	XO S	R
Rotate	Src. register	Dest. register	Shift amnt.	Mask	XO	R
Shift	Src. register	Dest. register	Shift amnt.	Shift type or mask	S	R

(b) Integer arithmetic, logical and shift/rotate instructions

Branch	Long immediate			A	L
Br Conditional	Options	CR bit	Branch displacement	A	L
Br Conditional	Options	CR bit	Indirect through link or counter register		L

(c) Branch instructions

CR	Dest. bit	Source bit	Source bit	And, Or, Xor, etc.	/

(d) Condition register logical instructions

(*Figure Contd*)

Flt Sgl/dbl	Dest. register	Src. register	Src. register	Src. register	Fadd, etc.	R

(e) Floating-point arithmetic instructions

A = Absolute or PC Relative
L = Link to Subroutine
O = Record Overflow in XER
R=Record conditions in CR1
XO = Op-Code Extension
S = Part of Shift Amount Field

Figure 5.22 *PowerPC Instruction Formats*

Addressing Modes

Like most RISC machines, the PowerPC uses a simple and relatively straightforward set of addressing modes. As shown in Table 5.7, these modes are conveniently classified with respect to the type of instruction.

Table 5.7 *PowerPC Addressing Modes*

Mode	*Description*
	Load/Store Addressing
Indirect	EA = (BR) + D
Indirect indexed	EA = (BR) + (IR)
	Branch Addressing
Absolute	EA = I
Relative	EA = (PC) + I
Indirect	EA = (L/CR)
	Fixed-Point Computation
Register	EA = GPR
Immediate	Operand = I
	Floating-Point Computation
Register	EA = FPR
EA = effective address	(X) = contents of X
BR = base register	IR = index register
L/CR = link or count register	GPR = general-purpose register
FPR = floating-point register	D = displacement
I = immediate value	PC = program counter

5.11 INTRODUCTION TO PIPELINING

Pipelining offers an economical way to realize parallelism in digital computers. Pipeline processing has led to the tremendous improvement of system throughput in the modern digital computers. The concept of pipeline processing in a computer is similar to assembly lines in an industrial plant. A job passes through various stages of the assembly (pipe) line and finally rolls out. The pipeline is

continuously fed at the input end with new jobs one at a time. At each stage a part of the job is done and then it is passed on to the next stage; when it passes through the last stage, it gets completed, and the finished product rolls out.

To get the basic idea of pipelining, let us consider a real-life example: "Suppose 100 students appear for an examination of a subject. There are 5 questions, all are to be attempted. After the examination, all 100 scripts are to be evaluated by examiner(s) for grading. Further, we are assuming that each question evaluation takes 5 minutes, for simplicity." For evaluation of all 100 scripts, we may employ two approaches:

Approach-1 *Employing one examiner for all scripts.*
In this case, the total evaluation time = $100 \times 5 \times 5 = 2500$ minutes.

Approach-2 *Employing five examiners*, assuming each examiner is specialized for a single question evaluation.

Assumption is that ith examiner (E_i) is specialized for ith question (Q_i) evaluation for all scripts. All examiners are sitting adjacently and all 100 scripts are stacked in front of first examiner who is specialized for first question evaluation for all scripts. After first question evaluation, he/she passes the script to the adjacent second examiner for second question evaluation of that script. During that time, the first examiner is free to take the second script from the stack for its first question evaluation. Therefore, after the first 10 minutes, the first examiner takes script number 3 for its first question evaluation; second examiner takes script number 2 for its second question and third examiner takes script number 1 for its third question evaluation, during the next 5 minutes. This process continues. Thus, after first $5 \times 5 = 25$ minutes, the first script gets evaluated completely and after another 5 minutes (after first 30 minutes) the second script gets evaluated totally and so on. Thus, after first 25 minutes, a question gets evaluated in each 5 minutes. This process is shown in Figure 5.23. All 100 scripts are evaluated in this manner. Thus, total evaluation time = $25 + 99 \times 5 = 520$ minutes. This suggests that employing specialized examiners takes much less time compared to the one examiner employing approach.

The approach-2 uses pipelining concept, while approach-1 uses non-pipelining.

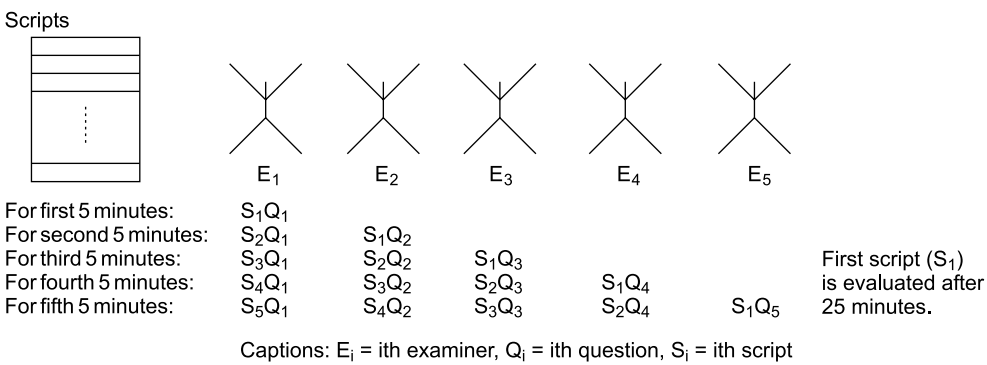


Figure 5.23 *Real-life example of pipelining*

5.11.1 Principles of Pipelining

Pipelining is a technique of decomposing a sequential task into subtasks, with each subtask being executed in a special dedicated stage (or segment) that operates concurrently with all other stages. Each stage performs partial processing dictated by the way the task is partitioned. Result obtained from a stage is transferred to the next stage in the pipeline. The final result is obtained after the instruction has passed through all the stages. All stages are synchronized by a common clock. Stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface clocked latches. The latches are made with master-slave flip-flops, which can isolate inputs from outputs. Upon the arrival of a clock pulse, all latches transfer data to the next stage simultaneously. Figure 5.24 shows the linear pipeline with k stages.

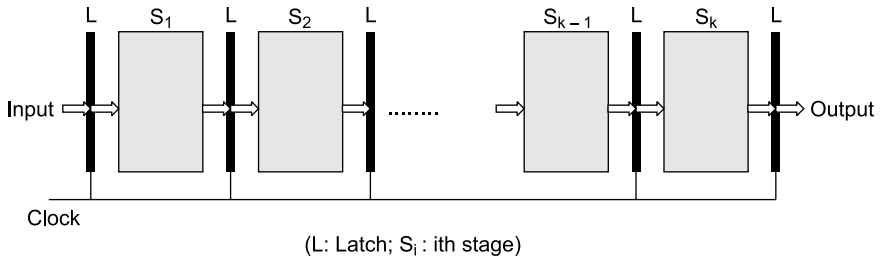


Figure 5.24 *Concept of pipelining*

Overlapped operations of pipeline processors are represented by a two-dimensional chart called *space-time diagram* as shown in Figure 5.25. Assume that the pipeline uses 4 stages.

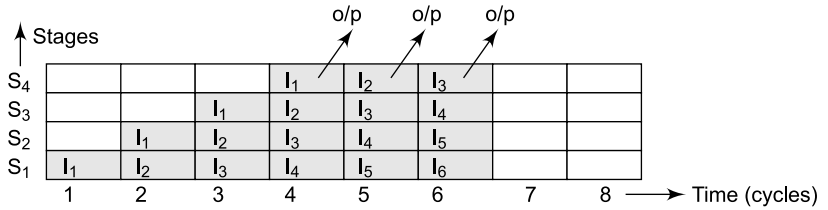


Figure 5.25 *Space-time diagram of 4-stage pipeline processor*

Clock-period Ideally, we expect the execution delay in all stages should be same. But, due to the difference in the hardware circuitry in different stages of a pipeline, execution delay cannot be same. Shorter latency stages may finish early and long latency stages are still continuing with their specified tasks. Since one single clock circuitry is used for the synchronization in the pipeline, we have to define the clock period uniformly so that no overwriting or no partial execution of results produced in any stage.

The logic circuitry in each stage S_i has a time delay denoted by τ_i. Let τ_l be the time delay of each interface latch. The clock-period of a linear pipeline is defined by

$\tau = \{\tau_i\}_{i=1}^{k} + \tau_l = \tau_m + \tau_l$ where τ_m = maximum stage delay.

The reciprocal of the clock-period is called the *frequency* $f = 1/\tau$ of a pipeline processor.

5.11.2 Performance of Pipeline Processor

Three parameters to measure the performance of a pipeline processor are

- speedup
- efficiency
- throughput

Speedup: It is defined as

$$S_k = \frac{\text{Time to execute n tasks in k-stage non - pipeline processor}}{\text{Time to execute n tasks in k - stage pipeline processor}}$$

$$= \frac{n.k.\tau}{\tau[k + (n - 1)]} \text{ where, } \tau = \text{clock period of the pipeline processor.}$$

Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n - 1)]$ units, where k clock periods (cycles) are needed to complete the execution of the first task and remaining (n – 1) tasks require (n – 1) cycles. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can enter the pipeline until the previous task finishes.

It can be noted that the maximum speedup is k, for $n >> k$ (i.e. $n \to \infty$). In other words, the maximum speedup that a linear pipeline can provide is k, where k is the number of pipeline stages. But this maximum speedup is never fully achievable because of data dependencies between instructions, interrupts, program branches, etc.

The larger the number k of pipeline stages, the higher the potential speedup performance. However, the number of pipeline stages cannot be increased indefinitely due to practical constraints on costs, control complexity, circuit implementation, and packaging limitations. Furthermore, the number of tasks n also affects the speedup; the longer the better in using a pipeline.

Efficiency To define it, we need to define another term "time-space span". It is the product (area) of a time interval and a stage space in the space-time diagram. A given time-space span can be in either a busy state or an idle state, but not both.

The *efficiency* of a linear pipeline is measured by the percentage of busy time-space spans over the total time-space span, which equals the sum of all busy and idle time-space spans. Let n, k, τ be the number of tasks (instructions), the number of pipeline stages and the clock period of a linear pipeline, respectively. Then the efficiency is defined by

$$\eta = \frac{n.k.\tau}{k.[k.\tau + (n-1).\tau]} = \frac{n}{k + (n-1)}$$

Note that $\eta \to 1$ (i.e., 100%) as $n \to \infty$. This means that the larger the number of tasks flowing through the pipeline, the better is its efficiency. Additionally, it can be observed that $\eta = S_k/k$, from the expressions of speedup and efficiency. This yields another view of the efficiency of a linear pipeline as the ratio of its actual speedup to the ideal speedup k. For the same reason as speedup, this ideal efficiency is not achievable.

Throughput The number of tasks that can be completed by a pipeline per unit time is called its throughput. Mathematically, it is defined as

$$\omega = \frac{n}{k.\tau + (n-1).\tau} = \frac{\eta}{\tau}$$

Note that in ideal case, $\omega = 1/\tau = f$, frequency, when $\eta \to 1$ as $n \to \infty$. This means that the maximum throughput of a linear pipeline is equal to its frequency, which corresponds to one output result per clock period.

Problem 5.1 Suppose the time delays of the four stages of a pipeline are $\tau_1 = 60$ ns, $\tau_2 = 70$ ns, $\tau_3 = 90$ ns and $\tau_4 = 80$ ns respectively and the interface latch has a delay $\tau_l = 10$ ns, then

(i) What would be the maximum clock frequency of the pipeline?

(ii) What is the maximum speedup of this pipeline over its equivalent non-pipeline counterpart?

Solution The clock-period should at least $\tau = 90 + 10 = 100$ ns.

So, the clock frequency, $f = 1/\tau = 1/100 = 10$ MHz.

In case of non-pipeline, the time delay $= \tau_1 + \tau_2 + \tau_3 + \tau_4 = 60 + 70 + 90 + 80 = 300$ ns.

So, the speed-up $= 300/100 = 3$. This means that the pipeline processor is 3 times faster than its equivalent non-pipeline processor.

5.11.3 Instruction Pipeline

We know that an instruction execution cycle may consist of many operations like, fetch instruction, decode instruction, fetch operands, execute instruction, and write-back the result into memory. These operations of the instruction execution cycle can be realized through the pipelining concept. Each of these operations forms one stage of a pipeline. Each operation may require one or more clock periods to execute, depending on the instruction type, processor and memory architectures used. The overlapping of execution of the operations through the pipeline provides a speedup over the normal execution. Thus, the pipeline used for instruction cycle operations is known as *instruction pipeline*. A typical instruction pipeline is shown in Figure 5.26. The *instruction fetch stage* (IF) fetches instructions from memory, presumably one per cycle. The *instruction-decode stage* (ID) resolves the instruction function like, add or subtract, etc., to be performed and identifies the operands needed. The *operand fetch stage* (OF) fetches the operand values needed for execution into processor registers. The *execute stage* (EX) executes the instructions on the stored operand values. The last *write-back stage* (WB) is used to write results into registers or memory. All high-performance computers are now equipped with this pipeline.

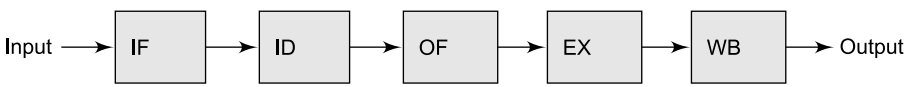


Figure 5.26 *A 5-stage instruction pipeline*

5.11.4 Pipeline Hazards

Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. The instruction is said to be *stalled*. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall. Thus, the

pipeline cannot execute instructions at its peak rate due to such situations. There are three types of pipeline hazards:

1. Control hazards
2. Structural hazards
3. Data hazards

Control Hazards Such hazards arise from the pipelining of branches and other instructions that change the content of program counter (PC) register. Branch instructions can cause delay in pipelined processors, because the processor cannot determine which instruction to fetch next until the branch instruction has been executed completely. A typical computer program consists of four types of instructions, as below:

Arithmetic/load type	:	60%
Store type	:	15%
Unconditional branch type	:	5%
Conditional branch type	:	20%.

The arithmetic/load and store instructions (75% of a typical program) do not alter the sequential execution order of the program (in-order execution). This implies that pipeline-flow is linear type. However, the branch type instructions (25%) may alter the program counter (PC) content in order to jump to a program location other than the next instruction. In other words, the branch-type instructions may cause out-orders execution. This means that the branch types of instructions causes some adverse effects on the pipeline performance. The effect of branching on pipeline performance described in Figure 3.13, using a linear instruction pipeline that consists of five stages: instruction fetch (IF), instruction decode (ID), operand fetch (OF), execution (EX) and write-back result (WB). Possible memory conflicts between overlapped fetches are ignored and sufficiently large cache memory is assumed.

As shown in Figure 5.27(a), a stream of instructions is executed continuously in an overlapped fashion in the instruction pipeline if any branch-type instruction is not encountered in stream of instructions. The performance, in this case, would be one instruction execution in each pipeline cycle after first k cycles in a k-stage pipeline, as discussed in the speedup definition.

On contrary, if a branch instruction enters in the pipeline, the performance would be hampered. After execution of an instruction halfway down the pipeline, a conditional branch instruction may be resolved and then the program counter (PC) needs to be loaded with a new address to which program should be directed, making all pre-fetched instructions (either in the cache memory or already in the pipeline) useless. The next instruction cannot be initiated until the completion of the current branch instruction. This causes extra time penalty (delay) in order to execute next instruction in the pipeline, as shown in Figure 5.27(b).

Techniques to Solve the Control Hazards Pipelined computers employ various techniques to minimize the performance degradation caused by instruction branching. Following are such techniques:

Prefetching One way of handling a conditional branch is to prefetch both the target instruction part and the instructions following the branch. In other words, instruction words ahead of the one currently being decoded in the instruction-decoding (ID) stage are fetched from the memory system before the ID stage requests them. Figure 5.28 illustrates the pre-fetching technique. The memory here is

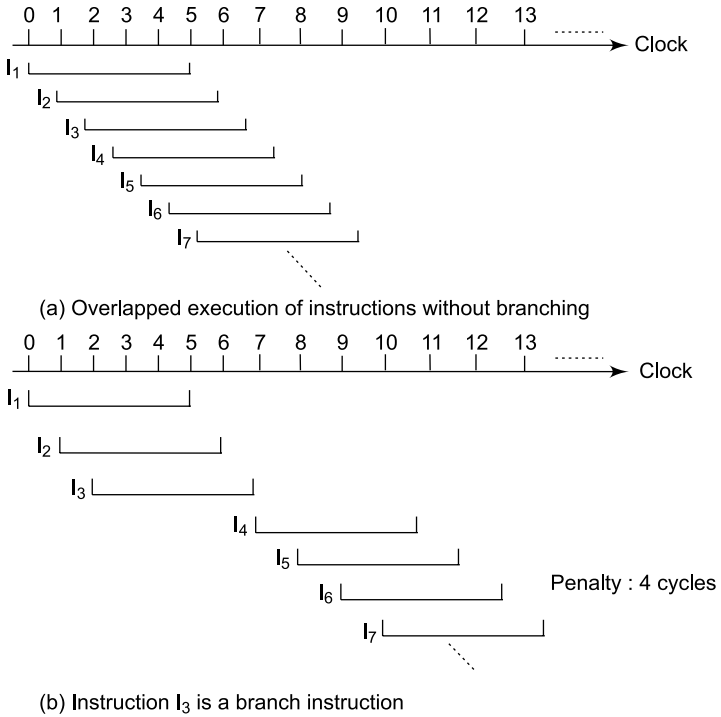


Figure 5.27 *The effect of branching on the performance of an instruction pipeline*

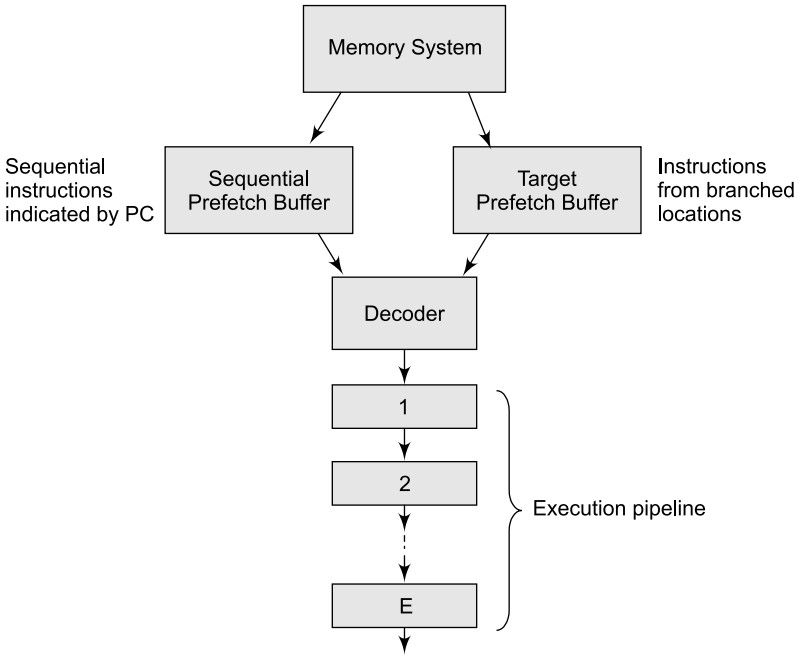


Figure 5.28 *An instruction pipeline with prefetching*

assumed to be in multiple modules, all modules can be accessed concurrently. There are two prefetch buffers (caches) of instructions used:

(a) Sequential prefetch buffer.
(b) Target prefetch buffer.

The *sequential prefetch buffer* holds instructions fetched during the sequential part of a program. The *target prefetch buffer* holds instructions fetched from the target of a conditional branch. When a branch is resolved as successful, the entire sequential prefetch buffer is invalidated, but the other buffer is validated. However, when a branch is unsuccessful, the reverse is true. If the instruction requested by the decoder is available in the sequential buffer used for sequential instructions or is available in the target buffer if a conditional branch has just been resolved and is successful, it enters the decoder with no delay. Otherwise, the decoder is idle until the instruction returns from memory.

Branch Prediction Another procedure that some computers use is branch prediction. A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed. The prediction is based on either branch code types statically or branch history during program execution. The probability of branch with respect to a particular branch instruction type can be used to predict branch (*Static branch strategy*). Sometimes the branch history is taken into consideration to predict whether or not the branch will be taken next time when it occurs (*Dynamic branch strategy*). According to the prediction, the pipeline begins prefetching the instruction stream from the predicted path. A correct prediction eliminates the wasted time caused by branch penalties.

Delayed Branch A procedure employed in most RISC processors is the *delayed branch*. In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions. If no useful instructions can be placed after a branch instruction, no-operation (NOP) instructions can be inserted there. This causes the computer to fetch the target instruction during the execution of the no-operation instruction, allowing a continuous flow of the pipeline.

As an example of a delayed branch, consider the execution of a code fragment in Figure 5.29(a). The original program is modified by moving the useful instruction I_1 into the delay slot (step) after the branch instruction I_3. By so doing, instructions I_1, I_4, and I_5 are executed regardless of the branch outcome.

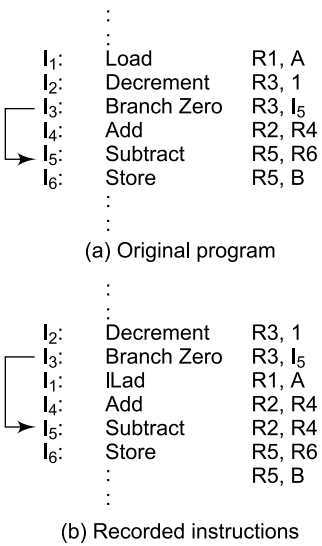


Figure 5.29 *Reordering of instructions for a delayed branch*

In case the branch is unsuccessful, the execution of the modified program produces the same results as the original program. In case the branch is successful in the modified program, execution of the delayed instructions I_1 and I_5 is needed anyway. Only one cycle was wasted in executing instruction I_4, which is not needed.

Therefore, if we use a five-stage instruction pipeline, the delay slot has been reduced to one for an

unsuccessful branch and reduced to two for a successful branch in this example.

In general, data dependence between instructions moving across the branch and the remaining instructions being scheduled must be analyzed. Since instructions I_1 and I_4 are independent of the remaining instructions (I_2, I_3, I_5, and I_6), leaving them in the delay slot will not create data hazards.

Sometimes NOP fillers can be inserted in the delay slot if no useful instructions can be found. However, inserting NOP fillers does not save any cycles in the delayed branch operation. From the above analysis, one sees that delayed branching is effective in short instruction pipelines with about four stages. Delayed branching has been built into most RISC processors, including the MIPS R4000 and Motorola MC88110.

Structural Hazards Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same clock period.

Example Instruction ADD R4, X fetches operand X from memory in the OF stage at third clock period. The memory doesn't accept another access during that period. For this, (i+2)th instruction cannot be initiated at third clock period to fetch the instruction from memory. Thus, one clock cycle is stalled in the pipeline for all subsequent instructions. This is shown in Figure 5.30.

Clock ⟶	1	2	3	4	5	6	7	8	9	10	11	12	13
ADDR4, X	IF	ID	OF	EX	WB								
Instr. i+1		IF	ID	OF	EX	WB							
Instr. i+2			Stall	IF	ID	OF	EX	WB					
Instr. i+3					IF	ID	OF	EX	WB				

Penalty : 1 cycle

Figure 5.30 *Structural hazard in instruction pipeline*

Technique to Solve Structural Hazards Certain resources are duplicated in order to avoid structural hazards. Functional units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time. A classical way to avoid hazards at memory access is by providing separate data and instruction caches.

Data Hazards Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard* (sometimes known as *logic hazard*).

Example Suppose we have two instructions, I_1 and I_2 in a program. In a pipeline the execution of I_2 can start before I_1 has terminated. If in a certain stage of the pipeline, I_2 needs the result produced by I_1, but this result has not yet been generated, then we have a data hazard.

According to various data update patterns in instruction pipeline, there are three classes of data hazards exist:

- Write After Read (WAR) hazards
- Read After Write (RAW) hazards
- Write After Write (WAW) hazards

Note that read-after-read is not a hazard, because nothing is changed on a read operation. To

discuss these three hazards, we define the following:

Resource object: It refers to working registers, memory locations and special flags.

Data object: The contents of resource objects are called data objects. Each instruction can be considered as a mapping from a set of data objects to a set of data objects.

Domain of instruction: The domain D(I) of an instruction I is the set of resource objects whose data objects may affect the execution of instruction I.

Range of instruction: The range R(I) of an instruction I is the set of resource objects whose data objects may be modified by the execution of instruction I. Obviously, the domain of an instruction holds operands to be read (retrieved) in the instruction execution and its range set will hold the results produced.

Now, consider two instructions I and J in a program, where I occurs before J. Between instructions I and J, there may be none or other instructions. Meanings of three hazards are as follows (see Figure 5.31):

- *RAW (read after write)*—J tries to read some data object before I writes it, so J incorrectly gets the old value of the data object. Program order must be preserved to ensure that J receives the value from I.
- *WAW (write after write)*—J tries to write (modify) some data object before it is written by I. The writes end up being performed in the wrong order, leaving the value written by I rather than the value written by J in the destination.
- *WAR (write after read)*—J tries to write some data object before it is read by I, so I incorrectly gets the new value.

Listed below are conditions under which possible hazards can occur:

$$R(I) \cap D(J) \neq \phi \quad \text{for RAW hazard}$$
$$R(I) \cap R(J) \neq \phi \quad \text{for WAW hazard}$$
$$D(I) \cap R(J) \neq \phi \quad \text{for WAR hazard}$$

These conditions are necessary conditions but not sufficient conditions. This means the hazard may not appear even if one or more of the conditions exist. The occurrence of a data hazard depends on the order in which the two instructions are executed. As long as the order is right, the hazard will not occur.

Detection of Data Hazards Hazard detection method can be implemented in the instruction-fetch (IF) stage of a pipeline processor by comparing the domain and the range of the incoming instruction with those of the instructions being processed in the pipeline. If any of the earlier necessary conditions is detected, a warning signal must be generated to prevent the hazard from taking place.

Solution of Data Hazards The system must resolve the interlock situation when a hazard is detected. Consider the sequence of instructions {…. I, I+1, …., J, J+1,….} in which a hazard has been detected between the current instruction J and a previous instruction I. This hazardous situation can be resolved in one of the two following ways:

- One simple solution is to stall the pipeline and to ignore the execution of instructions J, J+1,…., down the pipeline until the instruction I has passed the point of resource conflict.
- A more advanced approach is to ignore only instruction J and continue the flow of instructions J+1, J+2,…, down the pipeline. However, the potential hazards due to the suspension of J must be continuously tested as instructions J+1, J+2,… execute prior to J. Thus, multilevels of

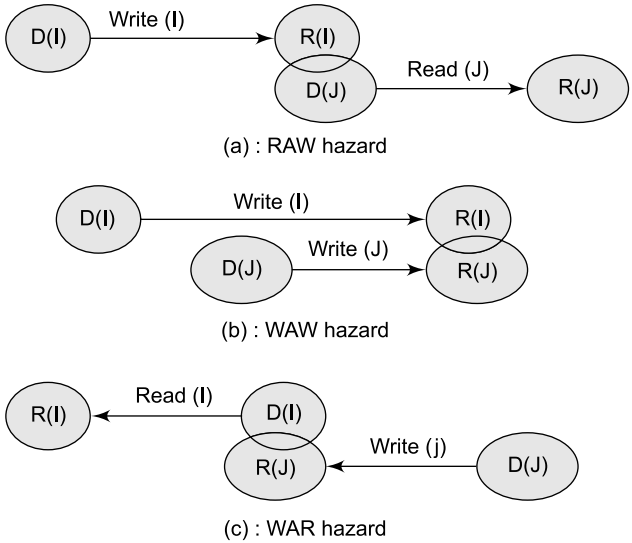


Figure 5.31 *Possible data hazards in an instruction pipeline*

hazard detection may be encountered, which requires much more complex control policies to resolve such multilevels of hazards.

SOLVED PROBLEMS

1. *If each register is specified by 3 bits and instruction ADD R1, R2, R3 is two-byte long; then what is the length of op-code field?*

Answer

The op-code field can have 16 – 3 – 3 – 3 = 7 bits.

Op-code 7-bit	R1 3-bit	R2 3-bit	R3 3-bit

2. *What is the maximum number of 0-address, 1-address and 2-address instructions if the instruction size is of 32-bit and 10-bit address field?*

Answer

In 0-address instructions, no address field is used. So, all 32-bit of the instruction size can be used as the size of op-code field. Therefore, maximum number of 0-address instructions is 2^{32}.

In 1-address instructions, one address field is used whose size is given as 10-bit. So, remaining 32 – 10 i.e. 22-bit can be used as op-code field. Therefore, maximum number of 1-address instructions is 2^{22}.

In 2-address instructions, two address fields are used; collectively they occupy 20-bit. So, remaining 32 – 20 i.e. 12-bit can be used as op-code field. Therefore, maximum number of 2-address instructions is 2^{12}.

3. *There are 58 processor registers, 7 addressing modes and 16K × 32 main memory. State the instruction format and size of each field if each instruction supports one register operand and one address operand.*

Answer

The processor has 58 registers, so 6-bit is used to specify each register uniquely (because 32 < 58 < 64).

No. of addressing modes is 7, so 3-bit is used in mode field in the instruction format (because 7 < 8).

The main memory size is 16K × 32; so to access each word of 32-bit in memory 14-bit address is generated.

Therefore, the op-code field requires 32 – 3 – 6 – 14 = 9-bit.

The instruction format will be as:

Op-code (9-bit)	Mode (3-bit)	Register addr. (6-bit)	Memory addr. (14-bit)

4. *The 32-bit value 40A75429 is stored to the location 1000. What is the value of the byte in address 1002 if the system is big-endian? If little-endian?*

Answer

In big-endian assignment, the most significant byte is stored in lower address and least significant byte is stored in higher address. Therefore, the byte 54 will be stored in location 1002.

In little-endian assignment, the least significant byte is stored in lower address and the most significant byte is stored in higher address. Therefore, the byte A7 will be stored in location 1002.

5. *What are advantages of general-register based CPU organizations over stack based CPU organizations?*

Answer

There are three main advantages of general-register based CPU organizations over stack based CPU organizations.

(a) In general-register based CPU organizations, reading a register does not affect its content, whereas, in stack based CPU organizations, reading value from the top of the stack removes the value from the stack.

(b) In general register-based CPU organizations, any register from register file can be chosen to keep values while writing a program; whereas, in stack based CPU organizations, accessing values is limited by the LIFO (last-in first-out) nature of the stack.

(c) Since, fewer memory references are made by programs written in general register-based CPU organizations, the effective execution is faster than that in stack based CPU organizations, where generally stack is implemented by memory locations and locations are accessed in LIFO nature.

6. *Assuming that all registers initially contain 0, what is the value of R1 after the following instruction sequence is executed?*

```
MOV R1, #6
MOV R2, #5
ADD R3, R1, R1
SUB R1, R3, R2
MULT R3, R1, R1
```

Answer

The instruction MOV R1, #6 places value 6 into register R1 and instruction MOV R2, #5 places value 5 into register R2. The instruction ADD R3, R1, R1 performs addition of values 6 in R1 and 6 in R1 and result 12 is stored in register R3. The instruction SUB R1, R3, R2 subtracts value 5 in R2 from 12 in R3 and result 7 is stored in register R1. Last instruction MULT R3, R1, R1 multiplies values 7 with 7 (content of R1) and result 49 is put into register R3. Therefore, the value of R1, after the execution of all instructions in the sequence, is 7.

7. *What value remains on the stack after the following sequence of instructions?*

PUSH #3 (symbol # indicates direct value of the number)
PUSH #5
PUSH # 4
ADD
PUSH # 7
SUB
MULT

Answer

After first three push operations, the stack contains 4, 5, 3 starting from the top of the stack. Then the ADD operation automatically pops two top most values from the stack, which are here 4 and 5, and then added and sum value 9 is push on the top of the stack. Thus, now stack has 9 and 3. After PUSH #7, the stack contains 7, 9, 3 starting from the top. Next instruction SUB subtracts the top value of the stack from the next value down in the stack. So after SUB instruction execution, the value 9 – 7 i.e. 2 is push on the top of the stack. Finally, MULT instruction pops 2 and 3 from the stack and pushes 2×3 i.e. 6 on the stack.

8. *Why stack is not generally implemented using a processor's register file?*

Answer

The stack gives the illusion of having large storage space to the programmer. Due to the limited number of registers in CPU, a system that used only register file to implement its stack would only be able to push small amount of data onto the stack. Thus, the programmers would have to take care of the finite size of the register implemented stack, while writing the programs, which would make programming harder.

9. *Explain why a given (Infix) arithmetic expression needs to be converted to Reverse Polish Notation (RPN) for effective use of a stack organization.*

Answer

Stack-organized computers are better suited to postfix (RPN) notation than traditional infix notation. In infix notation, operator is placed between operands. In postfix notation, operator is placed after operands. For example, infix notation A * B becomes AB * in postfix notation.

Once an expression is recoded in postfix notation, converting it into a stack-based program is very easy. Starting from the left, each operand is replaced with a PUSH operation to place the operand on the stack and operator is replaced with the appropriate instruction to perform the operation.

10. *Convert the given arithmetic expression to RPN Notation:*

(A + B ^ D) / (E – F) + G

Answer

Conventionally, three levels of precedence for the usual five binary operators as:

Highest: Exponentiation (^)
Next highest: Multiplication (*) and division (/)
Lowest: Addition (+) and subtraction (–)

Based on the priorities of operators, the given expression can be written as

((A + (B ^ D))/(E – F)) + G

Now, the equivalent postfix (RPN) expression can be evaluated as follows:

((A + (BD ^))/(EF –)) + G
⇒ ((ABD ^ +)/(EF –)) + G
⇒ (ABD ^ + EF – /) + G
⇒ ABD ^ + EF – / G +

11. *How stack is useful in subroutine call?*

Answer

Subroutine is a self-contained sequence of instructions that can be called or invoked from any point in a program. When a subroutine is called, a branch is made to the first executable instruction of the subroutine. After the subroutine has been executed, a return is made to the instruction following the point at which it was called. Consider the following code segment:

```
MAIN ()
{
------------
------------
CALL SUB1 ()
Next instruction
------------
}
SUB1 ()
{
 ------------
 ------------
 RETURN
}
```

After CALL SUB1 () has been fetched, the program counter (PC) contains the address of the "Next instruction" immediately following CALL. This address of PC is saved on the stack, which is the return address to main program. PC then contains address of the first executable instruction of subroutine SUB1 () and processor continues to execute its codes. The control is returned to the main program from the subroutine by executing RETURN, which pulls the return address (i.e. address of Next instruction) off the stack and puts it in the PC.

Since the last item stored on the stack is the first item to be removed from it, the stack is well suited to nested subroutines. That is, a subroutine is able to call another subroutine and this process can be repeated many times. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in last-in-first-out order. This suggests that the return addresses associated with subroutine calls should be pushed onto a stack.

12. *Suppose it takes 7 ns to read an instruction from memory, 3 ns to decode the instruction, 5 ns to read the operands from register file, 2 ns to perform the computation of the instruction and 4 ns to write the result into the register. What is the maximum clock rate of the processor?*

Answer

The total time to execute an instruction = (7 + 3 + 5 + 2 + 4) ns = 21 ns. That is, an instruction cycle takes 21 ns.

The time to execute an instruction by processor must be greater than the clock cycle time of the processor. Therefore, the maximum clock rate = 1/cycle time

$$= 1 / (21 \times 10^{-9}) \text{ Hertz}$$
$$= 1000/21 \text{ MHz } (1 \text{ MHz} = 10^6 \text{ Hz})$$
$$= 47.62 \text{ MHz}$$

13. *What do you mean by instruction cycle, machine cycle and T states?*

Answer

Instruction cycle: The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

Machine cycle: A machine cycle consists of necessary steps carried out to perform the memory access operation. Each of the basic operations such as fetch or read or write operation constitutes a machine cycle. An instruction cycle consists of several machine cycles.

T-states: One clock cycle of the system clock is referred to as T-state. A machine cycle consists of several T-states.

14. *A two-byte relative mode branch instruction is stored in memory location 1000. The branch is made to the location 87. What is the effective address?*

Answer

Since the instruction is two-byte, the content of PC is 1000 + 2 = 1002 after the instruction fetch. The effective branch address is calculated by adding the content of PC with address part of instruction (offset) which is 87. Thus the effective address is 1002 + 87 = 1089. Thus, after the instruction execution, the program branches to 1089 address in memory.

15. *What is load-store architecture? What are advantages and disadvantages of this architecture over other general register based architectures?*

Answer

In load-store architecture, only two instructions – load and store can access the memory system. In other general register based architectures, not only load and store, other instructions can access the operands in memory system.

Advantages:

(a) To implement a program, small number of instructions can be used.
(b) By minimizing the set of instructions that can access the memory system makes design of control unit simpler.
(c) By limiting the accesses to memory system increases the overall performance of the machine.

Disadvantages:

(a) As only two instructions can access memory; the length of the programs increases and thus storing the programs requires large memory.
(b) Large and complex instructions are difficult to programming.

16. *The first two bytes of a 2M × 16 main memory have the following HEX values:*
Byte-0 is FE
Byte-1 is 01
If these bytes hold a 16-bit 2's complement integer, what is its actual decimal value if:
(a) memory is big-endian?
(b) memory is little-endian?

Answer

(a) For big-endian memory access, the value of integer is = $FE01_{16}$ = 1111 1110 0000 0001_2 = -511_{10}
(b) For little-endian memory access, the value of integer is = $01FE_{16}$ = 0000 0001 1111 1110_2 = 510_{10}

17. *A digital computer has a memory unit with 24-bit per word. The instruction set consists of 150 different operations. All instructions have an operation code part (op-code) and an address part (allowing for only one address). Each instruction is stored in one word of memory.*
(a) How many bits are needed for the op-code?
(b) How many bits are left for the address part of the instruction?
(c) What is the maximum allowable size for memory?
(d) What is the largest unsigned binary number that can be accommodated in one word of memory?

Answer

(a) Since $2^7 < 150 < 2^8$, 8 bits are required for the op-code field.
(b) Since the word size is 24-bit, therefore the number of bits used in the address part of the instruction = 24 – 8 = 16-bit
(c) Since address field size of an instruction is 16-bit, the maximum allowable size for memory is 2^{16}, or 32M.
(d) The largest unsigned number in one word of memory consists of 24 ones, therefore its value = $2^{24} - 1$.

18. *What is meant by pipeline architecture? How does it improve the speed of execution of a processor?*

Answer

Pipelining is a technique of decomposing a sequential task into subtasks, with each subtask being executed in a special dedicated stage (segment) that operates concurrently with all other stages. Each stage performs partial processing dictated by the way the task is partitioned. Result obtained from a stage is transferred to the next stage in the pipeline. The final result is obtained after the instruction has passed through all the stages. All stages are synchronized by a common clock.

Several instructions are executed in different stages of the pipeline simultaneously. Thus, executions of several instructions are overlapped in the pipeline, giving the increased rate at which instructions execute.

19. *Consider the execution of a program of 20000 instructions by a linear pipeline processor with a clock rate 40 MHz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-order executions are ignored. Calculate the speed-up of the pipeline over its equivalent non-pipeline processor, the efficiency and throughput.*

Answer

Given,

No. of instructions (tasks) n = 20000

No. of stages k = 5

Clock rate f = 40 MHz

Clock period $\tau = 1/f = 1 / (40 \times 10^6)$ sec

$$\text{Speed up } S_p = \frac{n \times k}{k + (n - 1)}$$

$$= \frac{20000 \times 5}{5 + (20000 - 1)}$$

$$= 4.99$$

$$\text{Efficiency } \eta = \frac{n}{k + (n - 1)}$$

$$= \frac{20000}{5 + (20000 - 1)}$$

$$= 0.99$$

$$\text{Throughput } \omega = \frac{\eta}{\tau}$$

$$= 0.99 \times (40 \times 10^6) \text{ instructions per second}$$

$$= 39.6 \text{ MIPS}$$

20. *Deduce that the maximum speed-up in a k-stage pipeline processor can be k. Is this maximum speed-up always achievable? Explain.*

Answer
Speed-up is defined as

$$S_p = \frac{\text{Time to execute n tasks in k-stage non-pipeline processor}}{\text{Time to execute n tasks in k-stage pipeline processor}}$$

Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n - 1)]$ units, where k clock periods (cycles) are needed to complete the execution of the first task and remaining $(n - 1)$ tasks require $(n - 1)$ cycles. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can enter the pipeline until the previous task finishes. The clock period of the pipeline processor is τ.

Thus $$S_p = \frac{n.k.\tau}{\tau[k + (n-1)]} = \frac{nk}{k + (n-1)}$$

The maximum speedup can be deduced for $n >> k$, that means $n \to \infty$.

$$\text{Max } S_p = \lim_{n\to\infty} \frac{k}{(k/n) + 1 - (1/n)}$$
$$= k$$

But this maximum speedup k is never fully achievable because of data dependencies between instructions, interrupts, program branches, etc.

21. *How do you derive clock cycle time of a pipelined processor from its unpipelined implementation?*

Answer
The cycle time of a pipelined processor is dependent on four factors: (i) the cycle time of the unpipelined processing, (ii) the number of pipeline stages, (iii) how evenly the ALU logic is divided among the stages, and (iv) the delay of the pipeline latches. If the logic can be divided evenly among the pipeline stages, the clock period of the pipelined processor is given by

$$\text{Cycle time}_{\text{pipelined}} = \frac{\text{Cycle time}_{\text{unpipelined}}}{\text{Number of pipelne stages}} + \text{Pipeline latch delay}$$

This is written, because each stage contains the same fraction of the original logic, plus one pipeline latch. As the number of pipeline stages increases, the pipeline latch delay becomes a greater and greater fraction of the cycle time, limiting the benefit of dividing a processor into a very large number of pipeline stages.

22. *Given a non-pipelined processor with 15 ns clock period. How many stages of pipelined version of the processor are required to achieve a clock period of 4 ns? Assume that the interface latch has delay of 0.5 ns.*

Answer
We can write the clock period of a pipeline processor as:

$$\text{Clock period of pipeline} = \frac{\text{Clock period of non-pipeline}}{\text{No. of pipeline stages}} + \text{Interface latch delay}$$

This can be written as:

$$\text{No. of pipeline stages} = \frac{\text{Clock period of non-pipeline}}{\text{Clock period of pipelne} - \text{Interface latch delay}}$$

Therefore, for our problem,

$$\text{No. of pipeline stages} = \frac{15}{4 - 0.5}$$

$$= 4.3$$

Since the number of stages in pipeline cannot be fraction, by rounding up gives the number of stages as 5.

23. *What is clock skewing? Describe.*

Answer

Ideally, we expect the clock periods to arrive at all stages (latches) at the same time. However, due to a problem known as clock skewing, the same clock pulse may arrive at different stages with a time offset of Δ. Let t_{max} be the time delay of the longest logic path within a stage and t_{min} that of the shortest logic path within a stage.

To avoid a race in two successive stages, we must choose maximum stage delay, $\tau_m \geq t_{max} + \Delta$ and interface latch delay $\tau_l \leq t_{min} - \Delta$. These constraints translate into the following bounds on the clock period when clock skew takes effect:

$$\tau_l + t_{max} + \Delta \leq \tau \leq \tau_m + t_{min} - \Delta$$

In the ideal case, $\Delta = 0$, $t_{max} = \tau_m$, and $t_{min} = \tau_l$. Thus, we have $\tau = \tau_m + \tau_l$, which is consistent with the definition of clock period without the effect of clock skewing.

24. *Identify all of the RAW, WAR and WAW hazards in the instruction sequence:*

DIV R1, R2, R3
SUB R4, R1, R5
ASH R2, R6, R7
MULT R8, R4, R2
BEQ R9, #0, R10
OR R3, R11, R1

Also identify all of the control hazards in the sequence.

Answer

RAW hazard exists between instructions:

DIV and SUB
ASH and MULT
SUB and MULT
DIV and OR

WAR hazard exists between instructions:

DIV and ASH
DIV and OR

There is no WAW hazard.

There is only one control hazard between the BEQ and OR instructions.

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:

(i) An instruction set
   (a) is a complete collection of instructions understood by the CPU
   (b) for a machine is different from an instruction set of another machine
   (c) is common for all machines
   (d) both (a) and (b).

(ii) What is the correct definition of the term "instruction set"?
   (a) The range of op-codes which a CPU is programmed to recognize.
   (b) The list of instructions in memory which forms the program being executed.
   (c) A specific subroutine of a program, run if conditions relating to the flag register are satisfied.
   (d) The process by which a single instruction of a program is executed.

(iii) Use of short instructions in a program leads to
   (a) large program (b) small program (c) fast execution (d) both (a) and (c).

(iv) Where does the control unit look in order to find the address of the next instruction to be fetched?
   (a) Memory Address Register (MAR)
   (b) Instruction Register (IR)
   (c) Memory Buffer Register (MBR)
   (d) Accumulator (AC)

(v) Which of the following is NOT one of the three stages of the instruction execution cycle?
   (a) Decode (b) Fetch (c) Flag (d) Execute

(vi) The addressing mode of an instruction is resolved by
   (a) ALU (b) DMA controller (c) CU (d) program.

(vii) An one-address machine has accumulator organization based CPU, supports two addressing modes and has 8 registers. All arithmetic and logic instructions use accumulator and one destination register only. If the instruction length is 80 bit, what is the length of op-code field in the instruction?
   (a) 3 (b) 4 (c) 5 (d) 6.

(viii) The stack organized computers use instructions of
   (a) zero-address (b) one-address (c) two-address (d) three-address.

(ix) The stack-pointer (SP) register holds the address of an element in the stack. What is the position of that element in stack?
   (a) bottom (b) any position (c) top (d) none.

(x) The length of an instruction in the instruction set depends on the number of
   (a) addresses in the address field
   (b) bits in op-code field
   (c) bits in mode field
   (d) bits in any special field used in the instruction.

(xi) Suppose we have 32-bit data word $94A203DE_{HEX}$ to be stored in memory from address 0 onwards. In little-endian the word is stored in the order as (address [data]):

(a) 0 [DE], 1[03], 2[A2], 3[94]
(b) 0 [94], 1[A2], 2[03], 3[DE]
(c) 0 [A2], 1[94], 2[DE], 3[03]
(d) 0 [94], 1[A2], 2[03], 3[DE].

(xii) Suppose an instruction cycle consists of four major phases:
1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.

The addressing mode of instructions is resolved in (a) step (1) (b) step (2) (c) step (3) (d) step (4).

(xiii) Immediate operand
(a) is a variable fetched from the processor register fast
(b) is a constant and is part of an instruction
(c) is an operand, which takes no time to fetch with
(d) can be used to load the registers.

(xiv) With indirect addressing, how many memory lookups are needed to obtain the required data?
(a) Two (b) Zero (c) Three (d) One

(xv) A computer uses words of size 32-bit. The instruction
(a) must always be fetched in two cycles with one byte in each cycle
(b) must always be fetched in one cycle with 2 bytes in each cycle
(c) may or may not be of one byte length
(d) must be of 2 bytes length.

(xvi) A two-byte long assembly language instruction BR 09 (branch instruction) stored at location 1000 (all numbers are in HEX). The program counter (PC) holds the address
(a) 1002 (b) 1000 (c) 1009 (d) 100B.

(xvii) The unit which performs the task of fetching, decoding, managing the execution and then storing results is
(a) ALU (b) CU (c) Memory unit (d) I/O processor

(xviii) An indexed addressed instruction having an address field with all 0 bits is equivalent to:
(a) register direct mode (b) register indirect mode
(c) memory indirect mode (d) memory indirect mode.

(xix) From quickest to slowest instruction execution time, order these three addressing modes:
(a) Immediate, indirect, direct (b) Indirect, immediate, direct
(c) Direct, indirect, immediate (d) Immediate, direct, indirect

(xx) The addressing mode in which the data is directly available as an operand is
(a) immediate (b) relative (c) indexed (d) direct

(xxi) Name the addressing mode in which the base address is with the displacement and the effective address is calculated
(a) immediate (b) relative (c) indexed (d) direct

(xxii) The CPI value for a RISC processor is
(a) 1 (b) 2 (c) 3 (d) none.

(xxiii) Name the processor in which large number of complex instructions is available
(a) RISC (b) CISC (c) Hardwired (d) Micro-programmed

Group B

2. What is meant by "instruction set of a machine"? What are the different parameters that determine the design of an instruction set of a machine? Explain.
3. What are the different fields of an instruction generally used in a computer?
4. Classify the CPU organizations. Give one example for each.
5. What are 0, 1, 2 and 3-address machines? Give examples.
6. Write the assembly language procedures using 0, 1, 2 and 3 – address instructions to implement the instruction: $X = (A + B \times C)/(D - E)$, using suitable assumptions.
7. What is the importance of stack in some CPU organization?
8. What is reverse Polish notation? Convert the following expression from infix to reverse Polish: $(A \times B) + (C \times D) + E$
9. What is the difference between big-endian and little-endian?
10. Why do we need various addressing modes?
11. What is instruction cycle? Describe with the help of flowchart.
12. Briefly describe following addressing modes with example:
    (a) implied
    (b) immediate
    (c) stack
    (d) register
    (e) register indirect
    (f) auto increment
    (g) direct
    (h) indirect
    (i) relative
    (j) base
    (k) indexed.
13. What is the main difference between base and index register addressing modes?
14. What are the characteristics of a good instruction set?
15. Compare and contrast between CISC and RISC. Give two examples of each.
16. Why every computer is associated with a set of general purpose registers?
17. What is the maximum number of 0-address, 1-address, 2-address instructions if the instruction size is of 32-bit and 10-bit address field?
18. There are 54 processor registers, 5 addressing modes and $8K \times 32$ main memory. State the instruction format and size of each field if each instruction supports one register operand and one address operand.
19. Why would it not be a good idea to implement a stack using a processor's register file?
20. What value remains on the stack after the following sequence of instructions?

```
PUSH #6 (symbol # indicates direct value of the number)
PUSH #8
PUSH # 4
ADD
```

```
PUSH # 12
SUB
MULT
```

21. What are the merits and demerits of fixed-length and variable-length instruction formats?
22. Is there any possible justification for an instruction with two op-codes?
23. A relative mode branch instruction is stored in memory location 530 (decimal). The branch is made to the location 30 (decimal). What is the effective address?
24. What are the different address modes supported by (a) PowerPC machine (b) SPARC machine? Briefly discuss each.
25. What is pipelining? How does it improve the performance of a processor?
26. How do you define the clock period of a pipeline?
27. What are the different parameters to measure the performance of a pipeline? Explain.
28. Define speedup of a pipeline processor. Show that the maximum speedup of a k-stage pipeline is k. Is this maximum speedup always achievable? Explain.
29. What is pipeline hazard? What are the different pipeline hazards? Briefly explain.

CHAPTER

6
Design of Control Unit

6.1 INTRODUCTION

A CPU can be considered as a collection of three major components:

- Arithmetic logic unit (ALU)
- Control unit (CU)
- Register set.

The ALU performs arithmetic and logic operations on the data values stored in registers, where as the sequence of operations is controlled by the CU.

The function of the CU is to control system operations by routing the selected data items to the selected processing hardware of ALU at the right time. A control unit's responsibility is to activate the associated processing hardware units by generating a set of signals that are synchronized with a master clock. The inputs to the control unit are the master clock, status information from the processing units and command signals from the external devices like memory, I/O system. The outputs produced by the typical control unit are the signals that activate the processing units and responses to an external environment (such as operation complete and operation aborted) due to exceptions (integer overflow or underflow).

A control unit performs the following responsibilities:

- Instruction interpretation
- Instruction sequencing

During the interpretation phase, the control unit reads instructions from the memory (using the PC register as a pointer). It then resolves the instruction type and addressing mode, gets the necessary operands and routes them to the appropriate functional units of the execution unit. Required signals are then issued to the different units of ALU to perform the desired operation and the results are routed to the specific destination. Thus, this phase is done in "instruction decoding" step of the instruction cycle.

During the sequencing phase, the control unit finds the address of the next instruction to be executed and loads it into the PC. Thus, this phase is done in "instruction fetch" step of the instruction cycle.

In this chapter we will discuss basic operational concept of the CU and design techniques of it.

6.2 PRIMARY CONCEPTS

The preliminary concepts forming the basis for control unit design are the register transfer micro-operations and their analytical descriptions. In Section 3.4, we have discussed the register transfer in details. There we saw that register transfer occurs under some predetermined control condition(s), which is (are) generated by the control unit. Here we take another example for further illustration.

Example 6.1 If t = 0 and x = 1 then A ← B
else A ← D

where A, B and D are 4-bit registers. Here, depending on the x and t values, 4-bit content of B or D register is copied to A register.

Such a selective register transfer micro-operation can be expressed as follows:

C: A ← B

C′: A ← D [C′ indicates complement of C].

Where C = t′ ∧ x and C′ = (t′ ∧ x)′ = t ∨ x′.

A hardware implementation for this transfer is shown in Fig. 6.1.

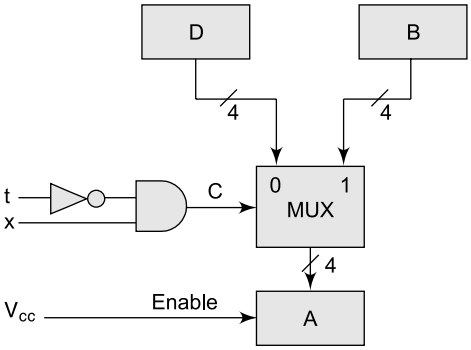


Figure 6.1 *Hardware implementation of "if t = 0 and x = 1 then A ← B else A ← D"*

The B register is selected by the MUX if condition C = 1; otherwise register D is selected as source register.

6.3 DESIGN METHODS

Control units are designed in two different ways:

- Hardwired approach
- Microprogramming approach

When the control signals are generated using conventional sequential logic design techniques, the control unit is said to be hardwired. The sequential logic circuit generates specific sequences of control signals in response to externally supplied instructions. Logic gates, flip flops, decoders and other digital circuits are used to implement hardwired control organization. As name suggests, if the design has to be changed or modified, a hardwired control unit requires changes in the wiring among the various components.

In the microprogrammed approach, all control functions that can be simultaneously activated are grouped to form control words stored in a separate ROM memory called the control memory. From the control memory, the control words are fetched one at a time and the individual control fields are routed to various functional units to activate their appropriate circuits. The desired task is performed by activating these circuits sequentially.

Figure 6.2 depicts the general structures of hardwired and microprogrammed control units.

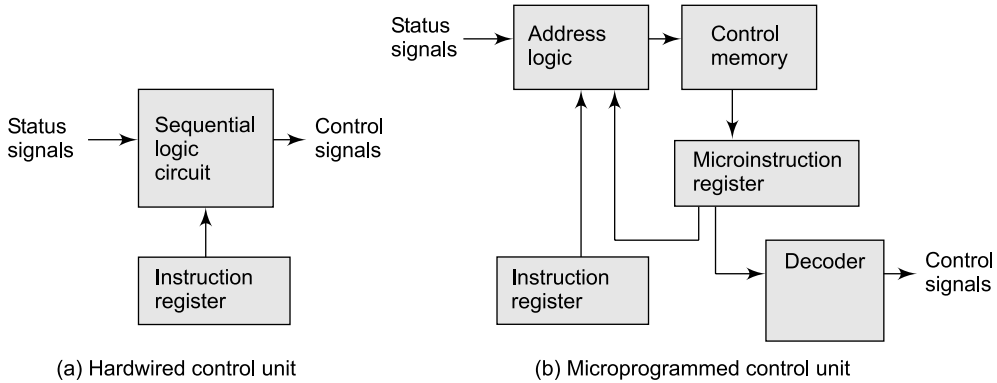


Figure 6.2 *General structures of two approaches of control unit design*

Comparison Between Two Methods The microprogramming approach is more expensive than hardwired approach. In microprogramming approach, a control ROM memory is needed.

The main advantage of microprogramming is it provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions). With microprogramming, many additions and changes are made by simply changing the microprogram in the control memory, as the control signals are embedded in a kind of two-level software called *firmware*. A small change in the hardwired approach may lead to redesigning the entire system.

Now-a-days microprogramming is accepted as a standard tool to design the control unit of a computer. For example, processors such as IBM 370, PDP-11 and Intel 80×86 family have a microprogrammed control units. However, some olden day computers like Zilog's 16-bit microprocessor Z8000 still use a hardwired control unit.

6.3.1 Hardwired Control Design

The hardwired control unit design includes the following summarized steps:

1. State the task to be performed.
2. Suggest a trial processing section.
3. Devise a register-transfer description of the algorithm based on the processing section outlined in the step 2.
4. Describe major characteristics of the hardware components to be used in the processing section.
5. Establish the design of the processing section by providing necessary control signals.
6. Provide a block diagram of the controller section.
7. Generate the state diagram of the controller section with different control states.
8. Specify the characteristics of the hardware components to be used in the controller section.
9. Give the complete design of the controller and draw a logic diagram of the final circuit.

Example 6.2 *Multiplier Control Unit* Let us take one example to illustrate the design procedure.

Step 1 (Statement of task) Implement a Booth's multiplier to multiply two signed 4-bit numbers.

We know that Booth's procedure inspects a pair of multiplier bits (refer Section 2.7.3) and performs one of the following actions:

Multiplier bits inspected Q[i]	Q[i-1]	Action (in ith position)
0	0	None
0	1	Add M
1	0	Subtract M
1	1	None

Step 2 To design Booth's multiplication method, the processing section is proposed in the Fig. 6.3. As mentioned in the Section 2.7.3, the 4-bit register M will hold the multiplicand. The multiplier Q register is 5-bit wide. Initially, the high-order 4-bit of this register will hold the 4-bit multiplier. The least-significant bit of this register is initialized with the fictitious 0. The 4-bit adder/subtractor unit is used to perform the operations A + M or A – M. The result produced by this hardware unit is always stored to the 4-bit accumulator A. Here, the accumulated partial product stored in (AQ) register pair is shifted right. The L register is used to keep track of the iteration count. In the example case, this register is initialized with decimal 4 and thus L is 3-bit in length and decremented by 1 after the completion of each iteration. Thus, the algorithm terminates when L reaches decimal 0. When L equals to decimal 0 (termination of algorithm), the high- and low-order 4 bits of the final product are found in the registers A and Q, respectively. The 4-bit data buses - Inbus and Outbus, are used to transfer data into and out of the processing section respectively.

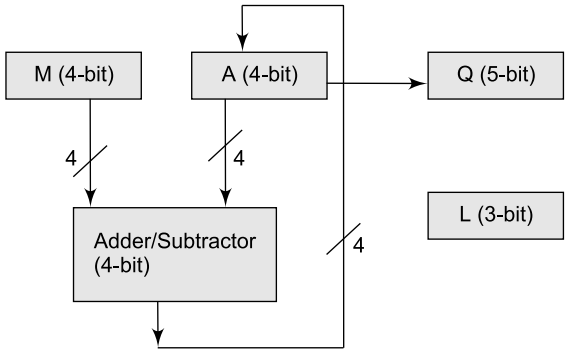


Figure 6.3 *Processing section for 4 × 4 Booth's multiplication*

Step 3 For 4 × 4 Booth's multiplication algorithm, a register transfer description is devised next. Q [0: –1] is used to indicate the low-order 2 bits of the Q register (Initially Q[0] indicates the lsb of Q register and Q[–1] indicates a fictitious 0). Similarly, Q[3:0] indicates the high-order 4 bits of the Q register. The last step, Go to HALT, introduces an infinite loop after the algorithm is completed.

```
Registers: M[4], A[4], Q[5], L[3];
Buses: Inbus[4], Outbus[4];
START       A ← 0, M ← Inbus, L← 4
            Q[3:0] ← Inbus, Q[–1] ← 0;
LOOP        If Q[0:–1] = 01 then go to ADD
            If Q[0:–1] = 10 then go to SUB
```

```
                    Go to RSHIFT;
ADD                 A ← A + M;
                    Go to RSHIFT;
SUB                 A ← A – M;
RSHIFT              ASR (AQ), L ← L–1;
                    If L ≠ 0 then go to LOOP
                    Outbus ← A;
                    Outbus ← Q[3:0];
HALT                Go to HALT;
```

Step 4 The processing section contains three main elements:

- 4-bit adder/subtarctor.
- General-purpose registers.
- Tri-state buffers.

The operational characteristics of these three elements are provided in Fig. 6.4. By introducing the proper values to control inputs C, L, R and D, four operations (clear, parallel load, right shift and decrement) can be performed. A clock circuit synchronizes all these operations. The 4-bit adder/subtractor can be implemented using a

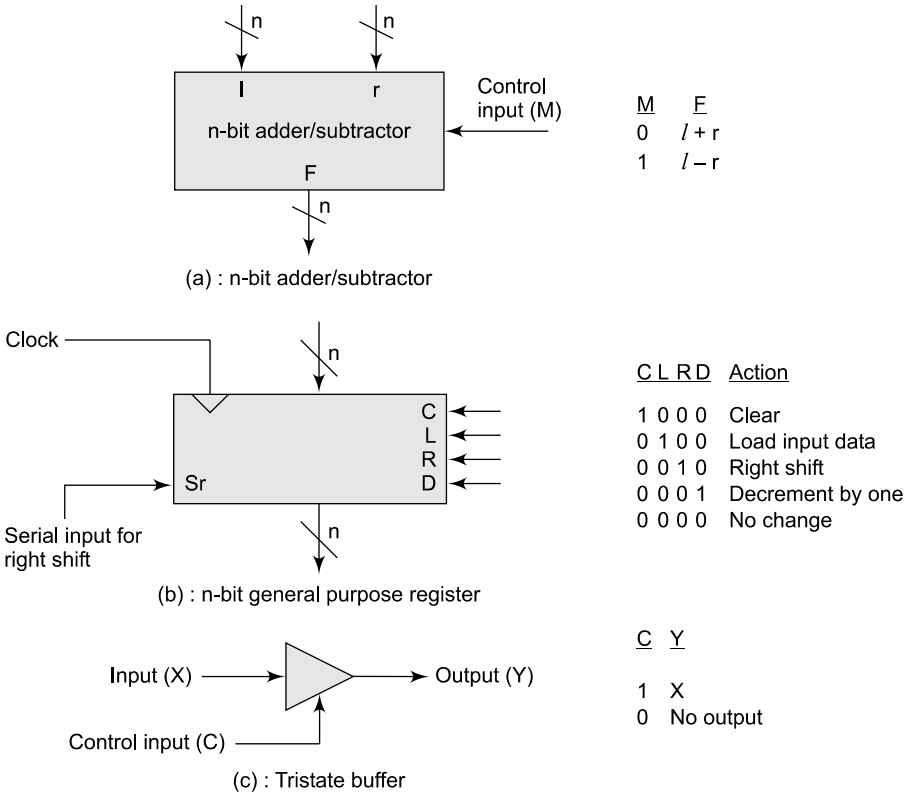


Figure 6.4 *Different major components in processing section*

4-bit parallel adder chip and four XOR gates (which is implemented in Section 3.5.4). To build a general-purpose register, standard flip-flops and gates can be used. The tri-sate buffers are used to control the data transfer to the outbus.

Step 5 There are 10 control signals required: C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9 and their tasks are provided next. The micro-operations A ← 0, M ← Inbus, L ← 4 will be executed when C_0, C_1, C_2 are held high. Similarly, other micro-operations are performed by activating proper signals. Though, the signals' tasks are self-explanatory. A detail logic diagram of the processing section along with various control-signal points is shown in Fig. 6.5. In the diagram, a total of 8 tri-state buffers are needed, out of which a set of 4 buffers are controlled by C_8 and another set of 4 buffers are controlled by C_9. Though two buffers are shown in Fig. 6.5, one is controlled by C_8 and other is controlled by C_9.

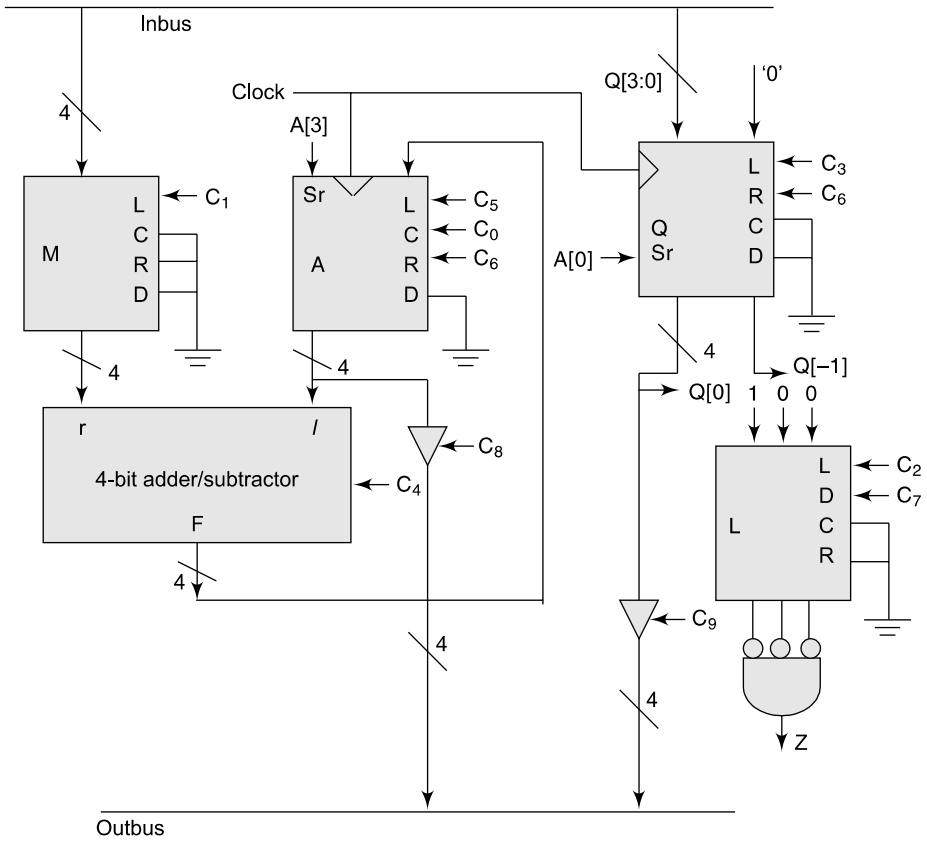


Figure 6.5 *4 × 4 Booth's multiplier processing section*

C_0: A ← 0
C_1: M ← Inbus
C_2: L ← 4
C_5: A ← F
C_6: ASR (AQ)
C_7: L ← L – 1

C_3: Q[3:0] ← Inbus
Q[–1] ← 0
C_4: F ← l + r
C_4': F ← l – r
C_8: Outbus ← A
C_9: Outbus ← Q[3:0]

Step 6 The processing section intermediately generates three outputs Q[0], Q[–1] and Z. When the content of the L register becomes 0, then Z register is set to 1. These outputs are status outputs and are used as inputs to the controller to allow the controller to decide the next step of the algorithm. With this information a block diagram for the controller section can be generated, as shown in the Fig. 6.6.

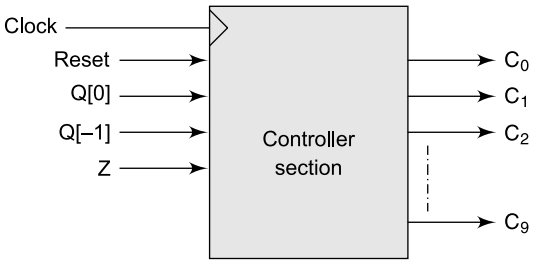


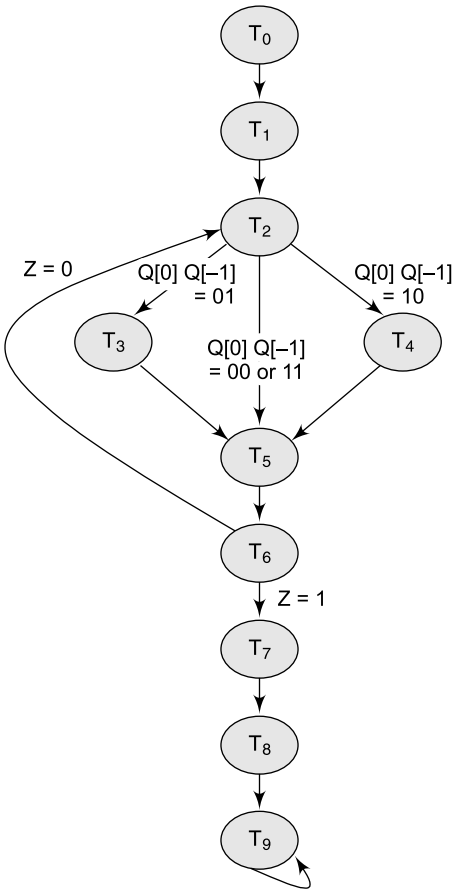
Figure 6.6 *Block diagram of the Booth's multiplier controller*

Step 7 The controller has 5 inputs and 10 control outputs. The clock input is used to synchronize the controller's activities. The Reset input is asynchronous input used to reset the controller so that a new computation can start. The controller must initiate a set of micro-operations in a specified sequence controlled by the clock input. Thus, it is recognized as a sequential logic circuit. The state diagram for the Booth's multiplier controller is shown in the Fig. 6.7.

Initially, the controller is in the state T_0. At this time the control signals C_0, C_1 and C_2 are generated at high state. Thus, the operations A ← 0, M ← Inbus and L ← 4 are performed. The controller then moves to the state T_1 in the next clock cycle to perform the operation Q[3:0] ← Inbus and Q[–1] ← 0. The controller moves to the state T_9 only when a computation is completed and the controller stays in that state infinitely until a Reset input forces the controller to switch to the state T_0 and a new computation step starts.

The states are generated in the state diagram according to the following rules:

- If the two or more micro-operations are independent of each other and can be completed within one clock cycle, they are grouped into one state. For example, micro-operations A ← 0, M ← Inbus and L ← 4 are independent to each other. That is why they are executed in one clock period. If these micro-operations cannot be performed within the selected T_0 clock period, then either clock period duration needs to be increased or the micro-operations have to be divided into a sequence of micro-operations.
- Generally a new state is introduced for conditional testing. For example, the conditional testing of the bit pair Q[0] Q[–1] introduces the new state T_2 in Fig. 6.7.



Control State	*Operation performed*	*Signals to be activated*
T_0	A ← 0, M ← Inbus L ← 4	C_0, C_1, C_2
T_1	Q[3:0] ← Inbus Q[–1] ← 0	C_3
T_2	None	None
T_3	A ← A + M	C_4, C_5
T_4	A ← A – M	C_5
T_5	ASR (AQ) L ← L – 1	C_6, C_7
T_6	None	None
T_7	Outbus ← A	C_8
T_8	Outbus ← Q[3:0]	C_9
T_9	None	None

Figure 6.7 *Controller's state diagram and description*

There are 10 states in the controller state diagram. Ten non-overlapping timing signals (T_0 to T_9) must be generated for the controller to perform the Booth's algorithm. But, only one will be high for a clock pulse.

Step 8 Since minimum 10 clock cycles (periods) are needed for 10 states in the controller, a mod-16 counter and a 4-to-16 decoder are used to generate the clock periods and to select one of the control signals at the appropriate state respectively. The characteristic of the mod-16 counter is discussed in the Fig. 6.8.

Step 9 The controller and its logic diagram are shown in the Fig. 6.9. The main component of this design is the sequence controller (SC) hardware, which sequences the controller as indicated in the state diagram in Fig. 6.7. The truth table of SC is derived from the controller's state diagram and is shown in Table 6.1.

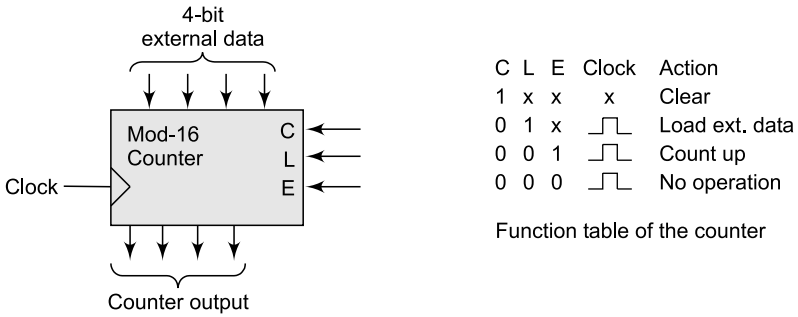


Figure 6.8 *Characteristics of the counter used in the controller design*

Table 6.1 *Truth Table of Sequence Controller*

Z	*Q[0]*	*Q[–1]*	T_2	T_3	T_6	T_9	*Load*	*External data*			
							(L)	d_3	d_2	d_1	d_0
x	0	0	1	x	x	x	1	0	1	0	1
x	1	1	1	x	x	x	1	0	1	0	1
x	1	0	1	x	x	x	1	0	1	0	0
x	x	x	x	1	x	x	1	0	1	0	1
0	x	x	x	x	1	x	1	0	0	1	0
x	x	x	x	x	x	1	1	1	0	0	1

For example, consider the logic involved in deriving the first entry of the SC truth table. Observe that the mod-16 counter is loaded with the specified external data if the counter control inputs C and L are 0 and 1 respectively. From the controller's state diagram, it can be observed that if the present control state is T_2 (counter output = 0010) and if the bit pair inspected is 00 (i.e., Q[0] Q[–1] = 00) then the next state will be T_5. When these input conditions occur, the counter must be loaded with external data value 0101 (When counter output = 0101, then T_5 =1). Therefore, the SC generates load (L) = 1 and d_3 d_2 d_1 d_0 = 0101.

Using the same reasoning, the last entry of the SC truth table is obtained. From the controller's state diagram, it can be observed that if the present state is T_9, the next control state will be same T_9 (it stays in the infinite loop). The SC must generate the outputs load (L) =1 and d_3 d_2 d_1 d_0 = 1001 to obtain the desired state sequence. Similarly, other entries of the SC truth table are derived.

The counter will automatically count up in response to the clock pulse (because the enable input E is fixed with 1), when the counter load control input L = 0. In other words, the sequential execution flow will be there when load input (L) = 0. Such normal sequencing activities are desirable in the following situations:

- Present state is: T_0, T_1, T_4, T_5, T_7, or T_8.
- Present state is: T_2 and Q[0] Q[–1] = 01.
- Present state is: T_6 and Z = 1.

These results suggest that the SC should not affect the counter load control input L. Hence these possibilities are excluded from the SC truth table. The SC must exercise control only when there is a need for the counter to deviate from its normal counting sequence.

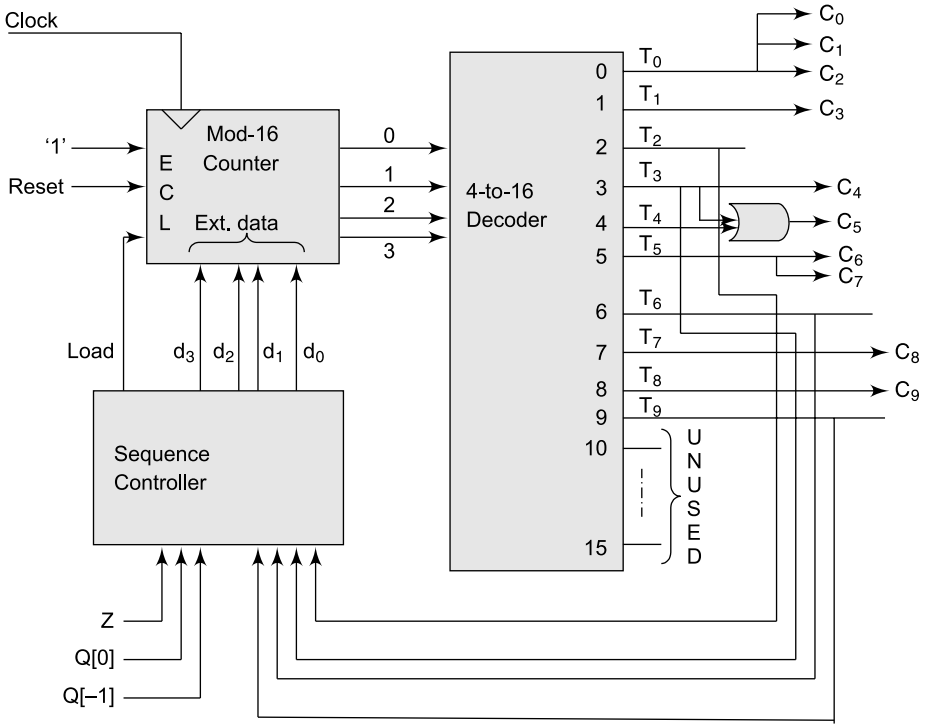


Figure 6.9 *Logic diagram of the Booth's multiplier controller*

CPU Hardwired Control Unit The general CPU hardwired control unit is depicted in Fig. 6.10. The inputs to the control unit are content of instruction register (IR), the clock signal, the status (flag) and control signals. Consider, the instruction register (IR). The control unit will perform different functions for different instructions. There must be a unique logic input for each op-code to simplify the control unit logic. In order to perform this function, a decoder is used which takes an encoded input and produces a single output. To synchronize all micro-operations, a clock unit is used which issues a repetitive sequence of pulses. The clock cycle time must be long enough to allow the propagation of signals along data paths and through processing units. The control unit generates different control signals at different time units within a single instruction cycle. For this reason, a counter is used as input to the control unit to generate different timing states T_1, T_2, T_3 and so on for different control signals. At the end of the instruction cycle, the control unit must reinitialize the counter at T_1 using special Reset signal.

6.3.2 Microprogrammed Control Unit

Microprogramming is a modern concept used for designing a control unit. It can be used for designing control logic for any digital system. As stated earlier, a microprogrammed control unit's control words are held in a separate ROM memory called the control memory (CM). Each control word contains signals to activate one or more micro-operations. When these words are retrieved in a sequence, a set of micro-operations are activated that will complete the desired task.

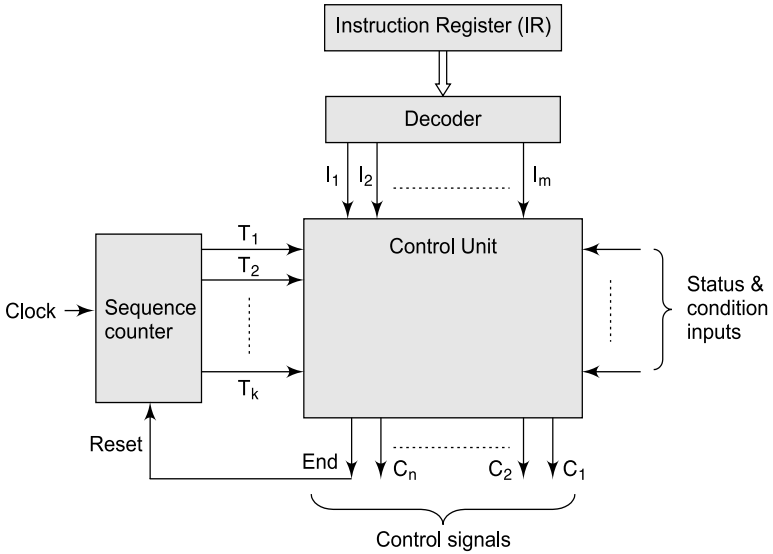


Figure 6.10 *Block diagram of CPU hardwired control unit*

Like conventional program, retrieval and interpretation of the control words are done. The instructions of a CPU are stored in the main memory. They are fetched and executed in a sequence. The CPU can perform different functions simply by changing the instructions stored in the main memory. Similarly, the control unit can execute a different control operation by changing the contents of the CM. Hence, the microprogrammed approach offers greater flexibility than its hardwired counterpart, since this approach is based on the programming concept giving an easy way for altering the contents of the CM.

Usually, all microinstructions have three important fields:

- Control field
- Next-address field
- Condition for branching.

A control memory in addition to the conventional main memory is used in the microprogramming approach. Thus, it is desired to give more emphasis on minimizing the length of the microinstruction. The length of the microinstruction decides the size of the control memory, as well as the cost involved with this approach. The following factors are directly involved with the length of a microinstruction:

- How many micro-operations can be activated simultaneously (the degree of parallelism).
- The control field organization.
- The method by which the address of the next microinstruction is specified.

Several micro-operations can be executed simultaneously. A single microinstruction with a common op-code can be specified for all micro-operations executed in parallel. This allows short microprograms to be written. The length of microinstruction increases, whenever there is a need for parallelism. Similarly, short microinstructions have limited capability in expressing parallelism. Since massive parallelism is not possible using short microinstructions, the overall length of a microprogram written using these instructions will increase.

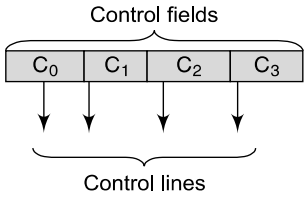
There are various ways to organize control information. A simple way to organize the control field would be to have one bit for each control line that controls the processing unit, allowing full parallelism,

and there is no need for decoding the control field. However, this method cannot use the control memory efficiently when it is impossible to invoke all control operations simultaneously.

Consider the example shown in Fig. 6.11. Assume there are four registers A, B, C and D whose contents are transferred to the destination register R when the appropriate control line is activated:

C_0: R ← A
C_1: R ← B
C_2: R ← C
C_3: R ← D

Since there is only one destination register R, it is not possible to allow more than one transfer at any given time. If one bit is allocated for each control in the control field, the result will appear as shown next:



C_0	C_1	C_2	C_3	Action
1	0	0	0	R ← A
0	1	0	0	R ← B
0	0	1	0	R ← C
0	0	0	1	R ← D
0	0	0	0	No operation

This method of organizing control fields is known as *unencoded* format.

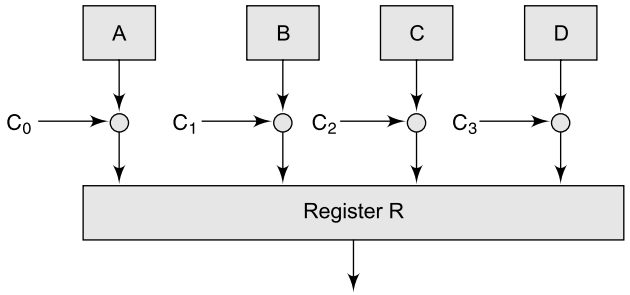


Figure 6.11 *A register can be loaded from four independent sources*

In the previous format, there are only five valid binary patterns. However, five distinct binary patterns can be represented by using only 3 bits according to the basic switching theory. Such an arrangement is illustrated in Fig. 6.12.

The control information is encoded into a 3-bit field, and a decoder is needed to get the actual control information, in Fig. 6.12. The relationship between the encoded and the actual control information is specified as follows:

E_2	E_1	E_0	*Action*
0	0	0	No operation (NOP)
0	0	1	R ← A
0	1	0	R ← B
0	1	1	R ← C
1	0	0	R ← D

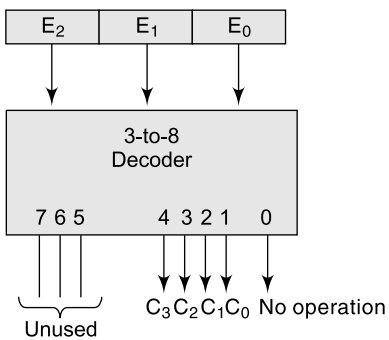
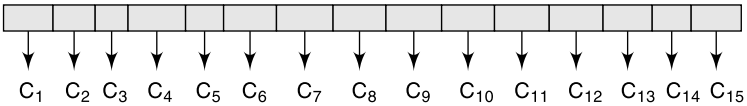


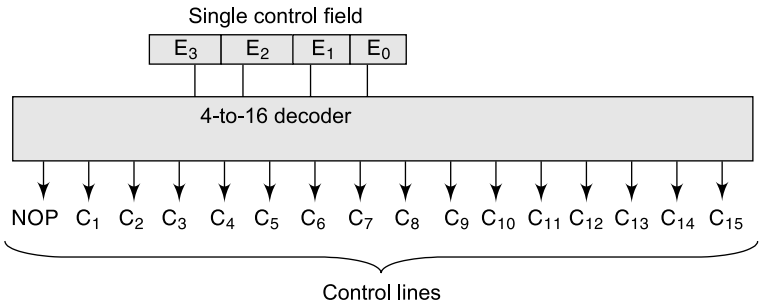
Figure 6.12 *Encoded control arrangement*

This way of organizing the control field is known as *encoded* format method, which specifies to a short control field and short microinstructions. One extra hardware element, namely decoder is needed for such a reduction. Therefore, a compromise must be made.

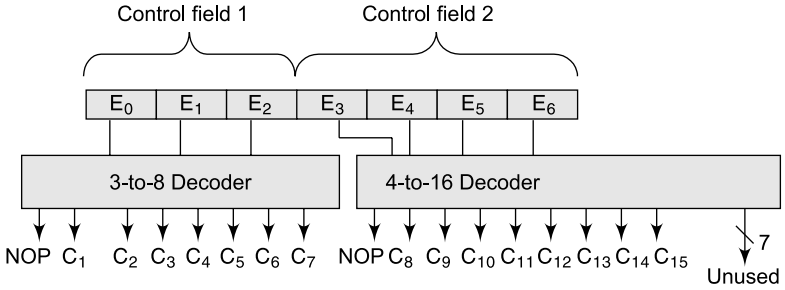
Fifteen control lines can be specified in a fully unencoded form as shown next:



The same information can be specified in the fully encoded form as shown next:



In the first and second cases, the sizes of the control field are 16 and 4 bits, respectively. However, the second approach needs a 4-to-16 decoder to generate the actual control signals. Thus, the control information is given by a partial encoding, as a measure of compromise and is shown next:



The control signals are partitioned into disjoint groups of control fields, so two signals of different groups can be enabled in parallel. For the above example, the control signals are partitioned into two groups as:

Group 1: $C_1\ C_2\ C_3\ C_4\ C_5\ C_6\ C_7$.
Group 2: $C_8\ C_9\ C_{10}\ C_{11} C_{12}\ C_{13}\ C_{14}\ C_{15}$.

With the above grouping, C_8 can be activated simultaneously with C_1 or C_2 but not C_1 and C_2. Using one 3-to-8 and one 4-to-16 decoders, the actual control signals are generated. In the last case, the control field requires 7 bits E_0 to E_6. This technique is mixed one which lies between the unencoded and fully encoded approaches.

Horizontal and Vertical Microprogramming Based on the length of microinstructions, the microprogramming method is either horizontal or vertical.

Having an individual bit for each control signal in the microinstruction format is known as a horizontal microinstruction, as shown in Fig. 6.13. The unencoded method, discussed above, is

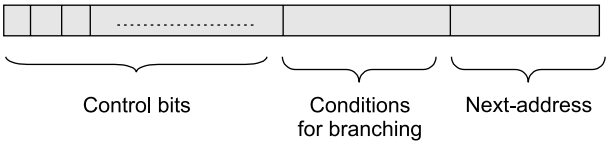


Figure 6.13 *Horizontal microinstruction*

usually followed in horizontal organization. Each bit in microinstruction activates one control signal. Several control signals can be simultaneously generated by a single microinstruction. The length of the microinstruction is large. Horizontal microinstructions have the following general attributes:

- Long formats.
- Ability to express a high degree of parallelism.
- Very little encoding of the control information.

In the IBM 360/model 50, the microinstructions used in control unit follow horizontal format.

In a vertical microinstruction (Fig. 6.14), a single field can produce an encoded sequence. The encoded technique is followed in this organization. The vertical microprogram technique takes more time for generating the control signals due to the decoding time and also more microinstructions are needed. But the overall cost is less since the microinstructions are small in size. The horizontal microprogram releases faster control signals but the control memory size is huge due to increased word length. Thus, the vertical microinstructions are characterized by:

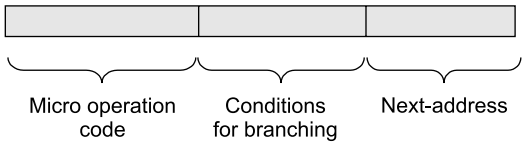


Figure 6.14 *Vertical microinstruction*

- Short formats.
- Limited ability to express parallel microoperations.
- Considerable encoding of the control information.

In the IBM 370/model 145, the microinstructions used in control unit follow vertical format.

Structure of Microprogrammed Control Unit We now describe the design of a typical microprogrammed control unit. The architecture of a typical modern microprogrammed control unit is shown in Fig. 6.15. This architecture was proposed by Maurice Wilkes in 1953.

The various components used in Fig. 6.15 are summarized next.

Control Memory Buffer Register (CMBR) The function of CMBR is same as the MBR (memory buffer register) of the main memory. It is basically a latch and acts as a buffer for the microinstructions retrieved from the CM. Typically, each microinstruction has three fields as:

Condition select	Branch address	Control functions

The condition select field selects the external condition to be tested. The output of the MUX will be 1, if the selected condition is true. The MPC will be loaded with the address specified in the branch address field of the microinstruction, because the output of the MUX is connected to the load input of the microprogram counter (MPC). However, the MPC will point to the next microinstruction to be executed, if the selected external condition is false. Thus, this arrangement allows conditional branching. The control function field of the microinstruction may hold the control information in an encoded form which thus may require decoders.

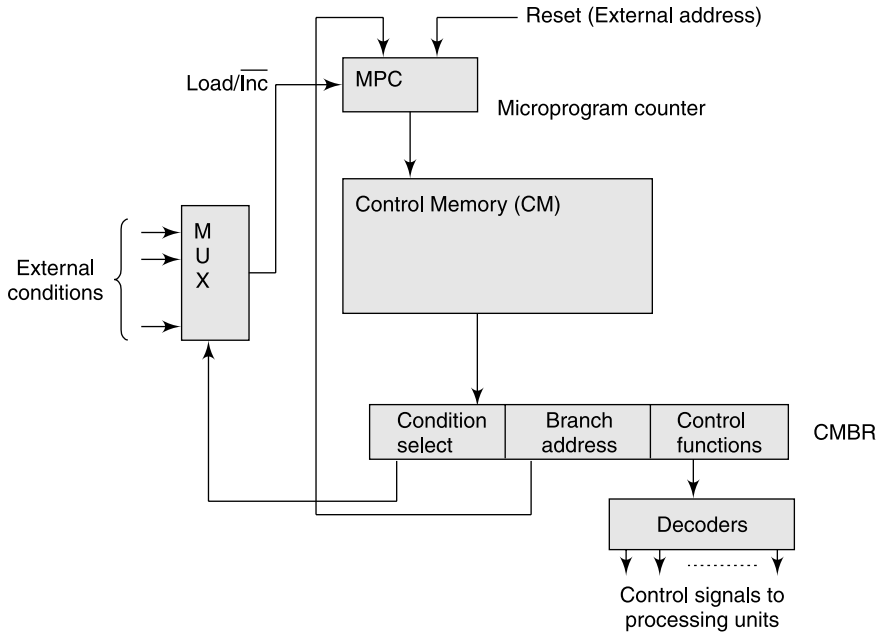


Figure 6.15 *General-purpose microprogrammed control unit*

Microprogram Counter (MPC) The task of MPC is same as the PC (program counter) used in the CPU. The address of the next microinstruction to be executed is held by the MPC. Initially, it is loaded from an external source to point to the starting address of the microprogram to be executed. From then on, the MPC is incremented after each microinstruction fetch and the instruction fetched is transferred to the CMBR. However, the MPC will be loaded with the contents of the branch address field of the microinstruction that is held in the CMBR, when a branch instruction is encountered.

External Condition Select MUX Based on the contents of the condition select field of the microinstruction, this MUX selects one of the external conditions. Therefore, the condition to be selected must be specified in an encoded form. Any encoding leads to a short microinstruction, which implies a small control memory; hence the cost is reduced. Suppose two external conditions X_1, X_2 are to be tested; then the condition-select and actions taken are summarized next:

Condition select	Action taken
00	No branching
01	Branch if X_1 =1
10	Branch if X_2 =1
11	Always branching (unconditional branching)

The multiplexer has four inputs V_0, V_1, V_2, V_3 where V_i is routed to the multiplexer's output when the condition select field has decimal equivalent i. Hence we require $V_0 = 0$, $V_1 = X_1$, $V_2 = X_2$, $V_3 = 1$ to control the loading of microinstruction branch addresses into MPC.

Example 6.3 The design of a typical microprogrammed control unit is discussed now. Consider the implementation of a microprogrammed control unit for the 4 × 4 Booth's multiplication. First step will be writing of the microprogram in a symbolic form, then next task will be generating of control signals and architecture of the control unit, and lastly we will give the microprogram in binary for 4 × 4 Booth's multiplication. The symbolic microprogram for 4 × 4 Booth's multiplication is as follows:

Control Memory Address		Control word
0	START	A ← 0, M ← Inbus, L ← 4
1		Q[3:0] ← Inbus, Q[–1] ← 0;
2	LOOP	If Q[0: –1] = 01 then go to ADD;
3		If Q[0: –1] = 10 then go to SUB;
4		Go to RSHIFT;
5	ADD	A ← A + M;
6		Go to RSHIFT;
7	SUB	A ← A – M;
8	RSHIFT	ASR (AQ), L ← L–1;
9		If Z = 0 then go to LOOP;
10		Outbus ← A;
11		Outbus ← Q[3:0];
12	HALT	Go to HALT;

In this task, three conditions, Q[0] Q[–1] = 01, Q[0]Q[–1] = 10 and Z = 0, are tested. Here, Z corresponds to the L register. When L ≠ 0, Z is reset to 0, otherwise Z is set to 1. These three conditions are applied as inputs to the condition select MUX. Additionally, to take care of no-branch and unconditional-branch situations, logic 0 and logic 1 are applied as data inputs to this MUX, respectively. Therefore, The MUX is able to handle five data inputs and thus must be at least an 8:1. The size of the condition select field must be 3 bits in length.

With this design, the condition select field may be interpreted as below:

Condition select	Action taken
000	No branching
001	Branch if Q[0] = 0 and Q[–1] = 1
010	Branch if Q[0] = 1 and Q[–1] = 0
011	Branch if Z = 0
100	Unconditional branching

With these details, the size of the control word is calculated as follows:

Size of a control word = size of the condition select field + size of the branch address field + number of control functions

= 3 + 4 + 10

= 17 bits.

Hence, the sizes of the CMDB and CM are 17 bits and 13 × 17, respectively. The complete hardware organization of the control unit and control signals is shown in Fig. 6.16.

C_0 : A ← 0
C_1 : M ← Inbus
C_2 : L ← 4
C_3 : Q[3:0] ← Inbus, Q[–1] ← 0
C_4 : F ← l + r
C_4' : F ← l – r
C_5 : A ← F
C_6 : ASR (AQ)
C_7 : L ← L – 1
C_8 : Outbus ← A
C_9 : Outbus ← Q[3:0]

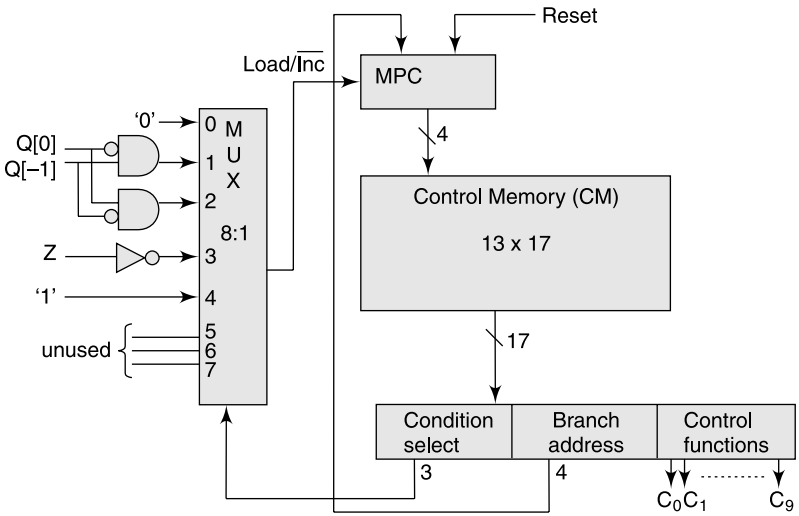


Figure 6.16 *Microprogrammed 4 × 4 Booth's multiplier control unit*

Finally, the generation of binary microprogram stored in the CM will be discussed. There exists a control word for each line of the symbolic program listing. For example, consider the first line (0^{th}) of the symbolic listing program mentioned previously. This instruction, being a simple load instruction, introduces no branching. Therefore, the condition-select field should be 000. Thus, the contents of the branch address field are irrelevant. However, without any loss of generality, the contents of this field can be reset to 0000. For this instruction, three micro-operations C_0, C_1 and C_2 are activated. Therefore, only the corresponding bit positions in the control function fields are set to 1. This results in the following binary microinstruction:

Condition select	Branch address	Control function
000	0000	1110000000

The binary microinstruction corresponding to third line of the symbolic microprogram does not activate any micro-operation. But, it branches to 5th location after checking one condition (Q[0: –1] = 01). So, the condition-select field should be 001 and the branch address will be 0101. Therefore the complete binary microinstruction corresponding to this instruction is as follows:

Condition select	Branch address	Control function
001	0101	0000000000

Continuing in this way, the complete binary microprogram for 4 × 4 Booth's multiplier can be produced, as in the Table 6.2.

Table 6.2

Control Memory address		*Condition select (3-bit)*	*Branch address (4-bit)*	*Control function (10-bit)*
In decimal	In binary			$C_0C_1 \ldots C_9$
0	0000	000	0000	1110000000
1	0001	000	0000	0001000000
2	0010	001	0101	0000000000
3	0011	010	0111	0000000000
4	0100	100	1000	0000000000
5	0101	000	0000	0000110000
6	0110	100	1000	0000000000
7	0111	000	0000	0000010000
8	1000	000	0000	0000001100
9	1001	011	0010	0000000000
10	1010	000	0000	0000000010
11	1011	000	0000	0000000001
12	1100	100	1100	0000000000

CPU Microprogrammed Control Unit Here, we want to design microprogrammed control unit for a basic accumulator-based CPU as shown in Fig. 6.17. This CPU consists of a data processing unit (DPU) designed to execute the set of 10 basic single-address instructions listed in Table 6.3. The instructions are assumed to be fixed length and to execute on data words of the same fixed length, say 32 bits. The function of control unit (CU) is to manage the control signals linking the CU to the DPU, as well as the control signals between the CPU and the external memory M.

In order to design the CU, first we have to identify the relevant control actions (micro-operations) needed to process the given instruction set using the hardware from Fig. 6.17. The instruction execution behavior of the CPU is shown in Fig. 6.18 using a flowchart. All instructions require a common instruction-fetch phase, followed by an execution-phase that varies with each instruction type. The content of the program counter (PC) is copied to the memory address register (AR) in the fetch phase. A memory read operation is then executed, which transfers the instruction word I to memory data register (DR); that is expressed by DR ← M(AR). Op-code of I is transferred to the

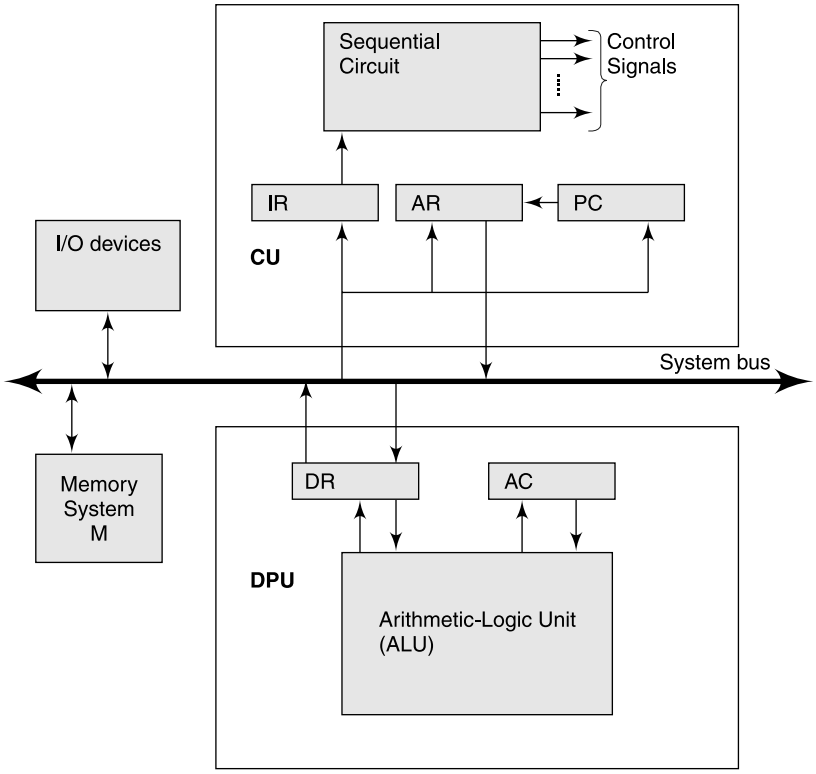


Figure 6.17 *An accumulator based CPU organization*

instruction register (IR), where it is decoded; at the same time PC is incremented to point to the next consecutive instruction in M.

The op-code type of current instruction determines the subsequent operations to be performed. For example, the store instruction ST X is executed in three steps: the address field X of ST X is transferred to AR, the content of the accumulator (AC) is transferred to DR and finally the memory write operation M(AR) ← DR is performed. The branch-on-zero instruction BZ adr is executed by first checking AC. If AC ≠ 0, no action is taken; if AC = 0, the address field adr, which is in DR(ADR), is transferred to PC, thus performing the branch operation. From Fig. 6.18, it can be seen that instruction fetching takes three cycles, while instruction execution takes from one to three cycles.

The control signals and control points needed by the CPU are determined implicitly by the micro-operations appearing in the flow chart. A suitable set of control signals for the CPU and their functions are listed in the Table 6.4. Figure 6.19 shows the approximate points of the corresponding control points in both the CU and DPU. The control signals generated by CU are used as control inputs to different units of DPU and memory. Three basic groups can be created for these control lines, as:

— Operation select: C_2, C_9, C_{10}, C_{11}, C_{12}.
— Memory control: C_1, C_8.
— Data transfer: C_0, C_3, C_4, C_5, C_6, C_7.

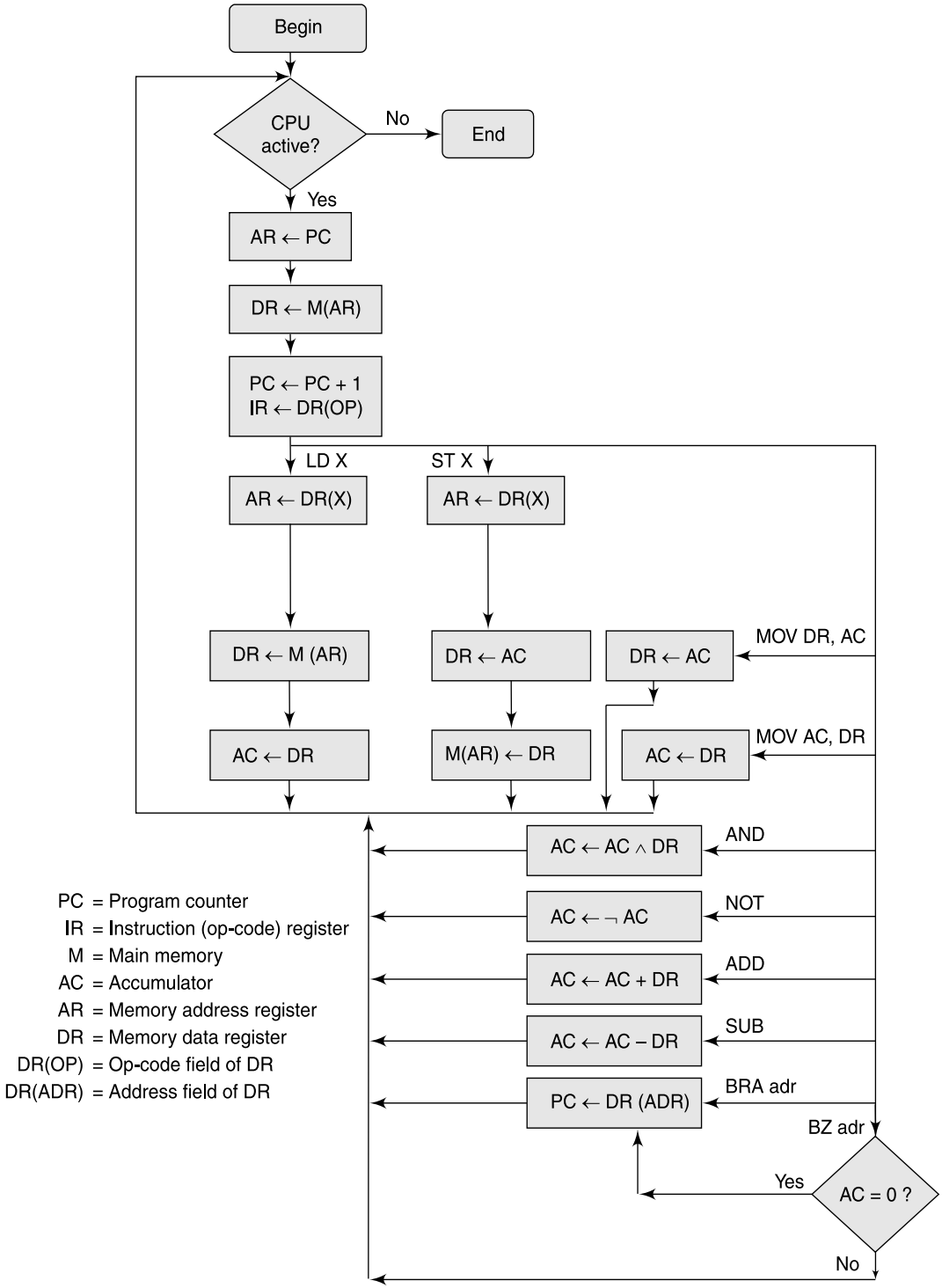


Figure 6.18 *Flowchart of the accumulator based CPU*

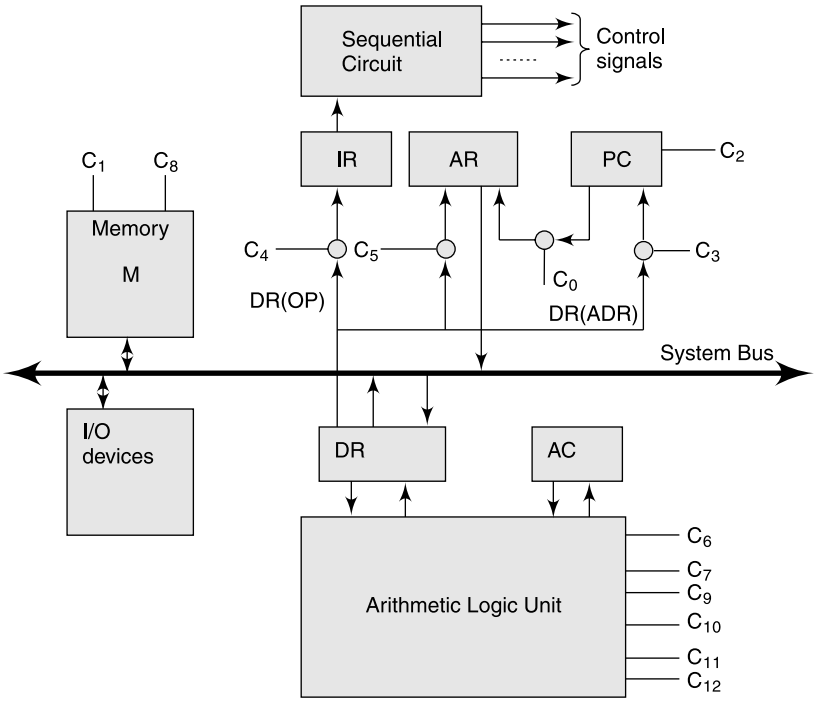


Figure 6.19 *Control points for accumulator based CPU*

Here memory control refers to the external memory M. Many of the control signals route information between the CPU's internal data and control registers.

Table 6.3 *Instruction set for CPU in Fig. 6.17*

Type	*Symbolic format*	*Assembly format*	*Remark*
Data transfer	AC ← M(X)	LD X	Load X from memory into AC
	M(X) ← AC	ST X	Store content of AC in memory as X
	DR ← AC	MOV DR, AC	Copy (transfer) content of AC to DR
	AC ← DR	MOV AC, DR	Copy content of DR to AC
Data processing	AC ← AC ∧ DR	AND	AND DR to AC bit-wise
	AC ← ¬ AC	NOT	Complement content of AC
	AC ← AC + DR	ADD	Add DR to AC
	AC ← AC – DR	SUB	Subtract DR from AC
Program control	PC ← M(adr)	BRA adr	Jump to instruction with address adr.
	if AC = 0 then PC ← M(adr)	BZ adr	Jump to instruction with address adr if AC = 0.

Table 6.4 *Control signals for accumulator based CPU*

Control signal	*Operation controlled*
C_0	AR ← PC
C_1	DR ← M(AR)
C_2	PC ← PC + 1
C_3	PC ← DR(ADR)
C_4	IR ← DR(OP)
C_5	AR ← DR(ADR)
C_6	DR ← AC
C_7	AC ← DR
C_8	M(AR) ← DR
C_9	AC ← AC ∧ DR
C_{10}	AC ← ¬ AC
C_{11}	AC ← AC + DR
C_{12}	AC ← AC – DR

Each machine instruction is executed by a microprogram stored in CM which acts as a real-time interpreter for the instruction, in a microprogrammed CPU. The set of microprograms that interprets a particular instruction set or machine language ML is called an *emulator* for ML.

Now we want to write an emulator for the target instruction set whose members are LD, ST, MOV1, MOV2, AND, NOT, ADD, SUB, BRA and BZ. In Fig. 6.19, the micro-operations that implement the various instructions appear, from which the required microprograms are deduced. The op-code of each instruction identifies the microprogram selected to emulate the instruction. Hence, the microprogram's starting address is determined by the content of the instruction register (IR). We will use the unmodified content of IR as the microprogram address for the current instruction. We will further assume that each microinstruction can specify a branch condition, a branch address that is used only if the branch condition is satisfied and a set of control fields defining the micro-operations to be performed. These microinstruction fields can easily be adapted to a variety of formats (horizontal, vertical or mixed), as discussed earlier.

A complete emulator for the given instruction set in symbolic form is listed in Fig. 6.20. The conversion of each microinstruction to binary code can be done easily. For this conversion, first the control signals need to be identified, depending on the micro-operations listed in the Fig. 6.20. The next task is to identify the external status conditions to be tested by the multiplexer in microprogrammed control unit. A distinct microprogram for each of the ten possible instruction execution cycles and another microprogram called FETCH have constituted this emulator. The FETCH microprogram controls the instruction-fetch cycle. The "go to IR" micro-operation is implemented by MPC ← IR, which transfers control to the first microinstruction in the microprogram that interprets the current instruction. Either such branch operations can be included in a general operate-with-branching format or separate branch microinstructions can be defined, depending on the microinstruction format chosen. In Fig. 6.20, it is assumed that MPC is the default address source for microinstructions and is incremented automatically in every clock cycle.

```
FETCH:  AR ← PC;
        DR ← M(AR);
        PC ← PC +1, IR ← DR(OP);
        go to IR;
LD:     AR ← DR(ADR)
        DR ← M(AR);
        AC ← DR, go to FETCH
ST:     AR ← DR(ADR);
        DR ← AC;
        M(AR) ← DR, go to FETCH;
MOV1:   DR ← AC, go to FETCH;
MOV2:   AC ← DR, go to FETCH;
AND:    AC ← AC ∧ DR, go to FETCH;
NOT:    AC ← ¬ AC, go to FETCH;
ADD:    AC ← AC + DR, go to FETCH;
SUB:    AC ← AC – DR, go to FETCH;
BRA:    PC ← DR(ADR), go to FETCH;
BZ:     if AC = 0 then PC ← DR(ADR), go to FETCH:
```

Figure 6.20 *A symbolic microprogrammed emulator for a small instruction set*

Microprogram Sequencer The microprogramming approach is systematic, flexible, and less error-prone. Advances in IC technology have made LSI designers think of a general solution for implementing a microprogrammed CPU. A microprogrammed CPU has two major activities to be performed:

1. Fetching and interpreting microinstructions.
2. Generating the next address of the microinstruction to be retrieved.

The first task is assumed by the control memory and the associated circuit elements.

Designers have replaced the next address generation of a microprogrammed control unit with a single LSI component called a *microprogram sequencer,* which checks certain bits in the microinstruction and finds the next address for the control memory. The sequencer contains a microprogramrned counter (MPC) and circuit elements necessary to perform functions such as address incrementing, address sequencing for subroutine calls, returns, and conditional branching. At the present time, many microprogrammed control units manufactured use the following components:

- A control memory (ROM or RAM)
- A microprogram sequencer

The general organization of a microprogrammed control unit constructed using a microprogram sequencer is shown in Figure 6.21.

The microinstruction in this figure is assumed to have the following format:

Condition select field	Branch-type field	Branch address field	Control function field

An additional field called *branch type field* (BT) has been included which generates the next address, a 3-bit field (B_2 B_1 B_0). Its interpretation is summarized as follows:

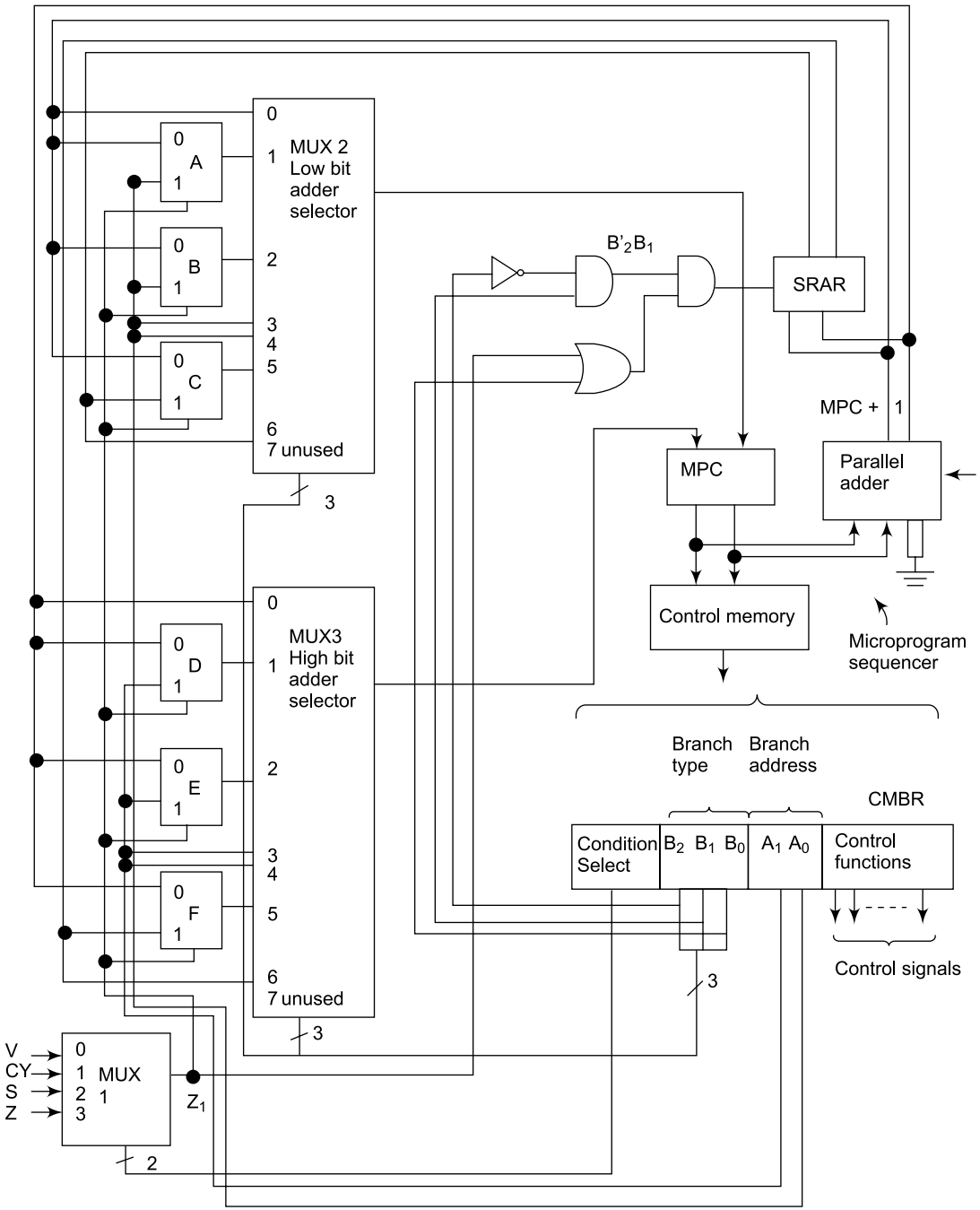


Figure 6.21 *Structure of a Microprogrammed CPU using Microprogram Sequencer*

$B_2\ B_1\ B_0$	*Action performed*
000	No branching and, therefore, the next address is MPC + 1
001	Branch to the address specified in the branch address field if the condition selected by the condition select field is true. The MPC is loaded with the contents of the branch address field
010	Branch to the subroutine if the condition is met. Here, the return address is saved in a subroutine-return address register (SRAR). MPC is loaded with the address of subroutine, or MPC is loaded with MPC + 1. The single register SRAR is replaced with a set of registers to form a stack, if it is desired to implement nested or recursive subroutine calls
011	Unconditional branch to subroutine loads the MPC in the address of subroutine and the return address is saved in SRAR
100	Unconditional branch loads the MPC with the contents of the branch address field
101	Conditional return from subroutine loads the MPC with SRAR, if the condition is met; otherwise, it is loaded with MPC + 1
110	Unconditional return from subroutine. Always loads the MPC with the return address saved in SRAR

The MPC is loaded with one of the following sources:

- MPC + 1 computed by the parallel adder
- Branch address field of the microinstruction
- SRAR

As shown from the branch-type field $B_2\ B_1\ B_0$ listed before, the SRAR is loaded only when $B_2'\ B_1\ B_0'\ Z_1 + B_2'\ B_1\ B_0$ is 1; where Z_1 refers to the output of the MUX1. This Boolean equation is true when there is an unconditional branch to the subroutine or when a branch to a subroutine with the selected condition is true.

This equation can also be simplified as follows:

$$
\begin{aligned}
&B_2'\ B_1\ B_0'\ Z_1 + B_2'\ B_1\ B_0 \\
&\quad = B_2'\ B_1\ (B_0'\ Z_1 + B_0) \\
&\quad = B_2'\ B_1\ [(B_0 + B_0')\ (B_0 + Z_1) && \text{Since } yz + x = (x + y)\ (x + z) \\
&\quad = B_2'\ B_1\ (B_0 + Z_1) && \text{Since } B_0 + B_0' = 1
\end{aligned}
$$

This hardware implementation has a 2-bit microprogram counter and three multiplexers, MUX1, MUX2, and MUX3. The MUX1 selects the desired condition, and multiplexers MUX2 and MUX3 select the low- and high-order address bits, respectively. First, the control word is fetched to the CMBR. Depending on the value of the BT, the next address is computed and loaded into the MPC. Consider some specific numeric examples:

If $B_2\ B_1\ B_0 = 000$, line 0 of MUX2 and MUX3 are selected. Since these lines are outputs of the 2-bit parallel adder, outputs of the multiplexer MUX2 and MUX3 transfer the value MPC + 1 into MPC.

Assume $B_2\ B_1\ B_0 = 001$ and the contents of the condition select field are 01. The binary pattern of the BT selects line 1 of the multiplexers MUX2 and MUX3. These lines are outputs of multiplexers A and D. These two multiplexers are controlled by output Z_1 of the condition-select multiplexer (MUX1). Line 0 of the A and D multiplexers are connected to the output of the 2-bit parallel adder with a value of MPC + 1. Line 1 of the A and D multiplexers are connected to branch address $A_1\ A_0$. The condition select field selects the CY flag. Therefore, if CY = 1, then $Z_1 = 1$. Line 1 of the A and D multiplexers are selected, and branch address $A_1\ A_0$ is transferred to the MPC. However, if CY = 0, the MPC is loaded with MPC + 1.

If the contents of the BT and the condition-select field are 010 and 10, respectively, line 2 of the multiplexers MUX2 and MUX3 are selected. These lines are outputs of multiplexers B and D; both of these multiplexers are controlled by output Z_1 of the MUX1. Therefore; if S-flag = 1, lines B and D are selected, and MPC is loaded with the starting address of the subroutine specified in A_1 A_0. The SRAR is simultaneously loaded with the return address MPC + 1 (since B'_2 B_1 (B_0 + Z_1) is true). If S-flag = 0, however, the MPC is loaded with MPC + 1. The SRAR remains unaffected (because B_0 Z_1 = 00).

The other operating modes of this circuit can be explained in a similar manner.

The microprogram sequencer described so far is simple in structure. A practical LSI sequencer offers more attractive features. Example of a popular microprogram sequencer is AM 2909, a 4-bit microprogram sequencer developed by Advanced Microdevices Incorporated.

Nanoprogramming A microprogam stored in a single control memory (CM) interprets an instruction fetch from memory, in most microprogrammed CPUs. However, in a few machines, the microinstructions do not directly issue the signals that control the hardware. Instead, they are used to access a second control memory called a *nanocontrol memory* (NCM) that directly controls the hardware. Therefore, there are two levels of control memories, a higher-level one termed a microcontrol memory (MCM) whose contents are microinstructions and the lower-level NCM that stores *nanoinstructions* (see Fig. 6.22).

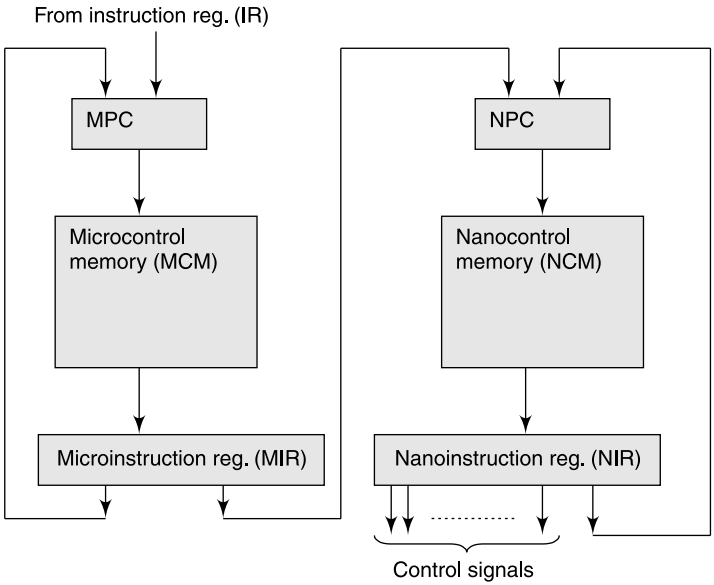


Figure 6.22 *Two level control organization for nanoprogramming*

Apparently, one may feel that two-level structure will increase the overall cost, but it actually improves the economy of the system by reducing the total space required.

The horizontal and vertical microprogramming concepts together give the concept of nanoprogramming. In reality, this method gives useful trade-offs between these two techniques. The nanoprogramming technique offers significant savings in space when a group of micro-operations occur many times in a microprogram.

Let us consider a nanoprogrammed computer in which control memory has dimension A × B. So the size of this memory is A*B bits to store the microprogram. Assume this microprogram has K (K < A) unique microinstructions. In nanocontrol memory (NCM) of size K × B, these K microinstructions are held. These K microinstructions occur once in the NCM. Each microinstruction in the original microprogram is replaced with an address that specifies the location of the NCM in which the original B-bit wide microinstruction is stored. Since the NCM has K addresses, only $\lceil \log_2 K \rceil$ bits are needed to specify one NCM address. Therefore, the size of each microinstruction in a two-level control memory is only $\lceil \log_2 K \rceil$ bits. This is shown in Fig. 6.23.

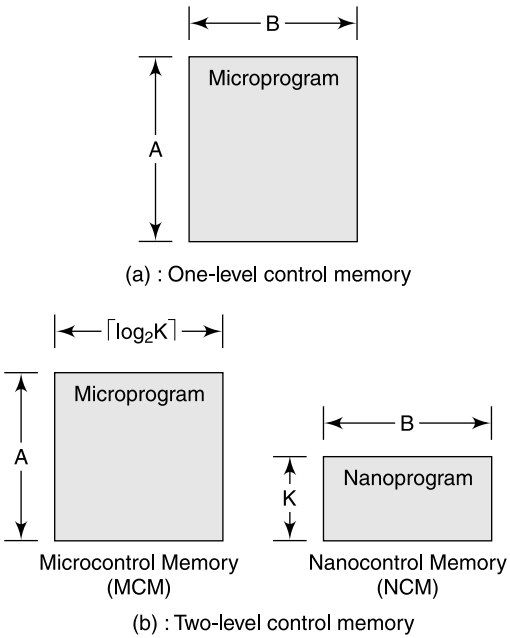


Figure 6.23 *Nanoprogramming concept*

The operation of a control unit using a two-level memory shown in Fig. 6.21 can be explained next. The first microinstruction from the microprogram of MCM is read. This microinstruction is actually the first address of the NCM. The content of this location in the NCM is the control word, which is transferred to the nanoinstruction register (NIR). The bits in this register are then used to control the gates for one cycle. After completion of the cycle, the second microinstruction is read from the MCM and the process continues. Therefore, two memory fetches (one for the MCM and the other for NCM) are required. The conventional control memory consists of a single memory; thus, one memory fetch is necessary. This reduction in control unit using single-level of memory is compensated by the cost of the memory when the same microinstructions occur many times in the microprogram. The main disadvantage of nanoprogramming is that a control unit using a NCM is slower than one using a conventional control memory, since the nanoprogramming concept consists of two-level memory.

The nanoprogramming concept was first used in the QM-1 computer designed around 1970 by Nanodata Corporation. It is also employed in the Motorola 680X0 microprocessors series.

Example 6.4 *Example of nanoprogramming* Let us take one practical example of the nanoprogramming concept: the nanomemory structure of the Motorola MC68000 16-bit microprocessor in Fig. 6.24. It has 640 microinstructions, out of which 280 are unique, as shown in Fig. 6.24. The contents of the MCM are pointers to the NCM. Each microinstruction of the MCM is of size = $\lceil \log_2 280 \rceil$ = 9 bits in length.

It can be seen that the MC68000 offers control memory savings. In the MC68000, the MCM is 640 × 9 bits and the NCM is 280 × 70 bits, since there are 280 unique microinstructions. If the MC68000 is implemented by using a single CM, this memory will have 640 × 70 bits. Therefore, the use of nanoprogramming saves a total of

= 640 * 70 – (640 * 9 + 280 * 70) bits

= 19, 440 bits.

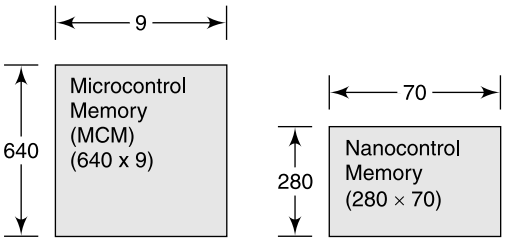


Figure 6.24 *MC68000 control unit's memory structure*

SOLVED PROBLEMS

1. *What are the different status flags in a processor?*

Answer

The processor uses one special register called *status register* to hold the latest program status. It holds 1-bit flags to indicate certain conditions that produced during arithmetic and logic operations. The bits are set or reset depending on the outcome of most recent arithmetic and logic operation. The register generally contains following four flags:

Carry (C): it indicates whether there is any end-carry from the most significant bit position.

Zero (Z): it indicates whether the result is zero or non-zero.

Sign (S): it indicates whether the result is positive or negative.

Overflow (V): it indicates whether the operation produces any overflow or not.

There may be other flags such as parity and auxiliary carry.

2. *What are the advantages and disadvantages of microprogram control unit over hardwired control unit?*

Answer

Advantages of microprogram control unit:

(a) It provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions). Logic gates, flip flops, decoders and other digital circuits are used to implement hardwired control organization.

(b) With microprogramming, many additions and changes can be made by simply changing the microprogram in the control memory. A small change in the hardwired approach may lead to redesigning the entire system.

Disadvantage of microprogram control unit:

The microprogramming approach is more expensive than hardwired approach. Since a control ROM memory is needed in microprogramming approach.

3. *Suppose there are 15 micro-instructions in control memory that generate the control signals for an instruction and it takes 7 micro-instructions to read an instruction from memory into instruction register IR and then to decode the instruction. Assuming a read of control memory address occurs in 1 ns, what will be the time taken by the processor for the instruction?*

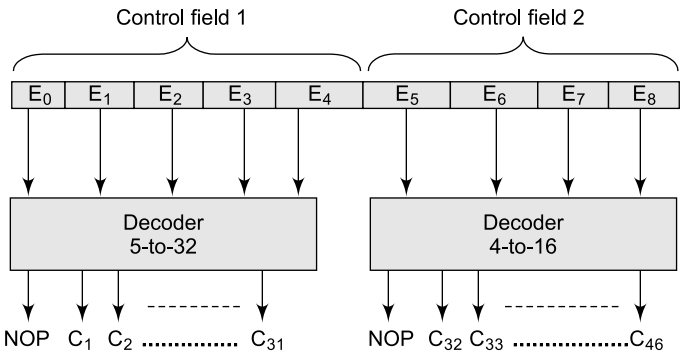
Answer

For reading an instruction from memory and decoding, number of micro-instructions required = 7 and that for execution = 15. Therefore, time required to process an instruction = 7 + 15 = 22 ns.

4. *An encoded microinstruction format is to be used in control unit. Show how a 9-bit micro-operation field can be divided into subfields to specify 46 different control signals.*

Answer

The 9-bit micro-operation can be divided into two subfields to specify 46 control signals by a partial encoding, as shown in figure below.



The control signals are partitioned into disjoint groups of control fields, so two signals of different groups can be enabled in parallel. The control signals are partitioned into two groups as:

Group 1: $C_1\ C_2\ \ldots C_{31.}$
Group 2: $C_{32}\ C_{33}\ \ldots C_{46.}$

With the above grouping, C_{32} can be activated simultaneously with C_1 or C_2 but not C_1 and C_2. Using one 5-to-32 and one 4-to-16 decoders, the actual control signals are generated.

5. *A processor has 28 distinct instructions with 13 instructions having 12 micro-instructions and 15 having 18 micro-instructions.*
   (a) *How many addresses are used in control memory?*
   (b) *If three instructions jump to another set of micro-instructions, each having four micro-instructions, then how many addresses are now used in control memory? Assume that each micro-instruction also stores a branch address.*

Answer

(a) Number of addresses in control memory for 28 instructions = $13 \times 12 + 15 \times 18 = 426$.
(b) Number of addresses in control memory = $13 \times 12 + 15 \times 18 + 3 \times 4 = 438$.

6. *Why do most modern processors follow microprogramming control organizations?*

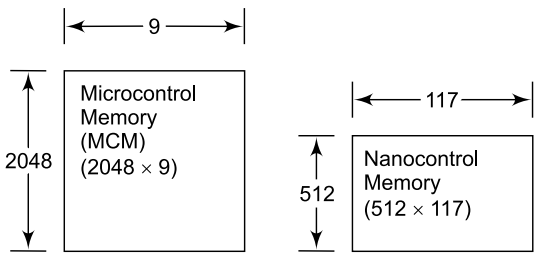
Answer

Microprogramming allows relatively complex instructions to be implemented using small number of hardware circuitry, because control memory stores the necessary control signals directly instead of large number of logic gates as used in hardwired control organization. Moreover, many additions and changes can be made by simply changing the microprogram in the control memory.

7. *A conventional microprogrammed control unit includes 2048 words by 117 bits. Each of 512 microinstructions is unique. Calculate the savings achieved by having a nanoprogramming technique. Calculate the sizes of microcontrol memory and nanocontrol memory.*

Answer

It has 2048 microinstructions, out of which 512 are unique, as shown in figure below. The contents of the microcontrol memory (MCM) are pointers to the nanocontrol memory (NCM). Each microinstruction of the MCM is of size = $\lceil \log_2 512 \rceil = 9$ bits in length.



Control unit's memory structure using nanoprogramming

Now, control unit has: the MCM is 2048×9 bits and the NCM is 512×117 bits, since there are 512 unique microinstructions. In conventional microprogramming by using a single CM, this memory size is 2048×117 bits. Therefore, the use of nanoprogramming saves a total of

$= 2048 * 117 - (2048 * 9 + 512 * 117)$ bits
$= 1,61,280$ bits.

8. *What are the advantages and disadvantages of two-level control structure?*

Answer

Advantages:

The two-level control structure using nanoprogramming technique offers significant savings in memory space when a group of micro-operations occur many times in a microprogram.

Disadvantages:
The main disadvantage of two level control structure using nanoprogramming is that a control unit using a nanocontrol memory (NCM) is slower than one using a conventional control memory, since the nanoprogramming concept consists of two-level memory. Therefore, two memory fetches (one for the microcontrol memory (MCM) and the other for NCM) are required.

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:
   (i) Microprogrammed control unit is
      (a) sequential logic controller
      (b) low-cost control unit
      (c) easily modifiable, because it follows programming technique
      (d) none.
   (ii) A hardwired control unit needs
      (a) sequence controller and decoder, but does not use control memory
      (b) state machine and control memory
      (c) multiplexer and control memory
      (d) encoder and control memory.
   (iii) The control unit which is once designed no further change is possible is
      (a) Hardwired
      (b) Micro-programmed
      (c) Firmware
      (d) Programmed
   (iv) In horizontal microprogramming, microinstruction bits
      (a) need extra encoder
      (b) need extra decoder
      (c) generate smaller output control signals
      (d) does not require extra decoding circuitry to generate control signals.
   (v) In vertical microprogramming, microinstruction bits
      (a) need extra encoder
      (b) need extra decoder
      (c) generate smaller output control signals
      (d) does not require extra decoding circuitry to generate control signals.
   (vi) High degree of parallelism is possible in
      (a) horizontal microprogrammed control organization
      (b) vertical microprogrammed control organization
      (c) mixed control organization
      (d) none.
   (vii) Microprogram counter (MPC) is special register used in the microprogrammed control unit to hold
      (a) address of the next microinstruction to be executed
      (b) address of a branch instruction always

(c) address of the next sequential instruction
(d) none.

(viii) An emulator for an instruction set S is
(a) a hardwired control unit for S
(b) a microprogram for a particular program
(c) a microprogram control unit for S
(d) a set of microprograms that interpret S.

(ix) Relatively new processors use
(a) microprogrammed CU (b) hardwired CU
(c) merger of two (d) none.

(x) The MUX is used in microprogrammed control unit to
(a) select one of microinstructions to be executed
(b) select one of control signals generated simultaneously
(c) select one of the external conditions
(d) none.

(xi) The important advantage of using two-level nanoprogramming is
(a) rapid generation of control signals
(b) reduced required space in control store
(c) high parallelism in control signals
(d) less complex.

(xii) Control memory width can be optimized by
(a) reducing the length of microinstruction
(b) employing efficient encoding technique
(c) encoding maximally compatible signals in the same control field
(d) none.

(xiii) The length of control memory can be reduced by
(a) using unencoded microinstructions
(b) encoding maximally compatible signals in the same control field
(c) encoding microinstructions in different control fields which can be executed concurrently
(d) none.

Group B

2. Describe the functions of control unit (CU).
3. What are different methods for control unit design? Explain.
4. Compare different techniques to design control unit.
5. What are the advantages of hardwired control unit? Describe the design of hardwired control unit of CPU, with diagram.
6. Consider the following algorithm:

```
        Registers: A[8], B[8], C[8];
START:  A ← 0000 1101;
        B ← 0000 0101;
LOOP:   A ← A * B;
        B ← B – 1;
        If B ≠ 0 then go to LOOP;
        C ← A;
HALT:   go to HALT;
```

Design a hardwired control unit to implement this algorithm.

7. What are the advantages of microprogrammed control unit? Describe the design of microprogrammed control unit of CPU, with diagram.
8. Define:
   (a) control memory (b) microinstruction (c) microprogram.
9. What are the different methods for organizing control field of microinstructions? Explain each of them in details.
10. What are the characteristics of horizontal and vertical microprogrammings? Give example for each.
11. Design a microprogrammed control unit for algorithm in Question No. 6.
12. An encoded microinstruction format is to be used in control unit. Show how a 9-bit micro-operation field can be divided into subfields to specify 46 different control signals.
13. What are the advantages and disadvantages of two-level control structure?
14. A conventional microprogrammed control unit includes 2048 words by 117 bits. Each of 512 microinstructions is unique. Calculate the savings achieved by having a nanoprogramming technique. Calculate the sizes of microcontrol memory and nanocontrol memory.
15. Show that it is possible to specify 675 micro-operations using a 10-bit control function field.

CHAPTER

7
Input-Output Organization

7.1 INTRODUCTION

Data transfer between the computer and external device takes place through I/O mechanism. One communicates with a computer system via the I/O devices interfaced to it. The user can enter programs and data using the keyboard on a terminal, executes the programs to obtain results and finally the results may be displayed on monitor of the computer. Therefore, the I/O devices connected to a computer system provide an efficient means of communication between the computer and the outside world. These I/O devices are commonly known as *peripherals* and popular I/O devices used are keyboard, monitor, printer, disk and mouse.

In this chapter, we will consider in detail interconnection system and various ways in which I/O operations are performed.

7.2 I/O INTERFACE AND I/O DRIVER

Every computer supports a variety of peripheral devices. To use a peripheral device, two modules are required:

(a) I/O interface or I/O controller.
(b) I/O driver.

7.2.1 I/O Interface or I/O Controller

I/O interface is a hardware device provides a means for transferring information between central system (i.e. CPU and main memory) and external I/O peripheral device. Peripheral devices connected to a computer need I/O interface circuits for interfacing them with the CPU and/or memory. Each peripheral has its own I/O controller that operates the particular device. The purpose of the I/O interface is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

- Peripherals are mainly electromechanical and electromagnetic devices and their manner of operations is different from the operation of the CPU and main memory, which are electronic devices. Hence, a conversion of signal values may be required.
- Data codes and formats in peripheral devices are different from the code format in the CPU and memory.
- The data transfer rate of the peripheral devices is usually slower than the transfer rate of the CPU and therefore, a synchronization mechanism may be required.
- The various peripheral devices attached to a computer have different modes of operations and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

In order to resolve these differences, computer systems must include special hardware device called *I/O interface* with each peripheral to supervise and synchronize all I/O transfers.

The linkage of the I/O interface to the central computer is via the bus. A typical communication linkage between the central computer (i.e. CPU and main memory) and several peripheral devices is shown in Fig. 7.1. The I/O bus consists of data lines, address lines and control lines. The magnetic disk, printer, monitor and keyboard are used in practically any general-purpose computer. Each peripheral device has associated with it an I/O interface unit. A device's interface circuit constitutes of an address decoder, the data and status registers, and the control circuitry. Each I/O interface unit decodes the address and control signal received from the I/O bus, interprets them for the peripheral and provides signals for the peripheral. It synchronizes the data flow and supervises the transfer between peripheral and processor. For example, the printer controller controls the paper motion, the printing timing and the selection of printing characters. An I/O controller may be incorporated separately or may be physically integrated with the peripheral.

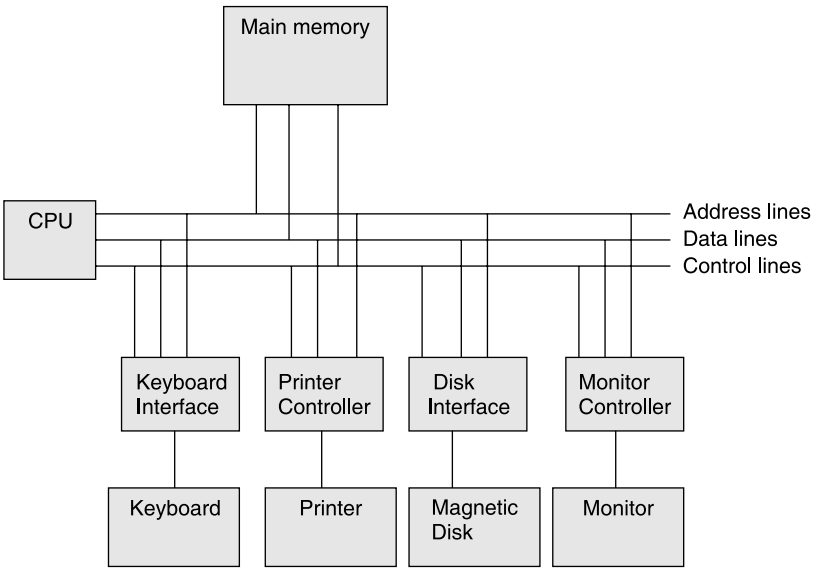


Figure 7.1 *Connection of I/O bus to I/O controllers*

The I/O bus from the central computer is attached to all peripheral controllers. To communicate with a particular I/O device, the processor places a device address on the address lines. The address decoder of I/O interface monitors the address lines. When a particular I/O interface detects its own address, it activates the path between the bus lines and the peripheral device that it controls. All peripherals whose addresses do not correspond to the address in the bus are disabled by their interfaces.

When the address is made available in the address lines, at that time the processor provides an operation code in the control lines. The interface selected responds to the operation code and proceeds to execute it. The operation code is referred to as an I/O command. The meaning of the command depends on the peripheral type that the processor is addressing. There are four types of I/O commands that an I/O interface may receive when it is addressed by a processor:

1. ***Control*** It is used to enable an I/O device and to give directive what to do. For example, a magnetic tape unit may be instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.

2. ***Test*** It is used to test various status conditions associated with an I/O interface and its peripheral. For example, the processor may want to know that the peripheral of interest is powered on and ready for use. It also may want to know if the most recent I/O operation is completed and if any error occurs.

3. ***Read*** This causes the I/O interface to obtain a data-item from the peripheral and places it in an internal buffer (data register). The processor can then obtain the data item by requesting that the I/O interface places it on the data bus.

4. ***Write*** This causes the I/O interface to take a data-item from the data bus and subsequently transfers that data item to the peripheral.

Some well known I/O interface devices are: SCSI (Small Computer System Interface), USB (Universal Serial Bus), IDE (Integrated Drive Electronics), RS-232C, FireWire, Centronics Interface.

There are two types of I/O interfaces available: *Serial interface* and *Parallel interface*. In *serial* interface, there is only one data line and thus data bits are transmitted serially one after other. Examples include: USB, RS-232C, and FireWire. In *parallel* interface, there are multiple data lines in parallel and thus multiple number of bits can be transmitted from the system simultaneously. Examples include: SCSI, Centronics Interface, and IDE.

7.2.2 I/O Driver

I/O driver is a software module that issues different commands to the I/O controller, for executing various I/O operations. Following are certain operations performed by different I/O drivers:

- Reading a file from a disk.
- Printing some lines by the printer.
- Displaying a message on monitor.
- Storing some data on disk.

The I/O driver program for a given peripheral device is developed only after knowing the architecture of the I/O controller device. The I/O driver program and I/O controller device together achieve the I/O operation done on behalf of corresponding peripheral device. An I/O operation can be performed by calling the relevant I/O interface (or I/O controller) and passing relevant signals for

operation. After completing the I/O operation, the I/O driver returns control to the called program and passes return signals about the completion of the operation. Figure 7.2 illustrates communication between the I/O controller and the application program. The collection of I/O driver programs is called BIOS (Basic Input Output System).

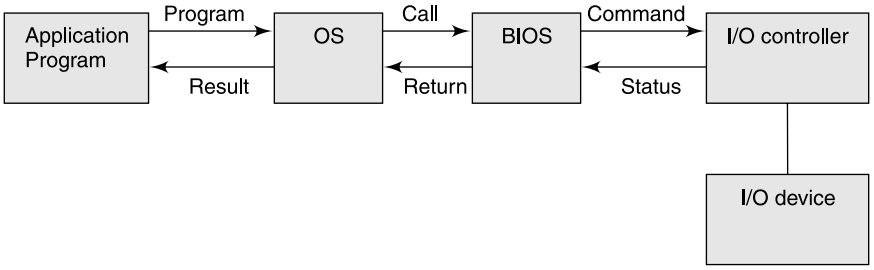


Figure 7.2 *Communication between I/O controller and application program*

The I/O drivers for the basic peripheral devices supported by the general PC are part of the BIOS which is physically stored in ROM part of main memory. The I/O drivers for other peripherals are provided on the floppy diskette or CD. This program is installed in the hard disk and brought into the RAM by bootstrap system program during booting.

7.3 ACCESSING I/O DEVICES

Like the I/O bus, the memory bus contains data, address and read/write control lines. In addition to communicating with I/O, the processor must communicate with the memory unit. There are three ways that processor uses computer buses to communicate with memory and I/O:

1. Use two separate buses (address, data and control), one for memory and the other for I/O.
2. Use one common bus (address and data) for both memory and I/O but have separate control lines for each.
3. Use one common bus (address, data and control) for memory and I/O.

In the first case, the computer has separate sets of data, address, and control buses, one for accessing memory and the other for I/O. This procedure is used in computers that provide a separate *I/O processor (IOP)*, also called *I/O channel* in addition to the CPU. The memory communicates with both the CPU and the IOP through a memory bus. The objective of the IOP is to provide a separate path for the transfer of information between external I/O devices and internal memory.

In the second case, computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines. The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines. I/O-read and I/O-write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is referred to as the isolated I/O or I/O mapped I/O method for assigning addresses in a common bus. This method isolates memory and I/O addresses

so that memory address values are not affected by interface address assignment since each has its own address space, as shown in Fig. 7.3. This method is followed in the IBM PC.

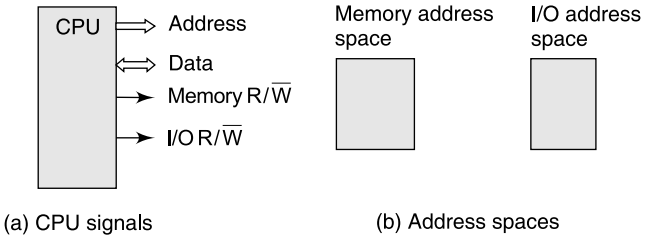


Figure 7.3 *I/O mapped I/O*

In the third case, computers use only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as memory mapped I/O, as depicted in Fig. 7.4. The processor treats an I/O interface register as being part of the memory system. In other words, the processor uses a portion of the memory addresses to represent I/O interface. Computers with memory mapped I/O can use memory type instructions to access I/O data. It allows the computer to use the same instructions for either I/O transfers or for memory transfers. Most of the modern computers use this technique; even though some computers like Intel 8088 support both I/O mapped I/O and memory mapped I/O techniques.

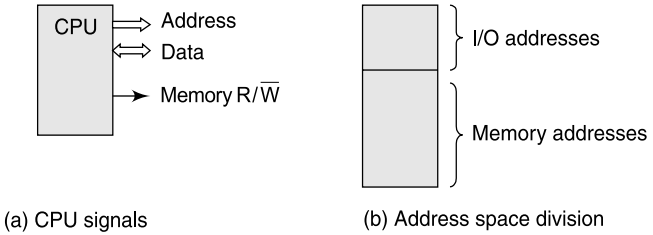
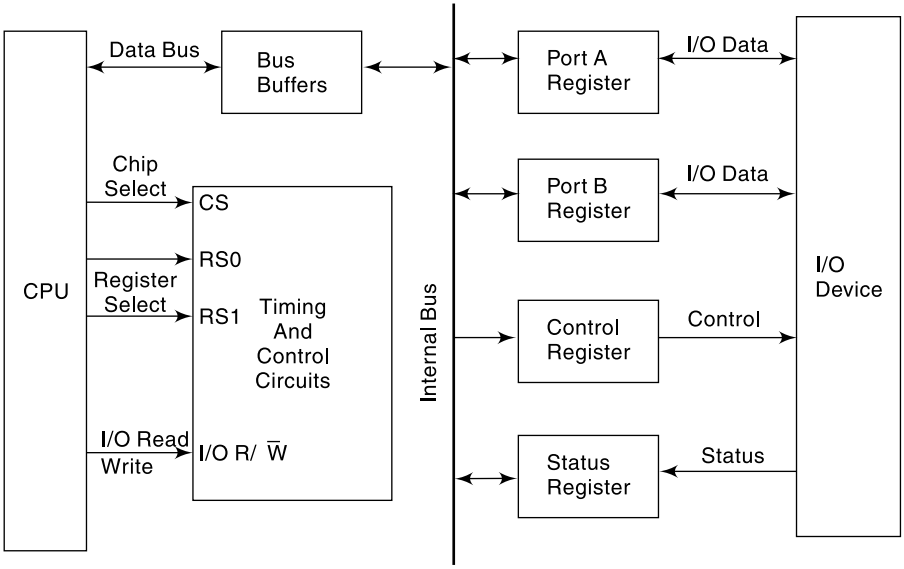


Figure 7.4 *Memory mapped I/O*

I/O Port Figure 7.5 shows a block diagram of an example of an I/O interface unit. It consists of two data registers called *ports,* a control register, a status register, bus buffers and timing and control circuits. Using the data bus, the interface communicates with the CPU. The chip select and register select inputs determine the address assigned to the interface. The I/O R/$\overline{W}$ is a common control line that specifies an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or port B. The interface may operate with an input device or with an output device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data. A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines. A command is passed to the I/O device by sending a word to the appropriate interface register. In a system like this, the command in the I/O bus is not needed because control is sent to the control register, status information is received from the status register and data are transferred to and from port A and port B registers. Thus the transfer of data, control and status information is always through the common data bus. The distinction between data, control or status information is determined from the particular interface register with which the CPU communicates.

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in various operating modes. For example, port A may be defined as an input port and port B as an output port. A magnetic tape unit may be instructed to start the tape moving in the forward direction or to rewind the tape. The bits in the status register are used for status conditions and for recording errors that may occur during



CS	RS1	RS0	Register Selected
0	X	X	None (Data bus is in high-impedance state)
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Figure 7.5 *Example of an I/O interface unit*

the data transfer. For example, a status bit may indicate that port B has received a new data item from the CPU that is to be transferred to the I/O device. Another bit in the status register may indicate that a parity error has occurred during the transfer.

The interface registers communicate with the CPU through bidirectional data bus. The address bus selects the interface unit through the chip select (CS) line and the two register select input lines (RS0 and RS1). Usually an address decoder circuit must be provided externally to detect the address assigned to the interface registers. This decoder circuit enables the chip select input when the interface is selected by the address bus. The two register select inputs (RS0 and RS1) are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as specified in the table associated with the diagram. The content of the selected register is transfer into the CPU through the data bus when the I/O R/$\overline{W}$ signal is at logic '1' (i.e. I/O read). The CPU transfers binary information into the selected register via the data bus when the I/O R/$\overline{W}$ input is at logic '0' (i.e. I/O write).

7.4 SYNCHRONOUS AND ASYNCHRONOUS DATA TRANSFERS

In order to execute a task in a computer, transfer of information among different devices is necessary.

When two units have to communicate with each other for data transfer, usually one of them is the master and the other one is slave. Sometimes one word of data is transferred between two units in one clock period or some times in more than one clock period. There are two types of data transfer depending on the mechanism of timing the data: Synchronous and Asynchronous.

7.4.1 Synchronous Transfer

In this mode of data transfer, the sending and receiving units are enabled with same clock signal. The synchronous transfer is possible between two units when each of them knows the behavior of the other. The master performs a sequence of actions for data transfer in a predetermined order; each action is synchronized with the common clock. The master is designed to supply the data at a time when the slave is definitely ready for it. Usually, the master will introduce sufficient delay to take into account the slow response of the slave, without any request from the slave. The master does not expect any acknowledgement signal from the slave, when a data is sent by the master to the slave. Similarly, when a data from slave is read by the master, neither the slave informs that a data has been placed on the data bus nor the master acknowledges that a data has been read. Both master and slave perform their own task of transferring data at designated clock period. Since both devices know the behavior (response time) of each other, no difficulty arises. Prior to transferring data, the master must logically select the slave either by sending slave's address or sending "device select" signal to the slave. But, there is no acknowledgement signal from the slave to master if device is selected.

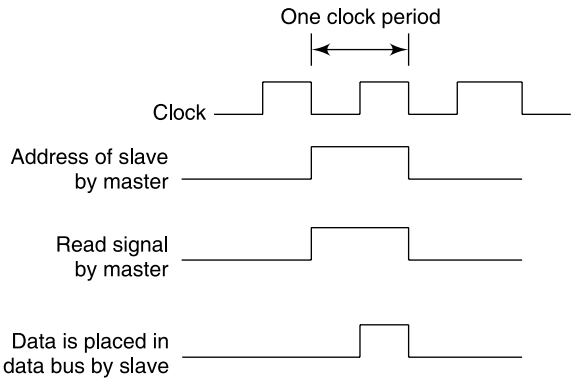


Figure 7.6 *Timing diagram for synchronous read operation*

As for example, the Fig. 7.6 shows timing diagram for synchronous read operation. The master first places slave's address in the address bus and read signal in the control line at the falling edge of a clock. The slave places data in the data bus at the raising edge of the clock. The entire read operation is over in one clock period.

Advantages of Synchronous Transfer

1. The design procedure is easy. The master does not wait for any acknowledge signal from the slave though the master waits for a time equal to slave's response time.
2. The slave does not generate acknowledge signal, though it obeys the timing rules as per the protocol set by the master or system designer.

Disadvantages of Synchronous Transfer

1. If a slow speed unit is connected to a common bus, it can degrade overall rate of transfer in the system.
2. If the slave operates at a slow speed, the master will be idle for some time during data transfer and vice versa.

7.4.2 Asynchronous Transfer

There is no common clock between the master and slave in asynchronous transfer. Each has its own private clock for internal operations. This approach is widely used in most computers. Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted. One simple way is to use a strobe signal supplied by one of the units to indicate the other unit when the transfer has to occur.

Strobe Control Technique A single control line is used by the strobe control method of asynchronous data transfer to time each transfer. The strobe may be activated by either the source or the destination unit. A source-initiated transfer is depicted in Fig. 7.7. The source takes care of proper timing delay between the actual data signals and the strobe signal. The source places the data first, and after some delay, generates the strobe to inform about the data on the data bus. Before removing the data, source removes the strobe and after some delay it removes the data. By these two leading and trailing end delays, the system ensures the reliable data transfer.

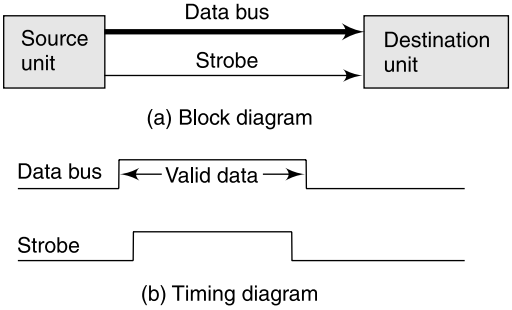


Figure 7.7 *Source-initiated strobe for data transfer*

Similarly, the destination can initiate data transfer by sending strobe signal to the source unit as shown in Fig. 7.8. In response, the source unit places data on the data bus. After receiving data, the destination unit removes the strobe signal. Only after sensing the removal of strobe signal, the source removes the data from the data bus.

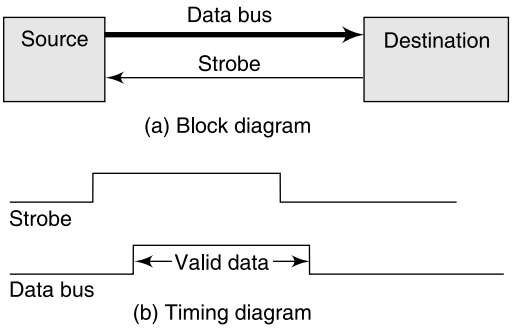


Figure 7.8 *Destination initiated strobe for data transfer*

The *disadvantage* of the strobe method is that the source unit that initiates the transfer cannot know whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer cannot know whether the source unit has actually placed the data on the bus.

Handshaking Technique To overcome this problem of strobe technique, another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as handshaking mode of transfer.

Figure 7.9 shows the data transfer method when initiated by the source. The two handshaking lines are “data valid”, which is generated by the source unit, and “data accepted” generated by the destination unit. The source first places data and after some delay issues data valid signal. On sensing data valid signal, the destination receives data and then issues acknowledgement signal data accepted to indicate

the acceptance of data. On sensing data accepted signal, the source removes data and data valid signal. On sensing removal of data valid signal, the destination removes the data accepted signal.

Figure 7.10 illustrates destination initiated handshaking technique. The destination first sends the *data request* signal. On sensing this signal, the source places data and also issues the *data valid* signal. On sensing data valid signal, the destination acquires data and then removes the data request signal. On sensing this, the source removes both the data and data valid signal.

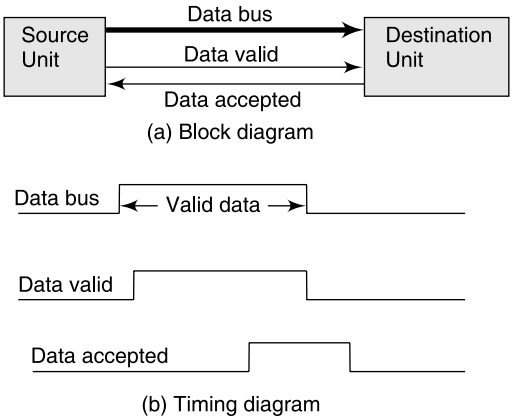


Figure 7.9 *Source-initiated transfer using handshaking*

The advantage of handshaking scheme is that it provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.

The disadvantages of handshaking scheme are:

1. A slow speed destination unit can hold up the bus whenever it gets a chance to communicate.
2. If one of the two communicating devices is faulty, the initiated data transfer cannot be completed.

Examples of asynchronous transfer

1. The centronics interface follows handshaking scheme.
2. Most microprocessors such as Motorola 88010 and Intel 80286 follow this bus transfer mechanism.

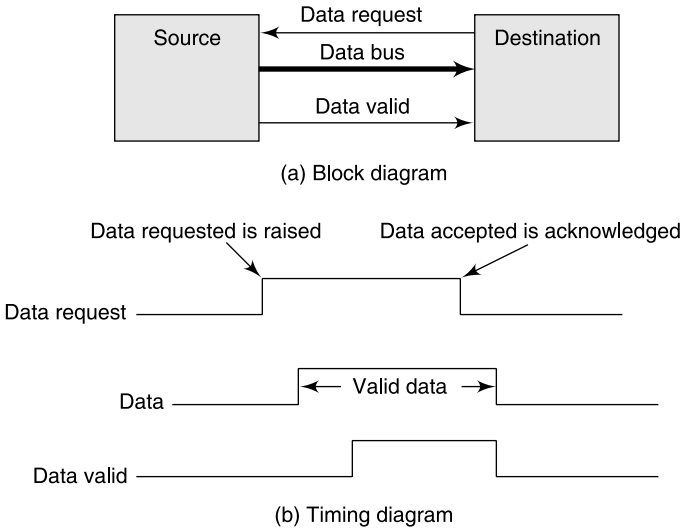


Figure 7.10 *Destination initiated handshaking technique*

7.5 MODES OF DATA TRANSFER

Information transferred from the central computer (i.e. CPU and main memory) into a peripheral device originates in the memory unit. Information received from a peripheral device is usually stored in memory for later processing. The CPU only executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Three possible modes are:

1. Programmed I/O.
2. Interrupt-initiated I/O.
3. Direct memory access (DMA).

The CPU executes a program to communicate with an I/O device via I/O interface registers for programmed I/O. This is a software method.

An external I/O device requests the processor to transfer data by activating a signal on the computer's interrupt line during interrupt-initiated I/O. In response, the computer executes a program called the *interrupt-service routine* (ISR) to carry out the function desired by the external I/O device. This is also a software method.

Data transfer between the computer's main memory and an external I/O device occurs without CPU involvement in direct memory access (DMA). It is a hardware method.

7.5.1 Programmed I/O

This is the software method where CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.In other words, the CPU polls the devices for next data transfer. This is why the programmed I/O is sometimes called polled I/O.Through the mid-1990s, programmed I/O was the only way that most systems ever accessed IDE/ATA hard disks.

Example 7.1 Example of Programmed I/O: Transferring data from I/O device to memory.

I/O device does not have direct access to memory in the programmed I/O method. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.

Figure 7.11 shows an example of data transfer from an I/O device through its interface into memory via the CPU. The handshaking procedure is followed here. The device transfers bytes of data one at a time, as they are available. The device places a byte of data, when available, in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables its data accepted line. A flag bit is then set in its status register by the interface. The device now disables the data valid line, but it will not transfer another byte until the data accepted line is disabled and flag bit is reset by the interface.

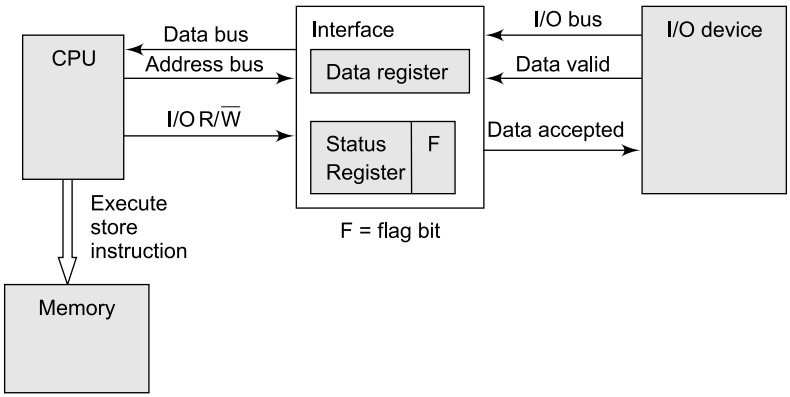


Figure 7.11 *Data transfer from I/O device to memory through CPU*

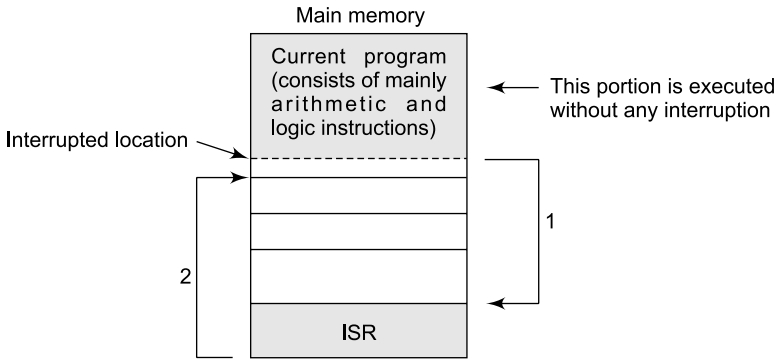
An I/O routine or a program is written for the computer to check the flag bit in the status register to determine if a byte is placed in the data register by the I/O device. By reading the status register into a CPU register and checking the value of the flag bit, this can be done. When the flag is set to 1, the CPU reads the data from data register and then transfers to the memory by executing store instruction. The flag bit is then reset to 0 by either the CPU or the interface, depending on the design of interface circuits. When the flag bit is reset, the interface disables the data accepted line and the device can then transfer the next data byte. Thus the following four steps to be executed by the CPU to transfer each byte:

1. Read the status register of interface unit.
2. Check the flag bit of the status register and go to step (3) if it is set; otherwise loop back to step (1).
3. Read the data register of interface unit for data.
4. Send the data to the memory by executing store instruction.

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. Generally the CPU is 5-7 times faster than an I/O device. Thus, the difference in data transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

7.5.2 Interrupt-initiated I/O

In the programmed I/O method, the program constantly monitors the device status. Thus, the CPU stays in the program until the I/O device indicates that it is ready for data transfer. This is time-consuming process since it keeps the CPU busy needlessly. It can be avoided by letting the device controller continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to an *interrupt-service-routine (ISR) or I/O routine or interrupt handler* to process the I/O transfer, and then returns to the task it was originally performing. Thus, in the interrupt-initiated mode, the ISR software (i.e. CPU) performs data transfer but is not involved in checking whether the device is ready for data transfer or not. Therefore, the execution



1. CPU branches to ISR on receiving interrupt during say, ith instruction execution. Before branch, ith instruction is executed.
2. On completion of ISR, CPU returns to original program and resumes execution of the program starting from (i+1)th instruction.

Figure 7.12 *Interrupt process*

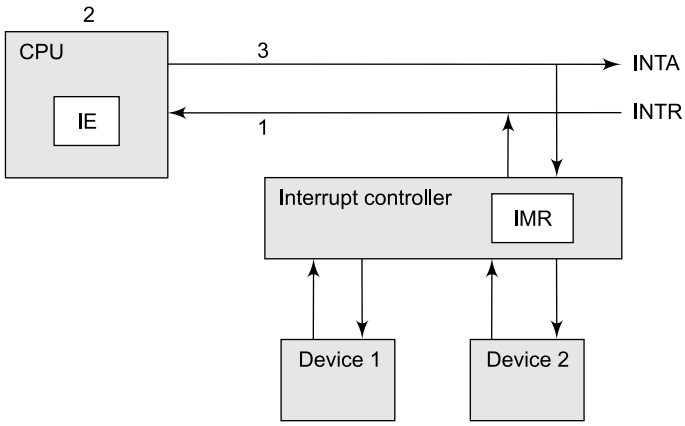
time of CPU can be optimized by employing it to execute normal program, when no data transfer is required. Figure 7.12 illustrates the interrupt process.

The CPU responds to the interrupt signal by storing the return address from the program counter (PC) register into a memory stack or into a processor register and then control branches to an ISR program that processes the required I/O transfer. The way that the CPU chooses the branch address of the ISR varies from one unit to another. In general, there are two methods for accomplishing this. One is called vectored interrupt and the other is non-vectored. In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory.

In interrupt-initiated I/O, the device controller should have some additional intelligence for checking device status and raising an interrupt whenever data transfer is required. This results in extra hardware circuitry in the device controller.

Interrupt Hardware An interrupt handling hardware implements the interrupt. To implement interrupts, the CPU uses a signal known as an *interrupt request (INTR)* signal to the interrupt handler or controller hardware, which is connected to each I/O device that can issue an interrupt to it. Here, interrupt controller makes liaison with the CPU on behalf of I/O devices. Typically, interrupt controller is also assigned an *interrupt acknowledge (INTA)* line that the CPU uses to signal the controller that it has received and begun to process the interrupt request by employing an ISR. Figure 7.13 shows the hardware lines for implementing interrupts.

The interrupt controller uses a register called *interrupt-request mask register (IMR)* to detect any interrupt from the I/O devices. Consider there is n number of I/O devices in the system. Therefore IMR is n-bit register, where each bit indicates the status of one I/O device. Let, IMR's content is denoted as E_0 E_1 E_2 … E_{n-1}. When $E_0 = 1$ then device 0 interrupt is recognized; When $E_1 = 1$ then device 1 interrupt is recognized and so on. The processor uses a flag bit known as *interrupt enable (IE)* in its status register (SR) to process the interrupt. When this flag bit is '1', the CPU responds to the presence of interrupt; otherwise not.



1. Interrupt from interrupt controller when data transfer is needed.
2. Using IE flip-flop, CPU detects interrupt.
3. CPU branches to a respective device's ISR after enabling INTA

Figure 7.13 *Hardware interrupt*

Enabling and Disabling Interrupts Sometimes a program does not want any interruption; it informs the CPU not to encourage any interrupt. The CPU ignores interrupt and hence any interrupts which arrived in the mean time remain pending, until the program issues another directive to the CPU asking it to allow interrupt. Therefore, there has to be some feature to enable or disable an interrupt. Disabling an interrupt is called *masking*. There can be two types of interrupts in an I/O organization.

- Non-maskable interrupts: The user program cannot disable it by any instruction. Some common examples are: hardware error and power fail interrupt.
- Maskable interrupts: The user program can disable all or a few device interrupts by an instruction.

A flag bit known as *interrupt enable (IE)* is used in processor status register (SR) to process the interrupt. When this flag bit is '1', the CPU responds to the interrupt. When this flag bit is '0', the CPU ignores the interrupt. The program must issue *enable interrupt (EI)* instruction to set the IE flag. The CPU sets the IE flag, when executing this instruction. The program must issue *disable interrupt (DI)* instruction to reset the IE flag. The CPU resets the IE flag, when executing this instruction. Thus, in respect of interrupt servicing the CPU's behavior is controlled by the program that is being executed currently. There are two following special situations when the CPU resets the IE flag on its own:

1. During non-maskable interrupts handling.
2. During interrupt servicing; the CPU resets the IE flag immediately after saving the return address into memory or into a special register and before branching to ISR. Thus, when CPU starts execution of ISR, interrupts are disabled. Therefore, it is up to the ISR to allow interrupts (by an EI instruction) if it wishes.

Interrupt Nesting During the execution of one ISR, if it allows another interrupt, then this is known as interrupt nesting. Suppose the CPU is initially executing program 'A' when first interrupt occurs. The CPU after storing return address of instruction in program 'A', starts executing ISR1. Now say in the mean time, second interrupt occurs. The CPU again after storing return address of

instruction in ISR1, starts executing ISR2. When it is executing ISR2, third interrupt occurs. The CPU again performs storing of return address for ISR2 and then starts executing ISR3. After completing ISR3, the CPU resumes the execution of ISR2 for remaining portion. Similarly, after completing ISR2, the CPU resumes the execution of ISR1 for remaining portion. After completing ISR1, the CPU returns to the program 'A' and continues from the location it branched earlier.

Priority Interrupt In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt controller is to identify the source of the interrupt. There is also the possibility that several sources may request interrupt service simultaneously. In this case the controller must also decide which to service first. A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. Devices with high-speed transfers such as magnetic disks are usually given high priority and slow devices such as keyboards receive low priority. When two devices interrupt the CPU at the same time, the CPU services the device, with the higher priority first.

The interrupt requests from various sources are connected as input to the interrupt controller. As soon as the interrupt controller senses (using IMR) the presence of any one or more interrupt requests, it immediately issues an interrupt signal through INTR line to the CPU. The interrupt controller assigns a fixed priority for the various interrupt requestor devices. For example, the IRQ0 is assigned the highest priority among the eight different interrupt requestors. Assigning decreasing order of priority from IRQ0 to IRQ7, the IRQ7 is the lowest priority. It (IRQ7) is serviced only when no other interrupt request is present.

7.5.3 Direct Memory Access (DMA)

To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA).*

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure 7.14 shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The

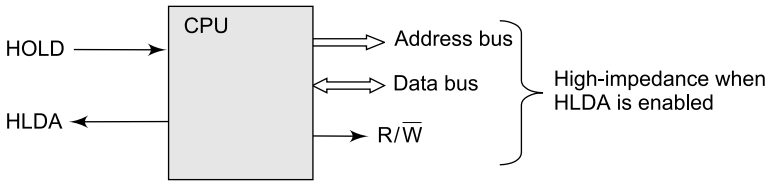


Figure 7.14 *CPU bus signals for DMA transfer*

CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.

DMA Controller To communicate with the CPU and I/O device, the DMA controller needs the usual circuits of an interface. In addition to that, it needs an address register, a word count register, a status register and a set of address lines. Three registers are selected by the controller's register select (RS) line. The address register and address lines are used for direct communication with the memory. The address register is used to store the starting address of the data block to be transferred. The word count register contains the number of words that must be transferred. This register is decremented by one after each word transfer and internally tested for zero after each transfer. Between the device and memory under control of the DMA, the data transfer can be done directly. The status register contains information such as completion of DMA transfer. All registers in the DMA controller appear to the CPU as I/O interface registers. Thus, the CPU can read from or write into the DMA registers under program control via the data bus.

When executing the program for I/O transfer, the CPU first initializes the DMA controller. After that, the DMA controller starts and continues to transfer data between memory and peripheral unit until an entire block is transferred. The DMA controller is initialized by the CPU by sending the following information through the data bus:

1. The starting address of the memory blocks where data are available for read or where data are to be stored for write.
2. The number of words in the memory block (word count) to be read or written.
3. Read or write control to specify the mode of transfer.
4. A control to start the DMA transfer.

DMA Transfer In DMA transfer, I/O devices can directly access the main memory without intervention by the processor. Figure 7.15 shows a typical DMA system. The sequences of events involved in a DMA transfer between an I/O device and the main memory are discussed next.

A DMA request signal from an I/O device starts the DMA sequence. DMA controller activates the HOLD line. It then waits for the HLDA signal from the CPU. On receipt of HLDA, the controller sends a DMA ACK (acknowledgement) signal to the I/O device. The DMA controller takes the control of the memory buses from the CPU. Before releasing the control of the buses to the controller, the CPU initializes the address register for starting memory address of the block of data, word-count register for number of words to be transferred and the operation type (read or write). The I/O device can then communicate with memory through the data bus for direct data transfer. For each word transferred, the DMA controller increments its address-register and decrements its word count register. After each word transfer, the controller checks the DMA request line. If this line is high, next

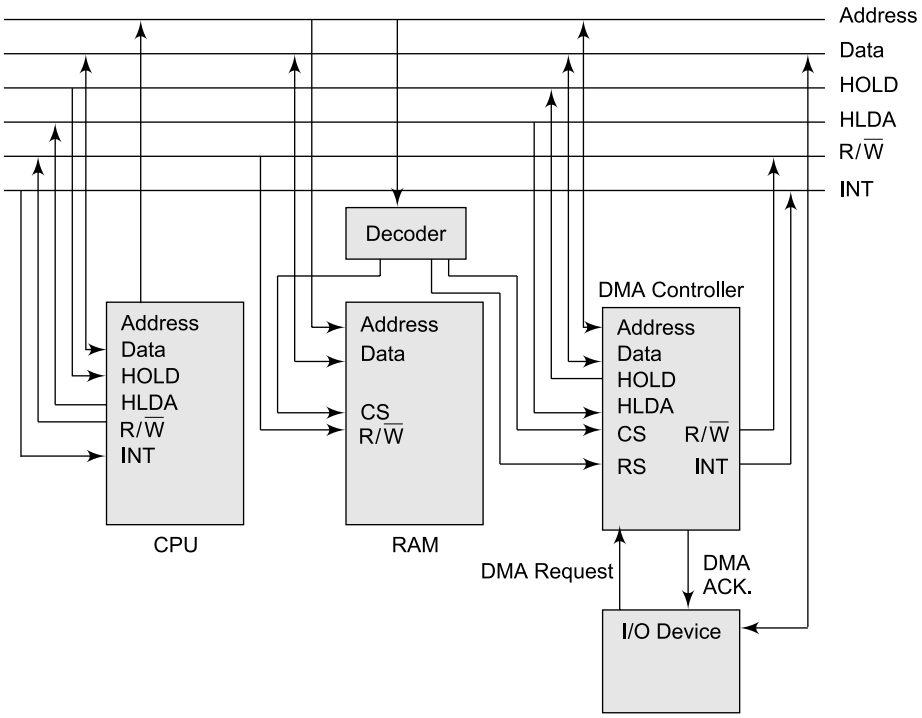


Figure 7.15 *Typical DMA system*

word of the block transfer is initiated and the process continues until word count register reaches zero (i.e., the entire block is transferred). If the word count register reaches zero, the DMA controller stops any further transfer and removes its HOLD signal. It also informs the CPU of the termination by means of an interrupt through INT line. The CPU then gains the control of the memory buses and resumes the operations on the program which initiated the I/O operations.

Advantages of DMA It is a hardware method, whereas programmed I/O and interrupt I/O are software methods of data transfer. DMA mode has following advantages:

1. High speed data transfer is possible, since CPU is not involved during actual transfer, which occurs between I/O device and the main memory.
2. Parallel processing can be achieved between CPU processing and DMA controller's I/O operation.

DMA Transfer Modes DMA transfers can be of two types: *cycle stealing* and *block (burst) transfer*.

Memory accesses by the CPU and the DMA controllers are interlocking. Requests by DMA devices for using memory buses are always given higher priority than processor requests. Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface or a graphics display device. Since the CPU originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the CPU. Hence, this interlocking technique is usually called *cycle stealing*.

When DMA controller is the master of the memory buses, a block of memory words is transferred in continuous without interruption. This mode of DMA transfer is known as *block (burst) transfer*. This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.

7.6 BUS ARBITRATION

A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those. Only one processor or controller can be bus master. The bus master is the controller that has access to a bus at an instance. To resolve these conflicts, bus arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers. *Bus arbitration* refers to a process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus. The *bus arbiter* decides who would become current bus master. There are two approaches to bus arbitration:

1. Centralized bus arbitration: A single bus arbiter performs the required arbitration.
2. Distributed bus arbitration: All devices participate in the selection of the next bus master.

7.6.1 Methods of Bus Arbitration

There are three bus arbitration methods:

1. Daisy Chaining Method.
2. Polling or Rotating Priority Method.
3. Fixed Priority or Independent Request Method.

Daisy Chaining Method The daisy chaining method is a centralized bus arbitration method. During any bus cycle, the bus master may be any device—the processor or any DMA controller unit, connected to the bus. Figure 7.16 illustrates the daisy chaining method.

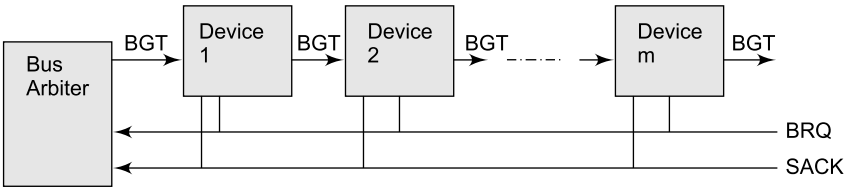


Figure 7.16 *Daisy chained bus arbitration*

All devices are effectively assigned static priorities according to their locations along a bus grant control line (BGT). The device closest to the central bus arbiter is assigned the highest priority. Requests for bus access are made on a common request line, BRQ. Similarly, the common acknowledge signal line (SACK) is used to indicate the use of bus. When no device is using the bus, the SACK is inactive. The central bus arbiter propagates a bus grant signal (BGT) if the BRQ line is high

and acknowledge signal (SACK) indicates that the bus is idle. The first device, which has issued a bus request, receives the BGT signal and stops the latter's propagation. This sets the bus-busy flag in the bus arbiter by activating SACK and the device assumes bus control. On completion, it resets the bus-busy flag in the arbiter and a new BGT signal is generated if other requests are outstanding (i.e., BRQ is still active). The first device simply passes the BGT signal to the next device in the line.

The main advantage of the daisy chaining method is its simplicity. Another advantage is scalability. The user can add more devices anywhere along the chain, up to a certain maximum value.

Polling or Rotating Priority Method In this method, the devices are assigned unique priorities and compete to access the bus, but the priorities are dynamically changed to give every device an opportunity to access the bus. This dynamic priority algorithm generalizes the daisy chain implementation of static priorities discussed above. Recall that in the daisy chain scheme all devices are given static and unique priorities according to their positions on a bus-grant line (BGT) emanating from a central bus arbiter. However, in the polling scheme, no central bus arbiter exists, and the bus-grant line (BGT) is connected from the last device back to the first in a closed loop (Fig. 7.17). Whichever device is granted access to the bus serves as bus arbiter for the following arbitration (an arbitrary device is selected to have initial access to the bus). Each device's priority for a given arbitration is determined by that device's distance along the bus-grant line from the device currently serving as bus arbiter; the latter device has the lowest priority. Hence, the priorities change dynamically with each bus cycle.

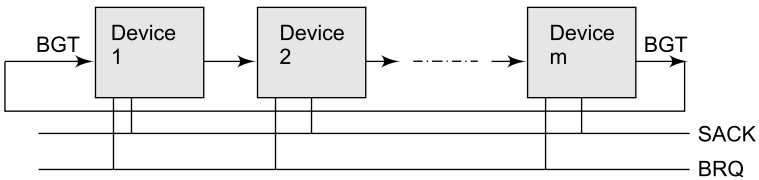


Figure 7.17 *Rotating priority method*

The main advantage of this method is that it does not favor any particular device or processor. The method is also quite simple.

Fixed Priority or Independent Request Method In bus independent request method, the bus control passes from one device to another only through the centralized bus arbiter. Figure 7.18 shows the independent request method. Each device has a dedicated BRQ output line and BGT input line. If there are m devices, the bus arbiter has m BRQ inputs and m BGT outputs. The arbiter follows a priority order with different priority level to each device. At a given time, the arbiter issues bus grant (BGT) to the highest priority device among the devices who have issued bus requests. This scheme needs more hardware but generates fast response.

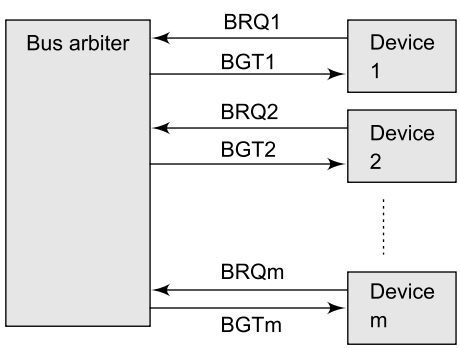


Figure 7.18 *Fixed priority bus arbitration method*

7.7 INPUT-OUTPUT PROCESSOR (IOP)

The DMA mode of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices which are much slower compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rise to the development of special purpose processor called IO processor (IO channel).

The IOP is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in a typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O-related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The block diagram of an IOP is shown in Fig. 7.19. The main memory unit takes the pivotal role. It communicates with processor by means of DMA.

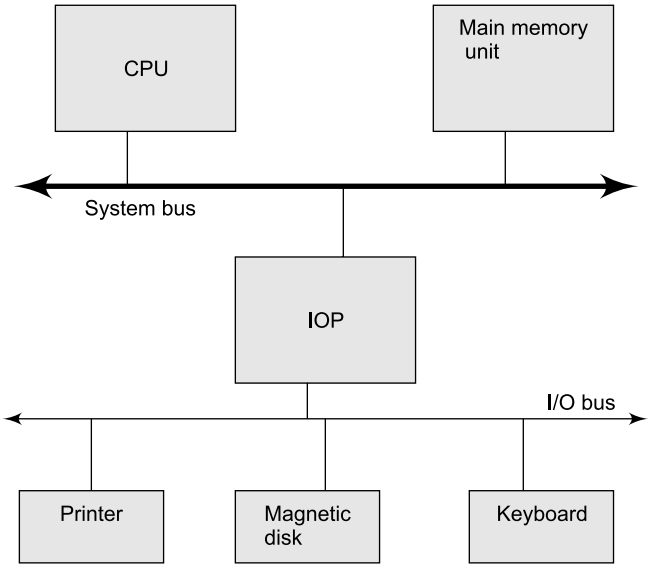


Figure 7.19 *Block diagram of a computer with IOP*

The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources. For example, it may be necessary to take four bytes from an input device and pack them into one 32-bit word before the transfer to memory. Data are gathered in the IOP at the device rate and bit capacity while the CPU is executing its own program. After the input data are assembled into a memory word, they are transferred from IOP directly into memory by "stealing" one memory cycle from the CPU. Similarly, an output word transferred from memory to the IOP is directed from the IOP to the output device at the device rate and bit capacity.

The communication between the IOP and the devices attached to it is similar to the program control method of transfer. Communication with the memory is similar to the direct memory access method. The way by which the CPU and IOP communicate depends on the level of sophistication included in the system. In very-large-scale computers, each processor is independent of all others and any one processor can initiate an operation. In most computer systems, the CPU is the master while the IOP is a slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP. CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities. The IOP, in turn, typically asks for CPU attention by means of an interrupt. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program. When an I/O operation is desired, the CPU informs the IOP where to find the I/O program and then leaves the transfer details to the IOP.

Instructions that are read from memory by an IOP are sometimes called *commands,* to distinguish them from instructions that are read by the CPU. Otherwise, an instruction and a command have similar functions. Commands are prepared by experienced programmers and are stored in memory. The command words constitute the program for the IOP. The CPU informs the IOP where to find the commands in memory when it is time to execute the I/O program.

CPU-IOP Communication The communication between CPU and IOP may take different formats, depending on the particular computer considered. In most cases the memory unit acts as a message center where each processor leaves information for the other.

The sequence of operations may be carried out as shown in the flowchart of Figure 7.20. The CPU sends an instruction to test the IOP path.

The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer. The CPU refers to the status word in memory to decide what to do next. If all are in order, the CPU sends the instruction to start I/O transfer. The memory address received with this instruction tells the IOP where to find its program.

The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer. When the IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer. From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program. The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory. It is not possible to saturate the memory by I/O devices in most systems, as the speed of most devices is much slower than the CPU. However, some very fast units, such as magnetic disks, can use an appreciable number of the available memory cycles. In that case, the speed of the CPU may deteriorate because it will often have to wait for the IOP to conduct memory transfers.

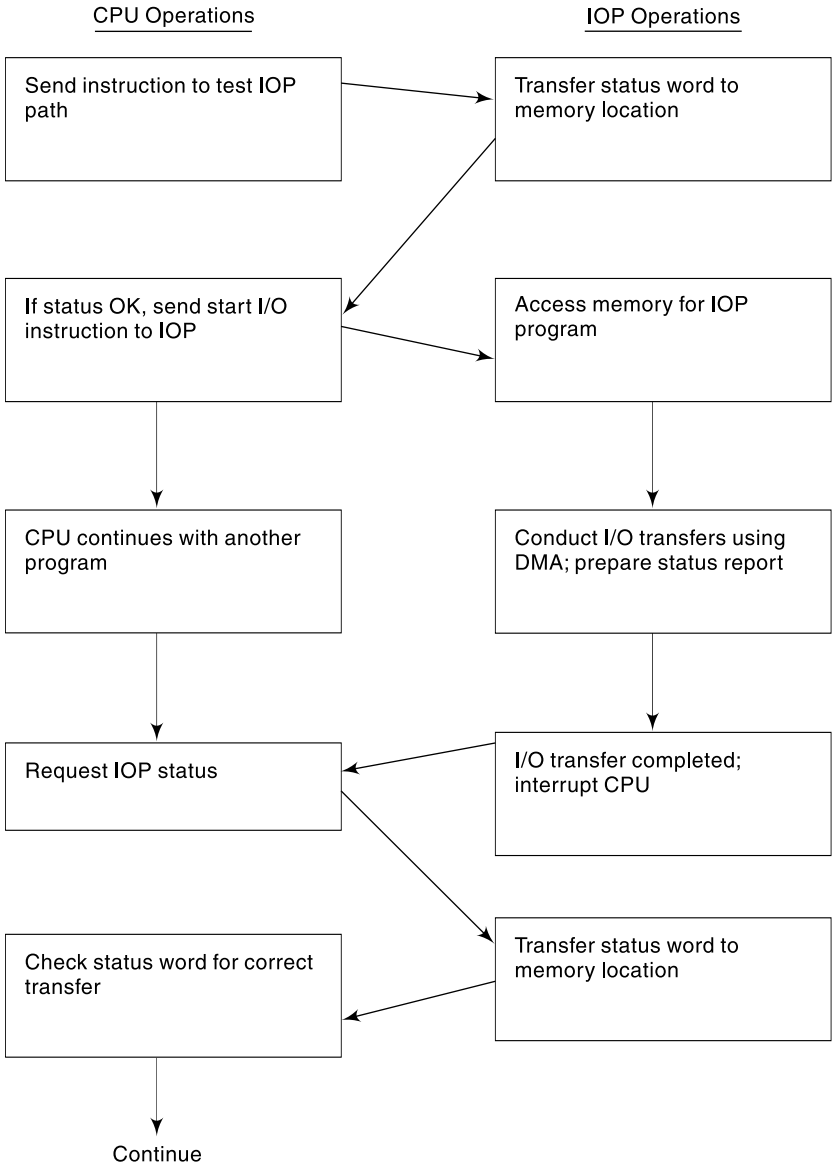


Figure 7.20 *CPU- IOP communication*

7.8 DATA TRANSFER MECHANISM

There are basically two methods of data transfer:

- Parallel
- Serial

7.8.1 Parallel Communication

In parallel mode, all bits of data (usually a byte) are transferred parallel over the communication lines known as buses. Thus, all the bits of data are transferred simultaneously within the timeframe allotted for the transmission. A total of 9-bit (8 bits of a byte data and a parity bit) data bus is commonly used between the sender and receiver. Accordingly, the interface circuit must be of parallel type so that all the bits of data are transferred simultaneously over the data bus (Figure 7.21).

In case of a parallel interface, the connection between the device and the computer uses a multi-pin connector and a cable with as many wires, typically arranged in a flat configuration. The circuits at either end are relatively simple, as there is no need for conversion between parallel and serial formats. This arrangement is suitable for devices that are physically close to the computer. A parallel data transfer scheme has traditionally been used for higher-speed peripherals such as magnetic tape and disk.

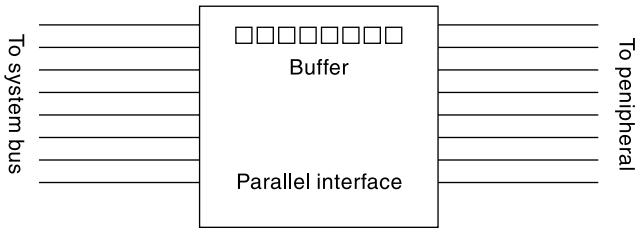


Figure 7.21 *Parallel mode of transfer*

As mentioned earlier, parallel data transfer scheme has the inherent advantage for high data transfer rate. However, it requires large bus width and thus the associated hardware cost is high. The serial format is much more convenient and cost-effective where longer cables are needed.

7.8.2 Standard Parallel Interfaces

IEEE 1284 The demand for higher transfer speed and bi-directional transfer between the PC and an external peripheral introduces the IEEE 1284 standard in 1997. The 1284 standard defines 5 modes of data transfer. Each mode provides a method of transferring data in either the forward direction (PC to peripheral), reverse direction (peripheral to PC) or bi-directional data transfer (half duplex). The defined modes are:

1. *Compatibility Mode* (Centronics or standard mode): Forward direction only.
2. *Nibble Mode* (Hewlett Packard Bi-tronics): 4 bits at a time using status lines for data. Reverse direction only.
3. *Byte Mode*: 8 bits at a time using data lines, sometimes referred to as a bi-directional port. Reverse direction only.
4. *EPP (Enhanced Parallel Port):* Used primarily by non-printer peripherals, CD ROM, tape, hard drive, network adapters, etc. It is bi-directional.
5. *ECP (Extended Capability Port):* Used primarily by new generation of printers and scanners. It is bi-directional.

All parallel ports can implement a bi-directional link by using the compatible and nibble modes for data transfer. Byte mode can be utilized by about 25% of the installed base of parallel ports. All three of these modes utilize software only to transfer the data. The driver has to write the data, check the handshake lines (i.e. BUSY), assert the appropriate control signals (i.e. STROBE) and then go on to the next byte. This is very software intensive and limits the effective data transfer rate to 50 to 100 Kbytes per second.

In addition to the previous 3 modes, EPP and ECP are being implemented on the latest I/O controllers by most of the Super I/O chip manufacturers. These modes use hardware to assist in the data transfer. For example, in EPP mode, a byte of data can be transferred to the peripheral by a simple OUT instruction. The I/O controller handles all the handshaking and data transfer to the peripheral.

SCSI (Small Computer System Interface) SCSI (pronounced as "Scuzzy") is an 8- or 16-bit parallel interface that allows up to seven devices to connect to a PC along a single cable, with each device having a unique address. Many computers use SCSI for interfacing to internal or external hard drives, tape back-ups, and CD-ROMs. SCSI interfaces are fast (synchronous version has a peak data rate of 4 Mbytes and asynchronous version has that of 1.5 Mbytes) and the cable can be as long as 19-feet (6 meters). The SCSI uses a handshaking protocol. A device connected to a SCSI is intelligent and expensive. But the parallel-port interface is simpler, cheaper and much more common. There are different SCSI standards: SCSI (or SCSI 1), SCSI 2 and SCSI 3.

IEEE 488 The IEEE-488 interface began as Hewlett Packard's GPIB (general-purpose interface bus). It is a parallel interface that enables up to 15 devices to communicate at speeds of up to 1 Megabit per second. This interface has long been popular for interfacing to lab instruments. Expansion cards with IEEE-488 interfaces are available.

7.8.3 Serial Communication

In serial data transfer, each data bit is sent sequentially over a single data bus line (Figure 7.22). In order to implement serial data transmission, the sender and receiver must divide the timeframe allotted for the transmission of a data (byte) into subintervals during which each bit is sent and received. For serial transmission, the interface, known as serial interface, uses only one line to transmit data bit by bit. Serial communication is usually selected for data transfer over long distance involving devices having low data-transfer rates. Typically through a serial communication link, the

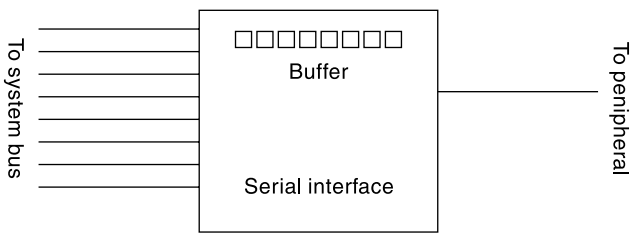


Figure 7.22 *Serial mode transfer*

devices such as the keyboard and mouse are connected directly to the computer. Other devices such as printers and scanners may be connected to a computer either directly or via a communication network so that they may be shared among several users.

Serial communication implies that data are sent one bit at a time. This requires that both the transmitting and the receiving devices use the same timing information for interpretation of individual bits. When the communicating devices are physically close to each other and multiple signal paths are available, a clock signal can be transmitted along with the data. However, this is not feasible over longer links, where only one signal path is available. More importantly, even if a second path is provided, the delays encountered by the data and the clock signals could be different. For these reasons, timing information and data are encoded on one transmission channel. A variety of encoding schemes have been developed that enable the receiver to decode the received signal and recover the timing and transmitted data correctly.

Modulation and Demodulation Computer systems often use standard telephone lines for transmission of digital data over a long distance. The bandwidth of such lines being very low (of the order of 3 kHz), to transmit normal digital pulse it is necessary to convert them to suitable format for reliable transmission. Most commonly used scheme is to convert digital logic 0 and 1 levels to low frequency sine waves. It is referred to as *frequency shift keying* (FSK) in which a logic '0' is represented by sine wave of low frequency, while a logic '1' is represented sine wave of high frequency. The duration of sine wave frequency is same as that of logic 1 or 0 bit time of serial data. Special circuit, termed as *Modem* is used at the sending station to convert logic level to sine wave (that is modulation). The reverse process (that is demodulation from sine wave to logic level) is implemented at the receiving end. Thus modem is used to convert serial input data to FSK output signal and also convert an FSK input to serial data output.

Therefore, as shown in Figure 7.23, a modem is installed at each end of a communications link to perform the desired signal transformations. The figure shows a computer connected to a network server. This could be a permanent connection or a dialed connection over a telephone line.

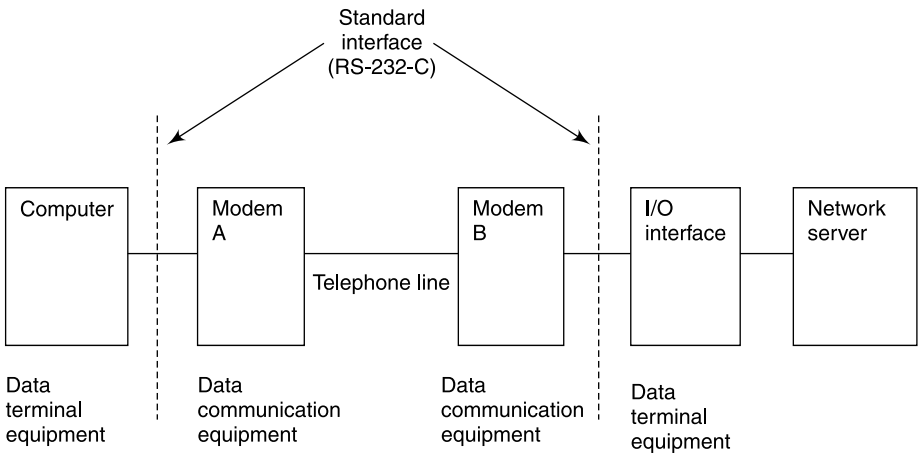


Figure 7.23 *Remote connection to a network*

Full-Duplex and Half-Duplex Links A communication link may be operated according to one of the following three schemes:

- *Simplex*: This allows transmission in one direction only.
- *Half duplex* (HDX): This allows transmission in either direction, but not at the same time.
- *Full duplex* (FDX): This allows simultaneous transmission in both directions.

The simplex configuration is useful only if the remote location contains an input or an output device, but not both. Therefore, based on economy and speed of operation required, the half or full duplex is chosen.

To obtain a half-duplex link, switches at both ends must be used to connect either the transmitter or the receiver, but not both to the line. When transmission in one direction is completed, the switches are reversed to enable transmission in the reverse direction. Control of the position of the switches is a part of the function of the devices at each end of the line.

Full-duplex operation is possible on a four-wire link, with two wires dedicated to each direction of transmission. It is also possible on a two-wire link by using two non-overlapping frequency bands. The two bands create two independent transmission channels, one for each direction of transmission.

Serial data communication generally uses either asynchronous or synchronous communication scheme. These two methods employ different techniques for transmitting data between sender and receiver.

Asynchronous Communication This is a simple scheme in which the sender and receiver use independent clock signals having the same nominal frequency. No attempt is made to guarantee that the two clocks have exactly the same phase or frequency. For data transmission within a few tens of kilobits per second, this scheme is used.

This simple scheme uses a technique called *start-stop*. To facilitate timing recovery, data are organized in groups of 6 to 8 bits, with a well-defined beginning and end. In a typical arrangement, alphanumeric characters encoded in 8 bits are transmitted as shown in Figure 7.24. The line connecting the transmitter and the receiver is in the 1 state when idle. Transmission of a character is preceded by a 0 bit, referred to as the *Start bit*, followed by eight data bits and one or two *Stop bits*. The Stop bits have a logic value of 1. The Start bit alerts the receiver that data transmission is about to begin. Its leading edge is used to synchronize the receiver clock with that of the transmitter. The Stop bits at the end delineate consecutive characters in the case of continuous transmission. When transmission stops, the line remains in the 1 state after the end of the Stop bits. It is the responsibility of the sender and receiver circuitry to insert and remove the Start and Stop bits.

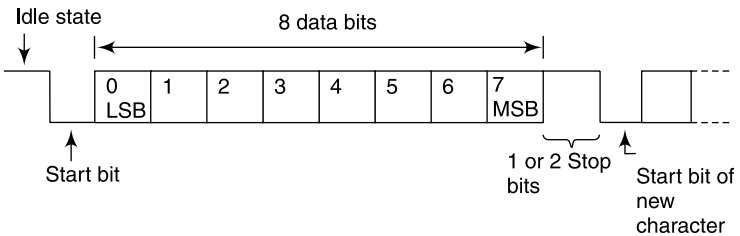


Figure 7.24 *Asynchronous serial character transmission*

A number of standard transmission rates are found in commercially available equipment ranging from 300 to 56,000 bits per second. Start-Stop transmission is used on short connections, such as the connection between the computer and the modem in Figure 7.23. For longer distances, such as for the connection between the two modems in the figure, Start-Stop can be used only at very low speeds. High-speed modems use the synchronous transmission schemes discussed in the next.

Synchronous Communication Synchronous transmission is needed to transmit data at higher speed. In this case, the receiver recovers the clock timing used by the transmitter by continuously observing the positions of the transitions in the received signal and adjusting the phase of its local clock accordingly. As a result, the receiver's clock is synchronized with the transmitter's clock and can be used to recover the transmitted data correctly. There is a wide range of techniques used to encode timing information over synchronous links. They vary in ability to make use of the bandwidth of the link and hence in the data rate they can achieve.

In the synchronous communication, after a fixed number of data bytes, a special bit pattern called SYNC is sent to mark the end. There are gaps between adjacent characters in the asynchronous communication, whereas there is no gap between adjacent characters in the synchronous communication. There is a continuous stream of data bits coming at a fixed speed in the synchronous communication scheme. Here, data are transmitted in blocks consisting of several hundreds or thousands of bits each.

The rate at which the data bits are sent is known as the *Baud Rate*, specified in BPS (Bits Per Second). The standard baud rates are: 50, 110, 134.5, 150, 300, 1200, 2400, 4800, 9600, 19200 and 38400. The baud rate chosen in a communication system depends on the quality of the transmission line and the capability of the transmitting and receiving end equipments.

7.8.4 Standard Serial Interfaces

This section introduces serial communication standards used for data transfer over long distances from a CPU to another CPU or IO devices like printer, VDU terminals, etc. Serial transmission requires only two wires to carry all the necessary data, address and control information. It does so one bit at a time and hence it is much slower than parallel transmission. However, parallel transmission over a long distance is too costly. Cost and benefit consideration has forced the introduction of some serial data transmission schemes.

RS-232C Standard The RS-232C interface is a standard interface for serial data communication; first introduced in the year 1962 and specified by the Electronics Industries Association (EIA) (RS stands for Recommended Standard).

The RS-232C interface expects a modem to be connected at the receiving and the transmitting end. The modem is DCE (Data Communication Equipment) and the computer, terminal, or printer with which the modem is interfaced is DTE (Data Terminal Equipment). The DCE and the DTE are linked via a cable whose length should not exceed 50 ft. Though not reliable, it may not affect the communication if the speed of data transfer is reduced when the distance is increased. The DTE has a 25 pin D type male connector and the DCE has a 25 pin D type female connector. However, some systems use 9-pin connectors.

The RS-232C standard follows negative logic. A logical 1 is represented by a negative voltage and logical 0 is represented by a positive voltage. The level 1 (High) varies from -3 to -15 V and the level 0 (Low) varies from +3 to +15 V. In practice the hardware circuits used for the RS-232C interface maintain the signal level at +12 V (logic 0) and at –12 V (logic 1).

Table 7.1 lists the RS-232C interface signals. The TXD carries the data bits sent by the DTE. The modem receives the TXD signal and uses it for modulating the carrier signal. The RXD is the data from the DCE to the DTE. The RXD is generated by the modem by demodulating the signal received from the other end modem.

Before sending data the DTE requests for permission from the modem by the RTS signal. When the modem finds that the communication path (consisting of telephone line, the other end modem and DTE) is ready for communication, it issues the CTS signal to the DTE as an acknowledgement for the RTS.

The DTE issues the DTR signal when it is powered-on, error-free and ready. The modem issues a DSR signal to indicate that it is powered-on and error-free.

The RI and RLSD signals are used with the dialed modem. When the telephone line is a shared (switched) line, a dialed modem is used and a telephone set is attached to the modem. When a DTE at one end wants to communicate with a DTE at the other end, it initiates a dial sequence. The modem at the sending end sends a dial tone. In response the called modem issues the RI Signal to its DTE, and sends an answer tone for 2 s to the calling modem. Then the calling modem sends an 8 ms duration tone on the telephone line. Now the called modem issues CD to its DTE. The CD is an indication to the DTE that it will soon be receiving the data sent by the other end DTE.

Table 7.1 *RS-232C signals*

Pin number	*EIA ckt.*	*Signal description*	*Common abbreviation*	*From DCE*	*To DCE*
1	AA	Frame ground	GND	X	X
2	BA	Transmitted data	TXD		X
3	BB	Received data	RXD	X	
4	CA	Request to send	RTS		X
5	CB	Clear to send	CTS	X	
6	CC	Data set ready	DSR	X	
7	AB	Signal ground/common return	SG	X	X
8	CF	Received line signal detector	RLSD OR CD	X	
9		Reserved			
10		Reserved			
11		Unassigned			
12	SCF	Secondary received line signal detector		X	
13	SCB	Secondary clear to send		X	
14	SBA	Secondary transmitted data			X
15	DB	Transmitter signal element timing (DCE)		X	
16	SBB	Secondary received data		X	
17	DD	Receiver signal element timing		X	
18		Unassigned			
19	SCA	Secondary request to send			X
20	CD	Data terminal ready	DTR		X
21	CG	Signal quality detector	SQ	X	
22	CE	Ring indicator	DCE	X	
23	CH	Data signal rate selector (DTE)			X
24	CI	Data signal rate selector (DCE)		X	
25	DA	Transmitter signal element timing (DTE)	DTE		X
26		Unassigned			

A few other serial interface standards like RS 422A and RS 423A have also evolved which aim to remove some of the drawbacks of RS 232C, such as all data and control signals are referred to same ground pin 7 (Table 7.1). By contrast, RS 422A uses two wires for each signal. Standard RS 485 is an upgraded version of RS 422A.

USB (Universal Serial Bus) The Universal Serial Bus (USB) has a lot of advantages over the RS232C serial port in respect of several parameters, such as easy installation, faster transfer rate, minimum cabling and multiple device connections. A new option for I/O interfacing is the USB, a project of a group that includes Intel and Microsoft. A single USB port can have up to 127 devices communicating at either 1.5 Megabits/second or 12 Megabits/second over a 4-wire cable. An USB device can be connected without switching-off a PC. The "Plug and Play" feature in the BIOS (Basic Input Output System) and the USB devices takes care of detection, device recognition and handling. This feature is known as *hot pluggability*. The user is totally free from the burden of the configuration procedures.

IEEE 1394 The IEEE-1394 high-performance serial bus, also known as *Firewire*, is another new interface. It allows up to 63 devices to connect to a PC, with transmission rates of up to 400 Megabits per second (i.e. up to 30 times higher bandwidth than USB). The 6-wire cables can be as long as 15 feet, with daisy chains extending to over 200 feet. It has the same Hot Pluggability feature as USB. The interface is especially popular for connecting digital audio and video devices. Unlike USB, the IEEE 1394 supports DMA transfers. IEEE-1394 expansion cards are available for PCs.

SOLVED PROBLEMS

1. *Differentiate between isolated I/O and memory mapped I/O.*

Answer

(a) In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines, one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

(b) The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.

(c) Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.

(d) Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

2. *A processor executes 50,000,000 cycles in one second. A printer device is sent 8 bytes in programmed I/O mode. The printer can print 500 characters per second and does not have a print-buffer.*
   (a) *How much time will be taken to acknowledge the character status?*
   (b) *How many processor cycles are used in transferring just 8 bytes?*

Answer

(a) Time taken to acknowledge the character status = 1/500 second = 2 ms, which is equivalent to $2 \times 50{,}000$ cycles = 100,000 cycles.

(b) Number of processor cycles used in transferring 8 bytes = $8 \times 100{,}000 = 800{,}000$.

3. *How does polling work?*

Answer

A processor is generally equipped with multiple interrupt lines those are connected between processor and I/O modules. Several I/O devices share these interrupt lines. There are two ways to service multiple interrupts: polled and daisy chaining technique.

In polling, interrupts are handled by software. When the processor detects an interrupt, it branches to a common interrupt service routine (ISR) whose function is to poll each I/O device to determine which device caused the interrupt. The order in which they are polled determines the priority of each interrupt. The highest priority device is polled first and if it is found that its interrupt signal is on, the CPU branches to the device's own ISR for I/O transfer between the device and CPU. Otherwise it moves to poll the next highest priority device.

4. *Differentiate between polled I/O and interrupt driven I/O.*

Answer

(a) In the polled I/O or programmed I/O method, the CPU stays in the program until the I/O device indicates that it is ready for data transfer, so CPU is kept busy needlessly. But, in interrupt driven I/O method, CPU can perform its own task of instruction executions and is informed by raising an interrupt signal when data transfer is needed.

(b) Polled I/O is low cost and simple technique; whereas, interrupt I/O technique is relatively high cost and complex technique. Because in second method, a device controller is used to continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer.

(c) The polled I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. However, interrupt I/O method is very useful in modern high speed computers.

5. *Discuss the advantage of interrupt-initiated I/O over programmed I/O.*

Answer

In the programmed I/O method, the program constantly monitors the device status. Thus, the CPU stays in the program until the I/O device indicates that it is ready for data transfer. This is time-consuming process since it keeps the CPU busy needlessly. It can be avoided by letting the device controller continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to an *interrupt-service-routine (ISR) or I/O routine or interrupt handler* to process the I/O transfer, and then after completion of I/O transfer, returns to the task it was originally performing. Thus, in the interrupt-initiated mode, the ISR software (i.e. CPU) performs data transfer but is not involved in checking whether the device is ready for data transfer or not. Therefore, the execution time of CPU can be optimized by employing it to execute normal programs, when no data transfer is required.

6. *What are the different types of interrupt? Give examples.*

Answer

There are basically three types of interrupts: external, internal or trap and software interrupts.

External interrupt: These are initiated through the processors' interrupt pins by external devices. Examples include interrupts by input-output devices and console switches. External interrupts can be divided into two types: maskable and non-maskable.

Maskable interrupts: The user program can enable or disable all or a few device interrupts by executing instructions EI or DI.

Non-maskable interrupts: The user program cannot disable it by any instruction. Some common examples are: hardware error and power fail interrupt. This type of interrupt has higher priority than maskable interrupts.

Internal interrupt: This type of interrupts is activated internally by exceptional conditions. The interrupts caused due to overflow, division by zero and execution of an illegal op-code are common examples of this category.

Software interrupts: A software interrupt is initiated by executing an instruction like INT n in a program, where n refers to the starting address of a procedure in program. This type of interrupts is used to call operating system. The software interrupt instructions allow to switch from user mode to supervisor mode.

7. *What are the differences between vectored and non-vectored interrupt?*

Answer

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially. The set-up time is quite large.

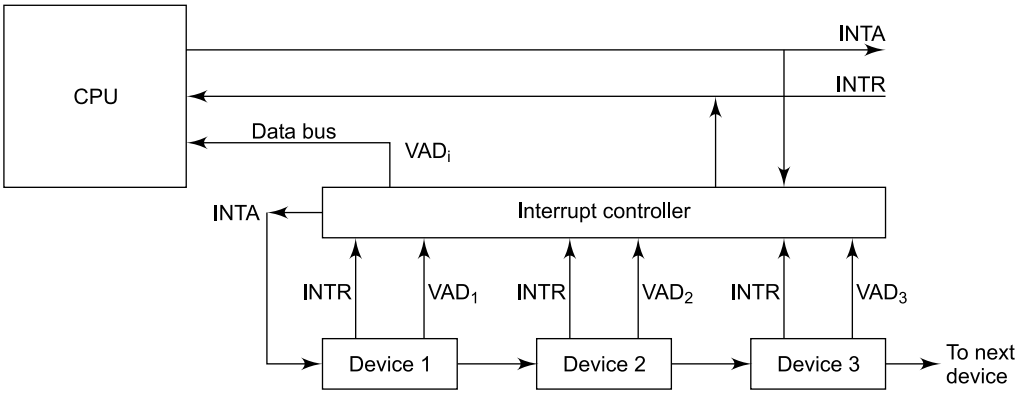
8. *Draw and discuss the schematic diagram for daisy chain polling arrangement in case of vectored interrupt for three devices.*

Answer

To implement interrupts, the CPU uses a signal, known as an *interrupt request (INTR)* signal to the interrupt controller hardware, which is connected to each I/O device that can issue an interrupt to it. Here, interrupt controller makes liaison with the CPU on behalf of I/O devices. Typically, the interrupt controller is also assigned an *interrupt acknowledge (INTA)* line that the CPU uses to signal the controller that it has received and begun to process the interrupt request by employing an ISR (interrupt service routine). Devices are connected in daisy chain fashion, as shown in the figure below, to set up a priority interrupt system.

The devices are placed in a chain-fashion with highest priority device in the first place (device 1), followed by lower priority devices. When one or more devices send interrupt signal through the interrupt controller to the CPU, the CPU then sets interrupt acknowledge (INTA) to the controller,

which in turn sends it to the highest priority device. If this device has generated the interrupt INTR, it will accept the INTA; otherwise it will pass the INTA signal to the next device until the INTA is accepted by one requestor device. When the INTA is accepted by a device, the device puts its own interrupt vector address (VAD) to the data bus using the interrupt controller.



9. *"Interrupt request is serviced at the end of current instruction cycle while DMA request is serviced almost as soon as it is received, even before completion of current instruction execution." Explain.*

Answer

In the interrupt initiated I/O, interrupt request is serviced at the end of current instruction cycle, because the processor takes part in the I/O transfer for which processor was interrupted. Thus processor will be busy in data transfer after this instruction.

But in DMA transfer, the processor is not involved during data transfer. It actually initiates the data transfer. The whole data transfer is supervised by DMA controller and at that time processor is free to do its own task of instruction execution.

10. *Suppose a disk transfers data at 5 MB per second at sustained rate. If the disk controller interrupts the processor for every 4-byte word, how often will the processor be interrupted for continuous operations?*

Answer

Since the word size is 4 bytes, here, 5 MB per second is same as 1.25×2^{20} words per second.

So, 1.25×2^{20} times per second, or in every 0.76 microseconds (approx) will be interrupted.

11. *Suppose the disk transfers data at 5 MB per second at sustained rate. If the disk controller interrupts the processor for every sector transfer, where a sector is 512 bytes, how often will the processor be interrupted for continuous operation?*

Answer

Since the sector size is 512 bytes, here. 5 MB per second is 10240 sectors per second.

So, 10240 times per second, or in every 97.7 microseconds (approx.), the process will be interrupted.

12. *Give the main reason why DMA based I/O is better in some circumstances than interrupt driven I/O?*

Answer

To transfer large blocks of data at high speed, DMA method is used. A special DMA controller is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. The data transmission cannot be stopped or slowed down until an entire block is transferred. This mode of DMA transfer is known as burst transfer.

13. *What are the different types of DMA controllers and how do they differ in their functioning?*

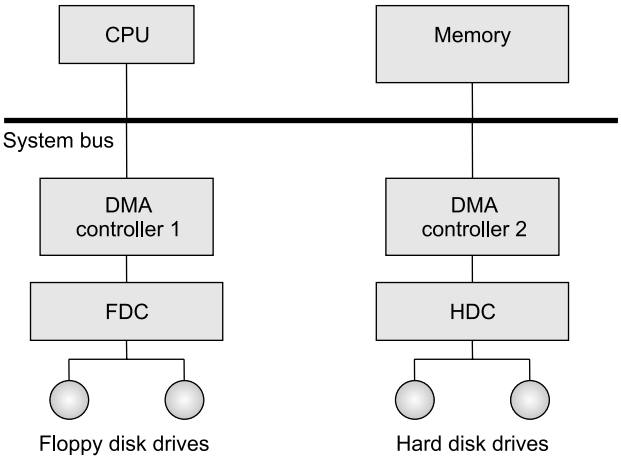
Answer

DMA controllers are of two types:

– Independent DMA controller
– DMA controller having multiple DMA-channels

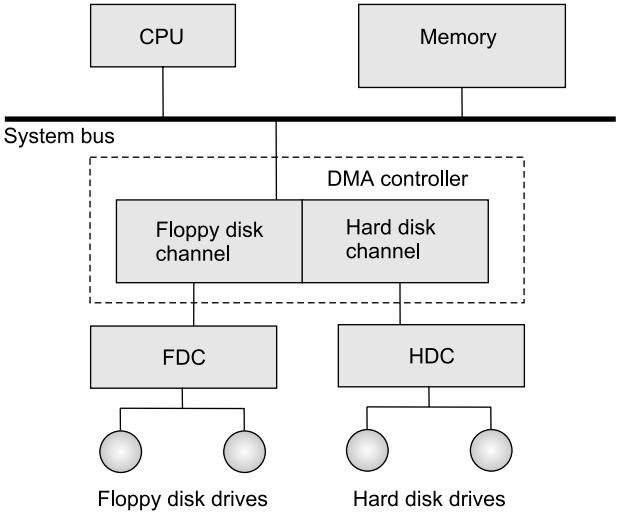
Independent DMA controller:

For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.



DMA controller having multiple DMA-channels:

In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels, each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC).



14. *What are the advantages and disadvantages of an asynchronous transfer?*

Answer

Advantages:

(a) High degree of flexibility and reliability can be achieved because the successful completion of a data transfer relies on active participation by both communicating units.
(b) Delays in transmission or in interface circuits are taken care of.
(c) There is no need of clock for synchronization of source and destination.

Disadvantages:

(a) A slow speed destination unit can hold up the bus whenever it gets a chance to communicate.
(b) If one of the two communicating devices is faulty, the initiated data transfer cannot be completed.
(c) Since handshaking involves exchange of signals twice, the data transfer is limited.

15. *Suppose n devices are connected in daisy chained mode. An ith device executes the interrupt service routine (ISR) in a time period T_i. What is the maximum waiting time of ith device? Assume that each device executes an ISR only once, context switch time can be neglected and 0^{th} device has highest priority.*

Answer

In daisy chained method, an ith device gets the service only after all devices 0 to i –1 are serviced. Now since each device executes an ISR only once, therefore maximum possible waiting time of ith device = $T_0 + T_1 + \ldots + T_{i-1}$.

16. *Using asynchronous serial format of 7-bit data, odd parity and two stop bits at bit rate of 1500 bits / sec, the message START is to be transmitted. What will be the time for transmission?*

Answer

Size of each character = 7 (data bits) + 1 (start bit) + 2 (stop bits) + 1 (parity bit)
= 11 bits

So, to transmit message START, total no. of bits transmitted = 5 × 11 = 55 bits
Bit rate = 1500 bits / sec
Therefore, time taken for transmission = 55 / 1500 sec = 36.67 msec

17. *Suppose a system has following specifications:*
    - *400 ns memory cycle time for read/write*
    - *3 microsec for execution of an instruction on average*
    - *interrupt service routine (ISR) consists of seven instructions*
    - *each byte transfer requires 4 cycles (instructions)*
    - *50% of the cycles use memory bus*

    Determine the peak data transfer rate for—(a) *programmed I/O* (b) *interrupt I/O, and* (c) *DMA.*

Answer
Given,
Instruction execution time (average) = 3 μ sec
Memory read/write cycle = 400 ns

(a) Instructions/IO byte = 4
In programmed I/O, CPU is continuously polling the I/O devices
Therefore, the peak data transfer rate = CPU speed/4
$= 1/(3 \times 10^{-6} \times 4)$ bytes/sec
= 83 Kbytes/sec

(b) ISR consists of seven instructions.
In interrupt driven I/O, the peak data transfer rate = CPU speed/7
$= 1/(3 \times 10^{-6} \times 7)$ bytes/sec
= 47.6 Kbytes/sec

(c) DMA:
Under burst transfer, peak data transfer rate = 1/memory cycle time
$= 1 / (400 \times 10^{-9})$ bytes/sec
= 2.5 Mbytes/sec
Under cycle stealing, 50% of the cycles use memory bus
Peak data transfer rate = No. of memory cycles/memory cycle time
$= 0.5/(400 \times 10^{-9})$ bytes/sec
= 1.25 Mbytes/sec

REVIEW QUESTIONS

Group A

1. Choose the most appropriate option for the following questions:
   (i) An I/O interface is
   (a) a hardware unit used for transferring data with central computer (CPU and memory) on behalf of a peripheral device
   (b) a software used for transferring data with central computer (CPU and memory) on behalf of a peripheral device

(c) a firmware used for transferring data with central computer (CPU and memory) on behalf of a peripheral device
(d) none.

(ii) In order to execute a program, instructions must be transferred from memory along a bus to the CPU. If the bus has 8 data lines, at most one 8-bit data can be transferred at a time. How many memory accesses would be needed in this case to transfer a 32-bit instruction from memory to the CPU?
(a) 1 (b) 2 (c) 3 (d) 4

(iii) Suppose that a bus has 16 data lines and requires 4 cycles of 250 ns each to transfer data. The bandwidth of this bus would be 2 Megabytes/second. If the cycle time of the bus was reduced to 125 ns and the number of cycles required for transfer stayed the same, what would the bandwidth of the bus?
(a) 1 Megabyte/second (b) 4 Megabytes/second
(c) 8 Megabytes/second (d) 2 Megabytes/second

(iv) Any computer must at least consist of
(a) data bus (b) address Bus (c) control Bus (d) all of the above.

(v) An I/O command
(a) is generated by CPU to communicate with a particular peripheral
(b) is generated by a peripheral which wants to communicate with CPU or memory
(c) initiates an I/O transfer
(d) is provided through data bus.

(vi) An I/O driver is
(a) hardware unit used for transferring data with central computer (CPU and memory) on behalf of a peripheral device
(b) software module that issues different commands to the I/O interface
(c) common program for all peripherals attached with the computer
(d) modifiable by the general users.

(vii) The objective of the IOP (I/O processor) is to use
(a) common buses (address, data and control) for the transfer of information between memory and I/O devices
(b) common address and data buses, but separate control lines
(c) a separate bus
(d) none.

(viii) The main advantage of memory mapped I/O technique is
(a) no extra instructions are required
(b) no extra control signals are required
(c) the operations are done directly at the I/O register address
(d) DMA operations are fast.

(ix) Synchronous mode of data transfer
(a) is suitable for high speed devices
(b) occurs between two communicating devices, where one device is master and another is slave
(c) is suitable for slow speed devices
(d) both (b) and (c).

(x) Asynchronous data transfer
  (a) can be initiated by source or destination device
  (b) is initiated by source device
  (c) is initiated by destination device
  (d) is controlled by clock and can be initiated by source or destination device.

(xi) Asynchronous mode of data transfer
  (a) is suitable for high speed devices
  (b) occurs between two communicating devices, where one device is master and another is slave
  (c) is suitable for slow speed devices
  (d) both (b) and (c).

(xii) When bulk data transfer is needed
  (a) programmed I/O technique is used
  (b) interrupt-initiated I/O technique is used
  (c) DMA mode is used
  (d) IOP is used.

(xiii) During data transfer in programmed I/O method
  (a) the CPU can be busy on its own task of instruction execution
  (b) the CPU is totally idle
  (c) the CPU monitors the interface status
  (d) none.

(xiv) When a device interrupts, the CPU finds the service routine address for processing from the
  (a) interrupt vector start address
  (b) interrupt vector location defined as per the device address
  (c) program already under execution
  (d) device control register.

(xv) DMA operations need
  (a) switching logic between the I/O and system bus
  (b) I/O bus
  (c) special control signals to CPU such as hold and hold acknowledge
  (d) no CPU control signals.

(xvi) DMA operations are initiated by
  (a) DMA controller (b) I/O interface (c) I/O driver (d) CPU.

(xvii) Bus arbitration means
  (a) master-slave synchronous or asynchronous data transfer
  (b) a process by which a bus controller controls the bus most of the time
  (c) a process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting device
  (d) a process to give bus accesses among many devices by polling the requesting device.

(xviii) Half-duplex communication allows transmission
  (a) in either direction, but not at the same time
  (b) in one direction only
  (c) simultaneous transmission in both directions
  (d) none of them

(xix) The handshaking technique
- (a) is used in synchronous data transfer
- (b) is used in asynchronous data transfer and uses two control signals in opposite direction
- (c) works even if one the communicating device gets faulty in the midway of data transfer
- (d) is not much flexible.

(xx) When processor architecture disables execution of other devices interrupt during execution of an ISR
- (a) higher priority devices must have short ISRs
- (b) interrupts should be used only when absolutely necessary
- (c) interrupt routines should be made as short as possible
- (d) DMA transfer be used.

(xxi) The data transfer technique in which no start and stop bits are used is
- (a) synchronous
- (b) asynchronous
- (c) both synchronous and asynchronous
- (d) neither synchronous nor asynchronous

(xxii) The bus arbitration technique in which the count lines are connected to all units is
- (a) Daisy chaining
- (b) polling
- (c) independent requesting
- (d) none.

(xxiii) The bus arbitration technique in which separate BUS REQUEST and BUS GRANT lines for every unit are available is
- (a) daisy chaining
- (b) polling
- (c) independent requesting
- (d) none

(xxiv) The bus arbitration technique in which all units are connected to a single BUS REQUEST line is
- (a) daisy chaining
- (b) polling
- (c) independent requesting
- (d) none

Group B

2. What is I/O interface or I/O controller? Why do we need I/O interfaces to the peripherals?
3. Discuss in brief how the data transfer take place between CPU and a peripheral.
4. What is I/O command? What are different types of I/O command?
5. Classify the I/O controllers. Give one example of each.
6. What is I/O driver? What are the functions of I/O drivers?
7. What is the difference between memory-mapped I/O and I/O-mapped I/O? Also state the advantages and disadvantages of each.
8. Describe the synchronous mode of data transfer with merits and demerits.
9. Describe the asynchronous mode of data transfer with merits and demerits. What are different techniques of asynchronous data transfer? Explain them with relative merits and demerits.

10. What are the different modes of data transfer between central computer and I/O devices? Briefly discuss each.
11. Explain the programmed I/O technique in brief. What are the advantages and disadvantages of using this method?
12. Discuss how data transfer from I/O device to memory occurs in programmed I/O method.
13. Explain the interrupt-initiated I/O technique in brief. What are the advantages and disadvantages of using this method?
14. Differentiate between vectored interrupt and non-vectored interrupt.
15. Compare between maskable and non-maskable interrupts.
16. Discuss how nesting of interrupts is handled.
17. Explain how an I/O operation based on an interrupt is more efficient than I/O operation based on a programmed mode of operation.
18. When a device interrupt occurs, how does the CPU determine which device issued the interrupt?
19. Explain DMA mode of data transfer. Where does DMA mode of data transfer find its use?
20. What are different modes of DMA transfer? Compare them.
21. When a DMA controller takes control of a bus, and while it retains control of the bus, what does the CPU do?
22. What is bus arbitration? What are different methods of bus arbitration? Explain them.
23. What is cycle stealing and how does it enhance the performance of the system?
24. In virtually all computers having DMA modules, DMA access to main memory is given higher priority than CPU access to main memory. Why?
25. Write short note on: input-output processor (IOP).
26. Discuss about the asynchronous serial data transmission with a diagram.
27. What is the main advantage of the synchronous serial transmission over its counterpart? Discuss about the synchronous serial data transmission in brief.
28. Write a short note on RS-232C standard communication interface.

APPENDIX
Digital Devices, Logic Design and Assembly Language Programming

Here, we want to discuss some important digital devices like logic gates, flip-flops, registers, full adders, and multiplexers etc, which are used throughout the book. Also, we discuss the assembly language basics.

A.1 LOGIC GATES

The digital systems consist of some primitive logic circuits known as logic gates. These circuits operate on binary variables that assume one of two distinct logic states, usually called 0 and 1. Logic gates are electronic circuits because they are made up of a number of electronic devices and components. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression. The input-output relationship of the binary variables for each gate can be represented in tabular form by a truth table. There are three basic types of gates—AND, OR and NOT (or Inverter). The other gates like XOR, NAND and NOR gates can be derived from these basic gates.

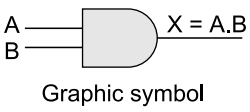
A.1.1 AND Gate

The AND gate is a circuit with two or more inputs and a single output. The output assumes the logic 1 state, only when each of its inputs is at logic state 1.

The output assumes the logic state 0, when at least one of its inputs is at logic 0 state. The graphic symbols and truth tables of two input and three-input AND gates are shown in Fig. A.1. The symbol for AND operation is '.' (dot) or '$\wedge$' and thus A AND B can be written as A.B or A $\wedge$ B.

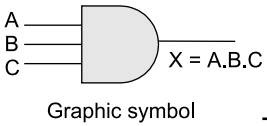
A.1.2 OR Gate

The OR gate, like AND gate, has two or more inputs and only one output. The output assumes the logic state 1, if at least one of the inputs is at logic state 1 and output assumes logic state 0 only when



Truth Table		
Inputs		Outputs
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

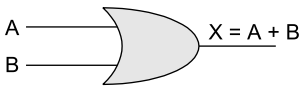
(a) : Two-input AND gate



Truth Table			
Inputs			Outputs
A	B	C	X
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

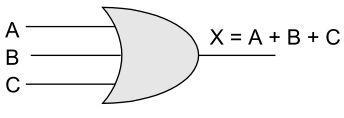
(b) Three-input AND gate

Figure A.1 *The AND gate*



Truth Table		
Inputs		Outputs
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(a) Two-input OR gate



Truth Table			
Inputs			Outputs
A	B	C	X
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

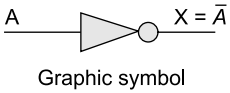
(b) Three-input OR gate

Figure A.2 *The OR gate*

all inputs are at logic 0. The graphic symbols and truth tables of two input and three-input OR gates are shown in Fig. A.2. The symbol for OR operation is '+' (plus) or '$\vee$' and thus A AND B can be written as A.B or A $\vee$ B.

A.1.3 NOT Gate

The NOT gate has only one input and only one output. It is sometimes referred to as *inverter*. It is a device whose output is always the complement of its input. That is, the output assumes logic state 1 if the input is at logic state 0. The output assumes logic state 0 if the input is at logic state 1. The graphic symbol and truth table of NOT gate are shown in Fig. A.3. The symbol for NOT operation is '$\bar{}$' (bar) '$\neg$'and thus NOT can be written as $\bar{A}$ or $\neg A$.



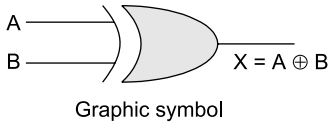
Truth Table	
Inputs	Outputs
A	X
0	1
1	0

Figure A.3 *The NOT gate*

A.1.4 XOR (Exclusive-OR) Gate

The XOR gate has two inputs and one output. The output assumes logic state 1 when one and only one of its two inputs is at logic state 1. The output assumes logic state 0 when both inputs are at logic state 0 or both at logic 1. In other words, the output assumes logic state 1 only when its two inputs are different and its output assumes logic state 0 when both inputs are same.

The graphic symbol and truth table of XOR gate are shown in Fig. A.4. The symbol for XOR operation is '⊕' and thus A XOR B can be written as $A \oplus B$, which is logically equivalent to $A.\overline{B} + \overline{A}.B$



Truth Table		
Inputs		Outputs
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

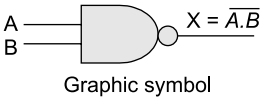
Figure A.4 *The XOR gate*

A.1.5 NAND and NOR Gates

The two most widely used gates in real circuits are the NAND and NOR gates. These two gates are called universal gates because other logic operations can be realized using either NAND gates or NOR gates. Both NAND and NOR gates can perform all three basic logic operations (AND, OR, NOT).

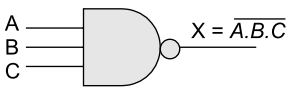
NAND Gate The NAND gate is combination of AND and NOT gates, which is shown in Fig. A.5.(a). The output assumes logic state 0, only when each of the inputs assumes a logic state 1. The output assumes logic state 1, for all other combination of inputs. The graphic symbols and truth

(a) Realization of NAND gate



Truth Table		
Inputs		Outputs
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b) Two-input NAND gate



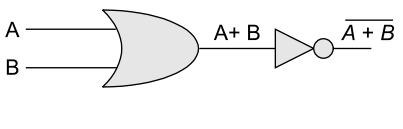
Truth Table			
Inputs			Outputs
A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(c) Three-input NAND gate

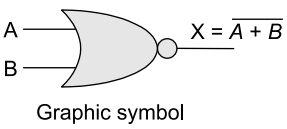
Figure A.5 *The NAND gate*

tables of two input and three-input NAND gates are shown in Figs A.5.(b) and A.5.(c) respectively. The expression for the output of the NAND gate can be written as $\overline{ABC\ldots}$

NOR Gate The NOR gate is the combination of OR gate and NOT gate, which is shown in Fig A.6.(a). The output assumes logic state 1, only when each of the inputs assumes a logic state 0. The output assumes logic state 0, for all other combination of inputs. The graphic symbols and truth tables of two input and three-input NOR gates are shown in Figs A.6.(b) and A.6.(c) respectively. The expression for the output of the NOR gate can be written as $\overline{A+B+C+\ldots}$.



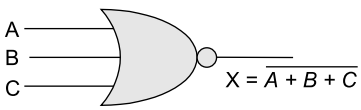
(a) Realization of NOR gate



Graphic symbol

(b) Two-input NOR gate

Truth Table		
Inputs		Outputs
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0



Graphic symbol

(c) Three-input NOR gate

Truth Table			
Inputs			Outputs
A	B	C	X
0	0	0	1
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	0

Figure A.6 *The NOR gate*

Realization of Other Logic Functions Using NAND/NOR Gates The NAND and NOR gates are said to be universal gates, because using only NAND or NOR gates other logic operation can be realized. It is easier to fabricate NAND or NOR gates using IC (integrated circuit) technology than AND or OR gates. In addition to that, NAND and NOR gates consume less power. As a consequence, they are used as fundamental building blocks in fabricating the digital devices. Figure A.7 shows how other gates can be realized using NAND gates.

Similarly, different gates can be realized using NOR gates.

A.2 CLASSIFICATION OF LOGIC CIRCUITS

There are two types of logic circuits: (i) combinational and (ii) sequential.

The combinational circuit is one whose output state at any instant is dependent only on the states of the inputs at that time. Thus, combinational circuit has no memory. Combinational circuits are

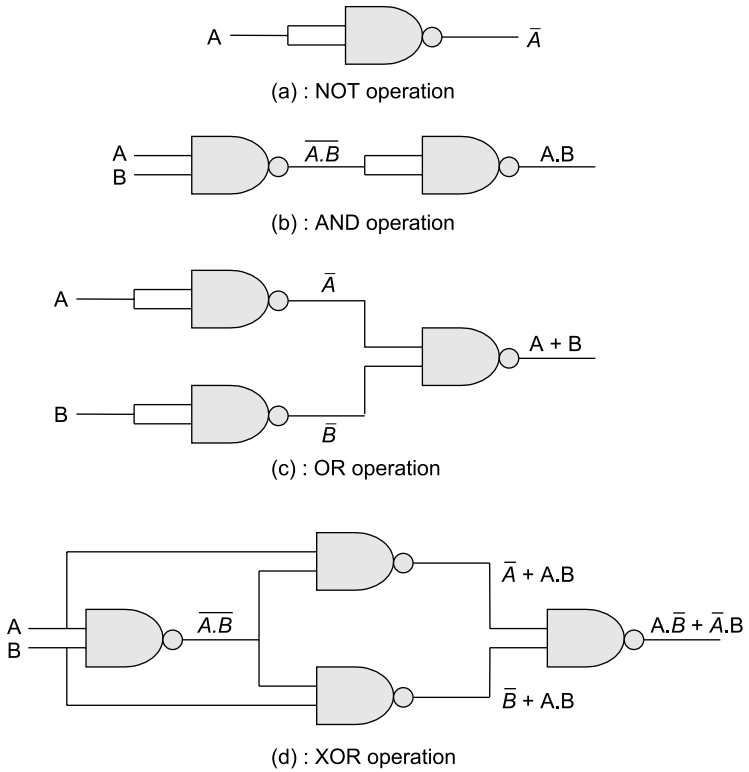


Figure A.7 *Realization of different gates using NAND gates*

realized by logic gates. Examples of combinational circuits are adder, subtractor, multiplexer, decoder, demultiplexer, etc.

The sequential circuit is one whose output state at any instant is dependent not only on the present states of inputs at that instant, but also on the prior input states. Thus, sequential circuit has memory. The sequential circuits are realized by logic gates in addition with flip-flops. Examples of sequential circuits are registers, counters, etc.

A.2.1 Combinational Circuits

Half Adder A half adder (HA) is an arithmetic circuit which adds two binary digits and produces two output bits: sum bit and carry bit. According to the binary addition rules, the sum (S) bit and the carry (C) bit are given by the truth table:

Inputs		*Outputs*	
A	*B*	*S*	*C*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From this table, we conclude that the sum (S) bit is obtained by XOR operation of A and B, and the carry (C) bit is obtained by AND operation of A and B. Therefore,

$$S = A.\overline{B} + \overline{A}.B = A \oplus B$$
$$C = A.B$$

Hence, half adder (HA) can be realized by using one XOR gate and one AND gate, as shown in Fig. A.8 (a). Figure A.8 (b) shows the block diagram of half adder.

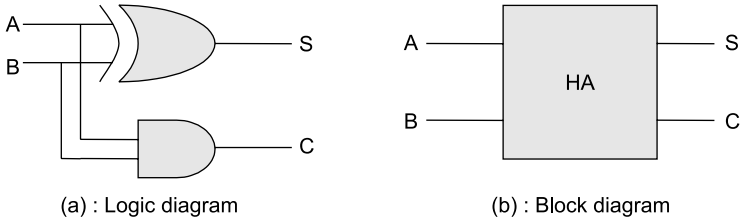


Figure A.8 *The half adder*

Full Adder A full adder (FA) is an arithmetic circuit that adds two binary digits and a carry bit and produces a sum bit and a carry bit. When two n-bit numbers are to be added, there may be a carry from one stage to the next stage. The carry generated from one stage is to be added to the next stage. Then we use the full adder which adds two normal input-bits A and B and the carry from the previous stage called the carry-in C_{in}. The outputs of the full adder are the sum bit S and the carry bit called C_{out}. The truth table of a full adder is as:

Inputs			*Outputs*	
A	*B*	C_{in}	*S*	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table, the expressions for sum S and carry-out C_{out} can be written as:

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C_{in}} + A.\overline{B}.\overline{C_{in}} + A.B.C_{in} = A \oplus B \oplus C_{in} \quad (1)$$

$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C_{in}} + A.B.C_{in} = A.B + (A + B).C_{in}$$
$$= A.B + (A \oplus B).C_{in} \quad (2)$$

Now, expressions (1) and (2) can be realized using XOR, AND and OR gates, to get sum S and carry-out C_{out}, which is shown in Fig. A.9. It is worth noting that a full adder is a combination of two half adders, as shown in Fig. A.9. The block diagram of a full adder (FA) is shown in Fig. A.10.

BCD Adder The BCD number is introduced in Chapter 2. When two BCD digits are added together with a possible carry from previous stage, the sum can be maximum 19. Since, maximum

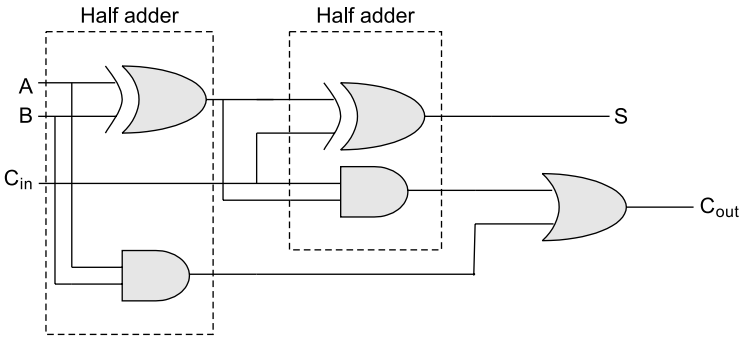


Figure A.9 *Logic diagram of full adder*

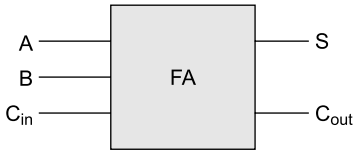


Figure A.10 *Block diagram of full adder*

BCD digit is 9, thus 9 + 9 + 1 (carry from previous stage) = 19. When two BCD digits are added using 4-bit binary adder, the sum will be in binary form and the range of sum will be 0 to 19 in equivalent decimal. The binary sum and BCD sum in this range is shown in Table A.1. By observing the table, it is clear that the two sums are identical upto decimal 9 and thus no correction is required. When sum is greater than 9, we find an invalid BCD sum representation. The binary 0110 (i.e. 6) is added to the binary sum to get the valid BCD sum representation. This will generate an output carry as required.

The BCD adder must be able to perform the following:

(a) Add two BCD digits using one 4-bit parallel adder.

(b) If the binary sum of this adder is greater than or equal to binary 1010 (i.e. decimal 10), add binary 0110 (decimal 6) to the sum and generates the required output carry.

From the table, the logic circuit for necessary correction can be derived. It is clear that when binary sum from 4-bit parallel adder is greater than 01001 (i.e. decimal 9) we need a correction. This can be said in other words as, BCD carry X will be in high state when either of the following conditions occurs:

1. When C = 1 (sums greater than or equal to 16)
2. When $S_3 = 1$ and either S_2 or S_1 (sums in the range 10 to 15). Since decimal numbers 8 and 9 also have $S_3 = 1$. To distinguish decimal numbers 10 to 15 from 8 and 9, we use S_2 S_1 bit positions.

Table A.1 *Relation of binary sum and BCD sum*

Binary sum	*BCD sum*	
$C\ S_3\ S_2\ S_1\ S_0$	$X\ \Sigma_3\ \Sigma_2\ \Sigma_1\ \Sigma_0$	*Decimal*
0 0 0 0 0	0 0 0 0 0	0
0 0 0 0 1	0 0 0 0 1	1
0 0 0 1 0	0 0 0 1 0	2
0 0 0 1 1	0 0 0 1 1	3
0 0 1 0 0	0 0 1 0 0	4
0 0 1 0 1	0 0 1 0 1	5

(*Contd*)

(*Contd*)

0 0 1 1 0	0 0 1 1 0	6
0 0 1 1 1	0 0 1 1 1	7
0 1 0 0 0	0 1 0 0 0	8
0 1 0 0 1	0 1 0 0 1	9
0 1 0 1 0	1 0 0 0 0	10
0 1 0 1 1	1 0 0 0 1	11
0 1 1 0 0	1 0 0 1 0	12
0 1 1 0 1	1 0 0 1 1	13
0 1 1 1 0	1 0 1 0 0	14
0 1 1 1 1	1 0 1 0 1	15
1 0 0 0 0	1 0 1 1 0	16
1 0 0 0 1	1 0 1 1 1	17
1 0 0 1 0	1 1 0 0 0	18
1 0 0 1 1	1 1 0 0 1	19

Thus, the condition for correction and the output carry can be expressed as:

$$X = C + S_3.S_2 + S_3.S_1$$

The BCD adder can be implemented as (see Fig. A.11): First two BCD numbers are added using simple 4-bit parallel adder, which produces the sum as C S_3 S_2 S_1 S_0. Then a correction is needed when this sum is equal to or greater than 10, which is implemented by using the above expression for X. Lastly, another 4-bit parallel adder is used to add binary 0110, produced by the correction logic, with the sum S_3 S_2 S_1 S_0. This will produce the required BCD sum output Σ_3 Σ_2 Σ_1 Σ_0. The carry output X is used as the carry to the next BCD adder.

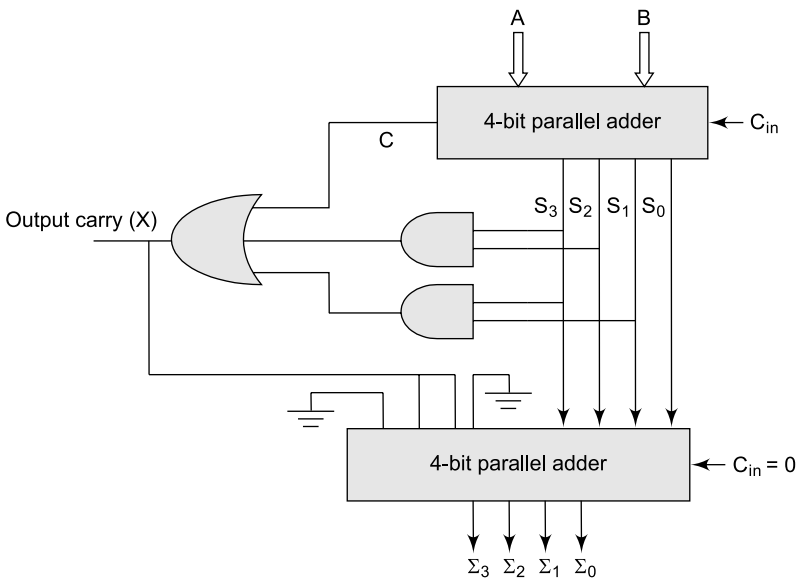
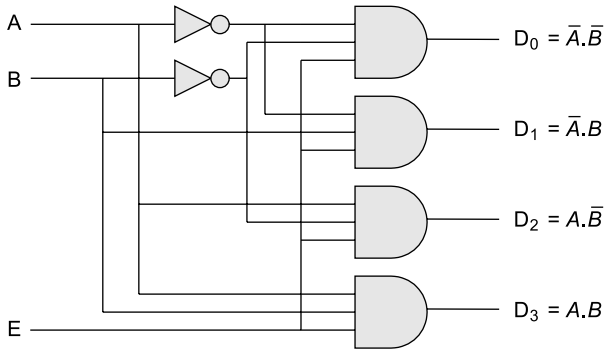


Figure A.11 *BCD adder*

Decoders A decoder is combinational circuit that converts n-bit coded information into a maximum of 2^n unique outputs. Thus, only one of 2^n unique outputs is activated for each of the possible

combinations of n-bit inputs. The logic diagram of a 2- to-4 line decoder is shown in figure A.12. It has two-input lines A and B and four-output lines D_0, D_1, D_2 and D_3. The truth table of the decoder is shown in Fig. A.12(b).

Some decoders have one or more *enable* inputs to control the operation of the circuit. For example, the 2-to-4 line decoder in Fig. A.12 has one enable line E. When E is high, the decoder is activated and when E is low, it is disabled.



(a) : Logic diagram

Enable	*Inputs*		*Outputs*			
E	*A*	*B*	D_0	D_1	D_2	D_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

(b) Truth Table for 2-to-4 line decoder

Figure A.12 *A 2-to-4 line decoder*

Encoder An encoder is a combinational circuit that performs the "reverse" function of the decoder. An encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code corresponding to the input value. An encoder has a number of input lines, out of which only one is activated at a given time.

An octal-to-binary encoder is an example of it. An octal-to-binary encoder (8-to-3 encoder) accepts 8 input lines and produces a 3-bit output code corresponding to the activated input. Figure A.13 shows the truth table and the logic circuit for an octal-to-binary encoder.

From the truth table we see that A_2 is a 1 if any of the digits D_4 or D_5 or D_6 or D_7 is a 1.
Thus,

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Similarly,

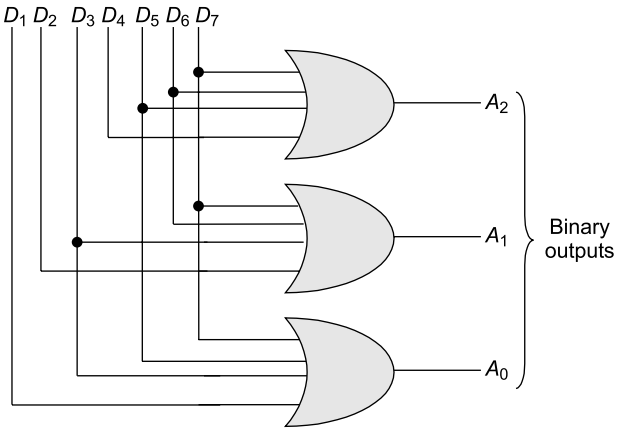
$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_0 = D_1 + D_3 + D_5 + D_7$$

We see that D_0 is not present in any of the expressions. So, D_0 can have any logic value (i.e. don't care).

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

Truth table

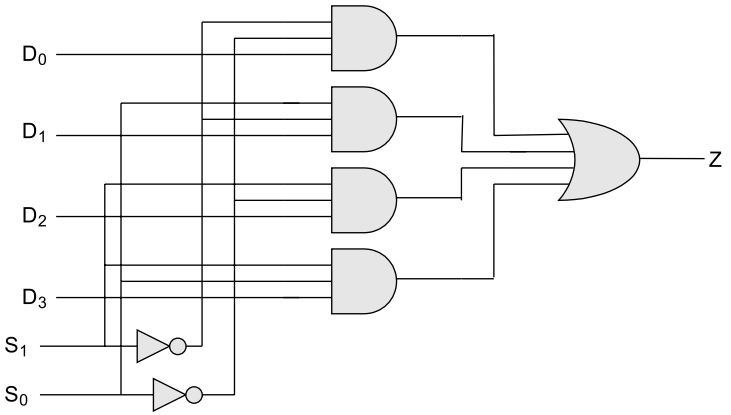


Logic diagram

Figure A.13 *An octal-to-binary encoder*

Multiplexers A multiplexer (simply known as MUX) is a combinational circuit that accepts several data inputs and directs only one of them at a time to go through the single output line. The selection of a particular data input to the output line is controlled by a set of select inputs. A 2^n-to-1 multiplexer has 2^n input data lines, one output line and n input select lines whose bit combination selects which input data is routed to the output line.

The logic diagram of a 4-to-1 line MUX is shown in Fig. A.14. Four data inputs are D_0, D_1, D_2, D_3 and two select lines are S_0, S_1. The logic value applied to the select inputs S_0, S_1 determine which AND gate is enabled, so that its data input the OR gate to the output. The function table of the multiplexer is shown in Fig. A.14(b).



(a) Logic diagram

Select inputs		*Output*
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

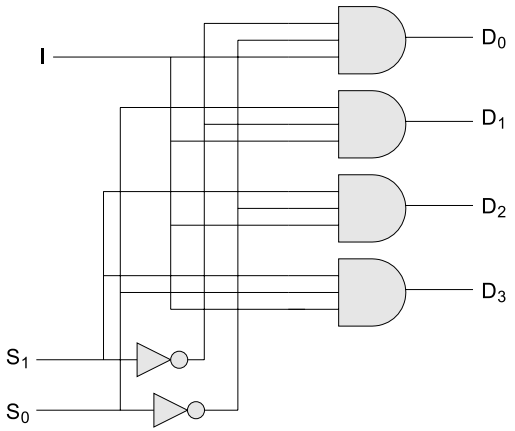
(b) Function table

Figure A.14 *A 4-to-1 line multiplexer*

Demultiplexers A demultiplexer (simply DEMUX) is a combinational circuit that performs reverse operation of the multiplexer. It accepts only one input data and transfers the input data through one of several output lines. Thus, a demultiplexer is a 1-to-2^n device, whereas, a multiplexer is a 2^n-

to-1 device. Like MUX, a DEMUX has n select input lines whose bit combination selects which output line is used to transfer the input data.

The logic circuit and function table for a 1-to-4 line DEMUX are shown in Fig. A.15. The two select lines S_0 and S_1 enable only one AND gate at a time and the data (I) on the input line passes through the selected gate to the associated output line.



(a) Logic diagram

Select inputs		Outputs			
S_1	S_0	D_0	D_1	D_2	D_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

(b) Function table

Figure A.15 *A 1-to-4 line demultiplexer*

A.2.2 Sequential Circuits

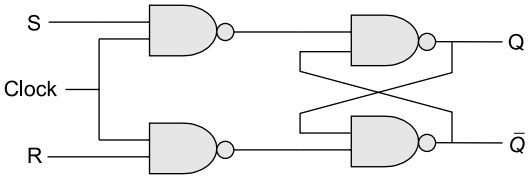
Generally, sequential circuits are of sequential type. In synchronous sequential circuits, signals are used to change the states of the storage cells only at the discrete instants of time. Synchronization is achieved by a timing device known as clock pulse generator that generates clock pulses.

Flip-Flops A flip-flop is a binary storage cell capable of storing one bit of information. It can retain its state indefinitely until its state is changed by applying the proper triggering signal. A flip-flop using the clock signal is called the *clocked flip-flop*. It has two outputs, one labeled Q for the normal value and another labeled $\bar{Q}$ for the complement value of the stored bit.

Even though a logic gate has no storage capability, several logic gates can be connected together to form a storage cell. Depending on the interconnection pattern of logic gates and number of inputs used, the flip-flops can be of different types. The common types of flip-flops are described next.

SR Flip-Flop The SR flip-flop has three input lines, one line labeled S for set, one line labeled R for reset and another line labeled C for clock signal. It has two inputs, one labeled Q for the normal value of the output and another labeled $\bar{Q}$ for the complement value of it. A realization of a SR flip-flop using universal gate, NAND and its truth table is shown in Fig. A.16.

When no clock pulse is applied, the output of the flip-flop cannot change irrespective of the input values at S and R. When clock pulse is applied, the flip-flop will be affected according to the input values at S and R. Thus, the flip-flop is enabled when clock is applied to the circuit. The truth table is given for enabled flip-flop. Q(t) and Q(t+1) indicate the binary state of the Q output at given time t and next time t+1 respectively. The SR flip-flop has invalid (indeterminate) state when both S and R are logic '1' simultaneously.



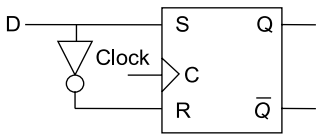
(a) : Realization using NAND gates (b) : Graphic symbol

Inputs		Output	
S	*R*	*Q(t+1)*	*Remarks*
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Indeterminate

(c) Truth table

Figure A.16 *The SR flip-flop*

D Flip-Flop The D (data) flip-flop has only one input line and is obtained from SR flip-flop by putting one inverter between the S and R input lines. Thus a D flip-flop can be realized using SR flip-flop as shown in Fig. A.17. If the input D has logic state 1, the output of the flip-flop assumes the state 1. If the input D has logic state 0, the output of the flip-flop assumes the state 0. This is shown in the truth table.



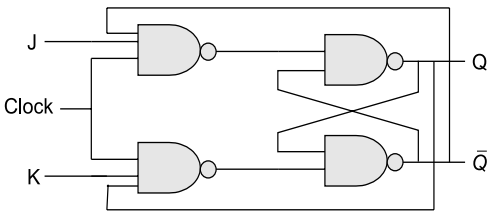
(a) Realization using SR flip-flop

Input	Output	
D	*Q(t + 1)*	*Remarks*
0	0	Reset
1	1	Set

(b) Truth table

Figure A.17 *D flip-flop*

JK Flip-Flop The JK flip-flop is versatile and most widely used flip-flop. The operation of JK flip-flop is same as the SR flip-flop, except that it has no indeterminate state for logic input '1' for both the inputs. In this situation, the state of the output is changed and the output state is the complement of the previous state. A realization of a JK flip-flop using universal gate, NAND and its truth table is shown in Fig. A.18.



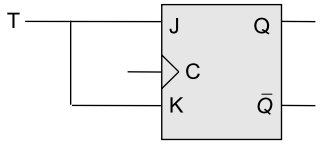
(a) Realization using NAND gates (b) Graphic symbol

Inputs		Output	
J	*K*	*Q(t+1)*	*Remarks*
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	$\bar{Q}$ (t)	Toggle

(c) Truth table

Figure A.18 *The JK flip-flop*

T Flip-Flop A T flip-flop has one input line, labeled T for toggle. T flip-flop acts as toggle switch. Toggle means switching over to the opposite state. It can be realized using a JK flip-flop with input T = J = K, as shown in Fig. A.19.



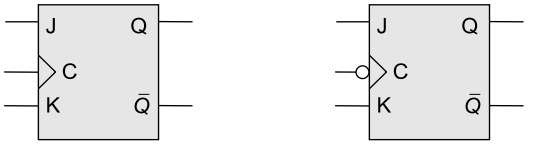
(a) Realization using JK flip-flop

Input *T*	*Output* *Q(t+1)*	*Remarks*
0	Q(t)	No change
1	$\bar{Q}$ (t)	Toggle

(b) Truth table

Figure A.19 *T flip-flop*

Triggering of Flip-Flops Clocked flip-flops can be positive edge-triggered or negative edge-triggered. Positive edge-triggered flip-flops are those in which state transitions take place only at the positive going (i.e. logic 0 to logic 1) edge of the clock pulse. Negative edge-triggered flip-flops are those in which state transitions take place only at the negative going (i.e. logic 1 to logic 0) edge of the clock pulse. Positive edge-triggering is indicated by a 'triangle' symbol at the clock terminal of the flip-flop. Negative edge-triggering is indicated by a 'triangle' symbol with a bubble at the clock terminal of the flip-flop. The triggering of flip-flops is shown in the Fig. A.20.



(a) Positive edge-triggered (b) Negative edge-triggered

Figure A.20 *Triggering of flip-flops*

Shift Registers A register is a collection of flip-flops each capable of storing one bit of information. An 8-bit register has 8 flip-flops and is capable storing 8-bit data. In a shift register, the flip-flops are connected in series. The output of each flip-flop is connected to the input of the adjacent flip-flop in the register. The content of a shift register can be shifted within the register without changing the order of the bits. Data can be shifted one position left or right at a time when one clock is applied to the register. Based on the ways of loading into and reading out the data, shift registers can be classified into four categories:

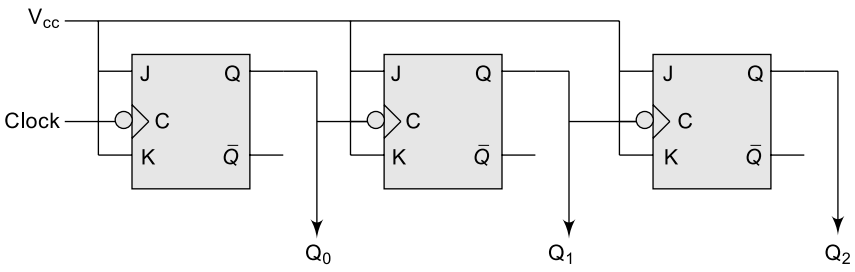
- Serial-in, serial-out
- Serial-in, parallel-out
- Parallel-in, serial-out
- Parallel-in, parallel-out

Counters A digital counter (simply counter) is a set of flip-flops whose states change in response to clock pulses applied at the input to the counter. The flip-flops are interconnected in such a way that their combined state at any time is the binary equivalent to the total number of pulses applied up to that time. Thus, counters are used to count clock pulses or time interval.

Counters can be of two types: *asynchronous* and *synchronous*.

In asynchronous counters (also called ripple counters), the flip-flops (FFs) within the counter do not change the states at exactly the same time. This is because FFs are not clocked simultaneously. Synchronous counters are counters in which all the FFs are clocked simultaneously. Synchronous counters are faster than asynchronous counters, because the propagation delay is less.

A counter can be *up-counter* or *down-counter*. An up-counter is a counter which counts in the upward direction, i.e. 0, 1, 2, …, N-1. A down-counter is a counter which counts in the downward direction, i.e. N-1, N-2, …, 1, 0. Each counts of the counter is called the *state* of the counter. The total number of states through which the counter can progress is called the *modulus* of the counter. A 3-bit counter is often referred to as a modulus-8 (or mod-8) counter since it has eight states. Similarly, a 4-bit counter is a mod-16 counter.



(a) Realization using JK FFs

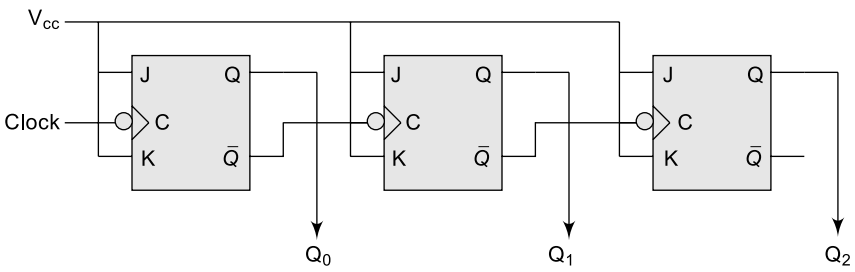
Clock pulses	States Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

(b) Truth table

Figure A.21 *3-bit asynchronous up-counter*

Asynchronous Counters Asynchronous counter can be realized by using by the use of JK flip-flops, as shown in Fig. A.21. The flip-flops are connected in series. The Q-output of each FF is connected to the clock input pf the next FF. Thus, the output of each FF drives the next FF. In this respect, counter is called *ripple* counter or *serial* counter. All the J and K inputs are tied to V_{cc}. This means that each FF will change state (toggle) with a negative transition at its clock input.

The counter shown in Fig. A.21 is a ripple up-counter. Similarly, a down-counter can be realized, as shown in Fig. A.22. The clock is still used at the clock input of the first FF, but the complement of Q_0 is used to drive second FF in the series. Similarly, the complement of Q_1 is used to drive next FF.



(a) Realization using JK FFs

Clock pulses	States Q_2	Q_1	Q_0
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8	1	1	1

(b) Truth table

Figure A.22 *3-bit asynchronous down-counter*

Synchronous Counters The ripple counter is the simplest to build, but each FF has a propagation delay. This problem can be eliminated in synchronous counters.

In synchronous counters, all FFs are triggered simultaneously by the same clock pulse so that all FFs change their state at the same time. For this, counters are called as parallel or carry look-ahead counters. Figure A.23 shows the schematic diagram of a 3-bit synchronous up counter.

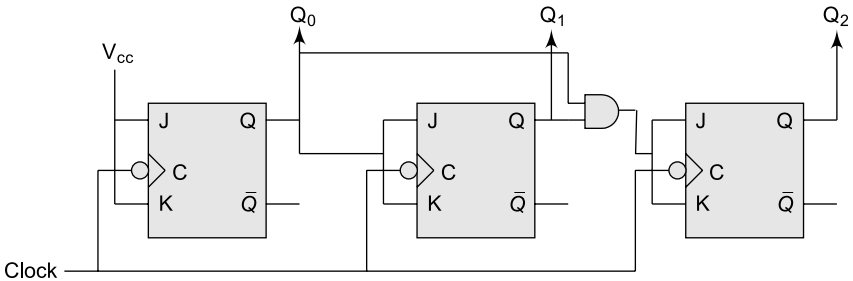


Figure A.23 *Realization of 3-bit synchronous up-counter*

A.3 MOS IMPLEMENTATION OF LOGIC GATES

The MOS (Metal Oxide Semiconductor) logic uses *metal oxide semiconductor field effect transistors* (MOSFETs). Compared to the bipolar logic families, the MOS families are simpler, inexpensive to fabricate and require much less power. There are three logic families constructed using MOS transistors:

- PMOS: Using p-channel transistors
- NMOS: Using n-channel transistors
- CMOS: Using both p-channel and n-channel transistors

PMOS is the slowest, oldest type and nearly obsolete today. NMOS is widely used in memories and microprocessors. CMOS is suitable for individual logic circuit design and it consumes very less power.

NMOS Transistors An n-channel transistor is said to be of NMOS-type and it behaves as a closed (on) switch when its gate input is equivalent to positive voltage V_{cc} (i.e. $V_{in} = V_{cc}$), as shown in Fig. A.24(a) and it acts as an open (off) switch when its gate input is 0 (i.e. $V_{in} = 0$), as shown in Fig. A.24(b).

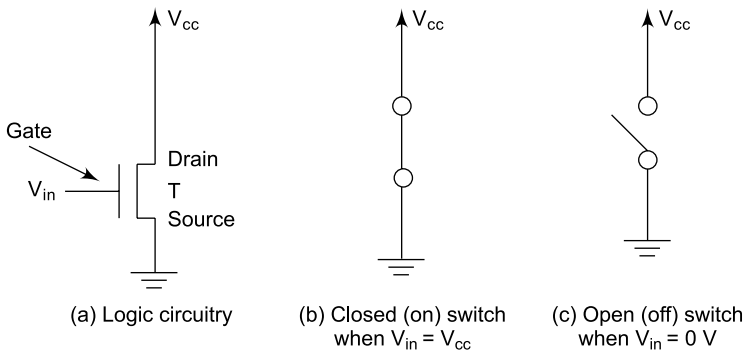


Figure A.24 *NMOS transistor*

A NOT gate can be realized using NMOS transistor, as shown in Fig. A.25. The transistor T acts as a switch.

When gate input voltage, $V_{in} = 0$, the transistor T acts as an open switch and output voltage, $V_{out} = V_{cc}$. When $V_{in} = V_{cc}$, the transistor T acts as a closed switch and output voltage V_{out} is near to 0. Thus, the circuit performs the function of a NOT gate.

PMOS Transistors A p-channel transistor is said to be of PMOS-type and it behaves as an open switch when the gate input, $V_{in} = V_{cc}$, as indicated by Fig. A.26(a). It acts as a closed switch when $V_{in} = 0$ V, as indicated by the Fig. A.26(b).

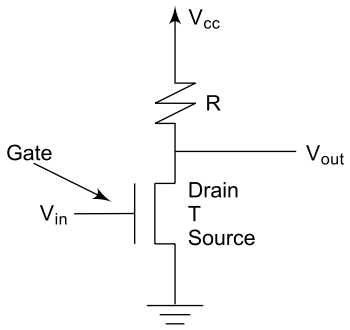


Figure A.25 *Realization of NOT gate using NMOS transistor*

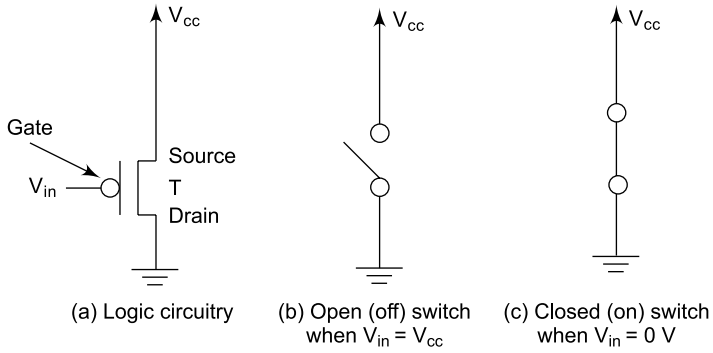
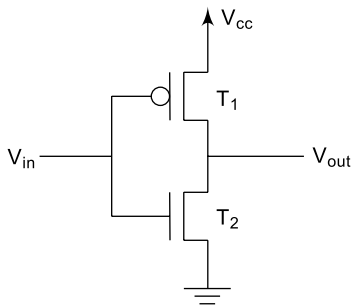


Figure A.26 *PMOS transistor*

Note that the PMOS transistor has a bubble on the gate input in its graphical symbol, which indicates that its functionality is complementary to that of an NMOS transistor. Also note that in PMOS transistors, the positions of source and drain terminals are revered in comparison to the NMOS transistors.

CMOS Transistors A CMOS (complementary metal oxide semiconductor) transistor uses both n-channel and p-channel transistors to take several advantages over PMOS and NMOS transistors. The CMOS transistor is faster and consumes less power than other two types. The basic idea of CMOS circuit is illustrated by the inverter circuit in Fig. A.27.



(a) Circuit

Input	*Transistors state*		*Output*
V_{in}	T_1	T_2	V_{out}
low (0)	on	off	high (1)
high (1)	off	on	low (0)

(b) Truth table

Figure A.27 *CMOS realization of a NOT gate*

When $V_{in} = V_{cc}$, transistor T_1 is turned off and transistor T_2 on. Thus the output voltage, V_{out}, drops to 0. When V_{in} is supplied with 0 V, transistor T_1 is turned on and transistor T_2 off. Thus the output voltage, V_{out}, gets up to V_{cc}. Hence, the logic values at points indicated by input V_{in} and output V_{out} are complements to each other and the circuit acts as a NOT gate.

In same way, other logic gates can be realized using CMOS transistors.

A.4 PROGRAMMABLE LOGIC DEVICES

In recent years, programmable logic devices (PLDs) have all but replaced special-purpose logic devices such as AND gates, flip-flops, counters, multiplexers, etc. PLDs are ICs (integrated chips) that can be programmed, and often re-programmed, to implement different logic functions.

The main reason for using programmable logic is to reduce total costs. This is due to a number of reasons. One important advantage is that design with PLDs is faster and this reduces the time required to bring a product to market. Programmable devices also reduce the risks associated with product development since they allow last-minute changes, often without having to redesign circuit boards. Since PLDs often replace several other special-purpose devices the design usually has fewer components and this reduces assembly, test and repair costs. Using PLDs also means fewer parts needs to be stocked and this reduces inventory costs. Since more of the logic is integrated into each chip the number of interconnections is decreased and this increases the reliability of the product.

A PLD is an IC that contains a large number of gates, FFs, and registers that are interconnected on the chip. Many of the connections, however, are fusible links that can be broken. The IC is said to be programmable because the specific function of the IC for a given application is determined by the selective breaking of some of the interconnections while leaving others intact. The "fuse blowing" process can be done either by the manufacturer in accordance with the customer's instructions, or by the customer himself. This process is called *programming* because it produces the desired circuit pattern interconnecting the gates, FFs, registers, and so on.

Of course, there are some disadvantages to using programmable logic. Design with PLDs requires additional development software and hardware which is often very expensive. Design staff often need to be trained to use new design tools. In addition, parts must be programmed before they can be assembled into a final product.

In spite of these disadvantages programmable logic usually makes economic sense except for very simple (e.g. bus buffers, latches, some decoders), very complex (e.g. CPU), or very high-speed circuits (e.g. DRAM controller). Even for one-off designs, it is often easier to use PLDs if the development tools are available.

The various PLDs used are PALs (programmable array logic), FPLAs (field programmable logic arrays) and PROMs (programmable read only memories).

A.4.1 Programmable Array Logic (PAL)

Programmable array logic (PAL) (a registered trade mark of Monolithic Memories) is a particular family of PLA devices that is widely used and available from a number of manufacturers. A PAL is a programmable logic device in which each output is computed as a two-level "sum of products" (an OR of ANDs).

The PAL circuits consist of a set of AND gates whose inputs can be programmed and whose outputs are connected to an OR gate, i.e. the inputs to the OR gate are hard wired. Some manufacturers also allow output inversion to be programmed. Thus, like AND-OR and AND-OR-INVERT logic, they implement a sum of products logic function. Figure A.28 shows a small example of the basic structure. The fuse symbols represent fusible links that can be burned open using equipment similar to a PROM (programmable read only memory) programmer. Note that every input variable and its complement can be left either connected or disconnected from every AND gate. We then say that the AND gates are programmed. Figure A.29 shows how the circuit is programmed to implement F = $\overline{A}$ BC + A $\overline{B}$ C. Note that all input variables and their complements are left connected to the unused AND gate, whose output is, therefore, $A\ \overline{A}$ B $\overline{B}$ C $\overline{C}$ = 0. The 0 has no effect on the output of the OR gate. On the other hand, if all inputs to the unused AND gate were burned open, the output of the AND gate would 'float' HIGH (logic 1), and the output of the OR gate in that case would remain permanently l. The actual PAL circuits have several groups of AND gates, each group providing inputs to separate OR gates.

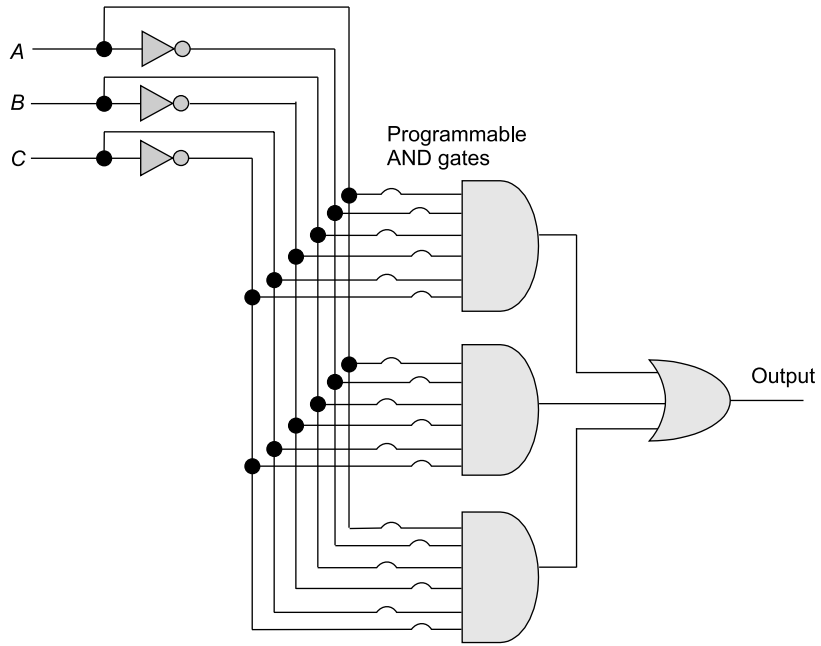


Figure A.28 *Basic structure of a PAL circuit*

Figure A.30 (a) shows a conventional means for abbreviating PLA connection diagrams. Note that the AND gate is drawn with a single input line, whereas in reality, it has three inputs. An X (cross) sign denotes a connection through an intact fusible link and a dot sign represents a permanent connection. The absence of any symbol represents an open or no connection by virtue of a burned-open link. In the example shown, input A is connected to the gate through a fusible link, input C is permanently connected, and input B is disconnected. Therefore, the output of the gate is AC.

Figure A.30 (b) shows an example of how the PAL structure is represented using the abbreviated connections. In this example, the circuit is un-programmed because all the fusible links are intact. Note that the 3-input OR gates are also drawn with a single input line.

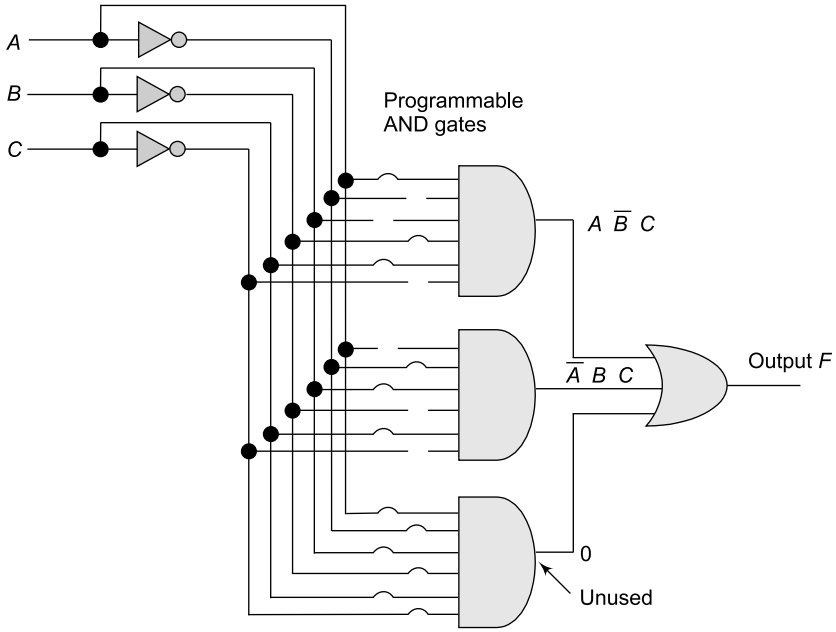


Figure A.29 *The circuit of Fig. A.28 programmed to implement* $F = \bar{A}BC + A\bar{B}C$

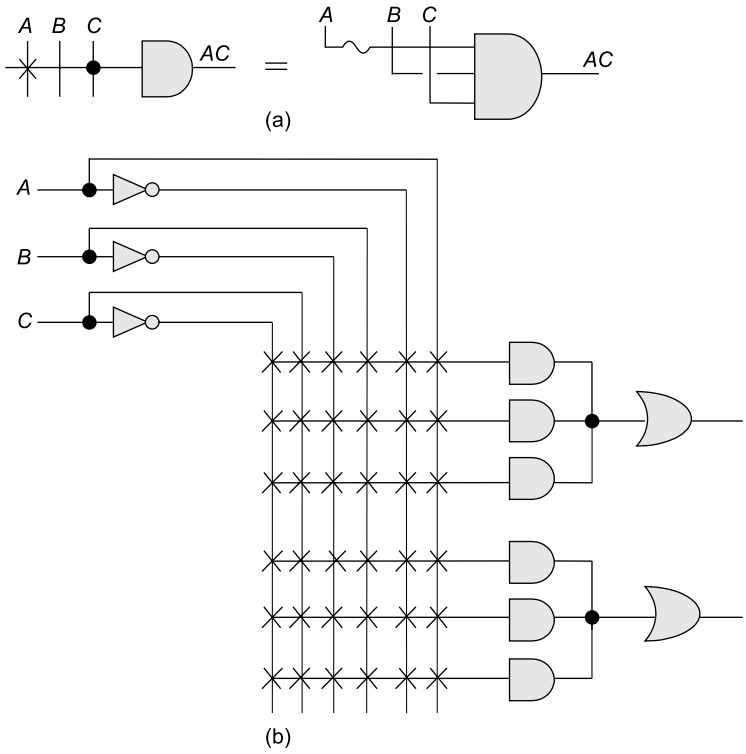


Figure A.30 *Simplified method for showing connections in PLA circuit*

The advantages of the PAL architecture include low and fixed (two gate) propagation delays (typically down to 5 ns), and simple, low-cost (free), design tools. However, the PAL architecture limits the design to simple state machines and simple combinational circuits.

A.4.2 Field Programmable Logic Array (FPLA)

The FPLA represents another type of programmable logic but with a slightly different architecture. The FPLA combines the characteristics of the PROM and the PAL by providing both a programmable OR array and a programmable AND array. This feature makes it the most versatile of the three PLDs. However, it has some disadvantages. Because has two sets of fuses, it is more difficult to manufacture, program and test it than a PROM or PAL. Figure A.31 demonstrates the FPLA structure, with every fusible link intact.

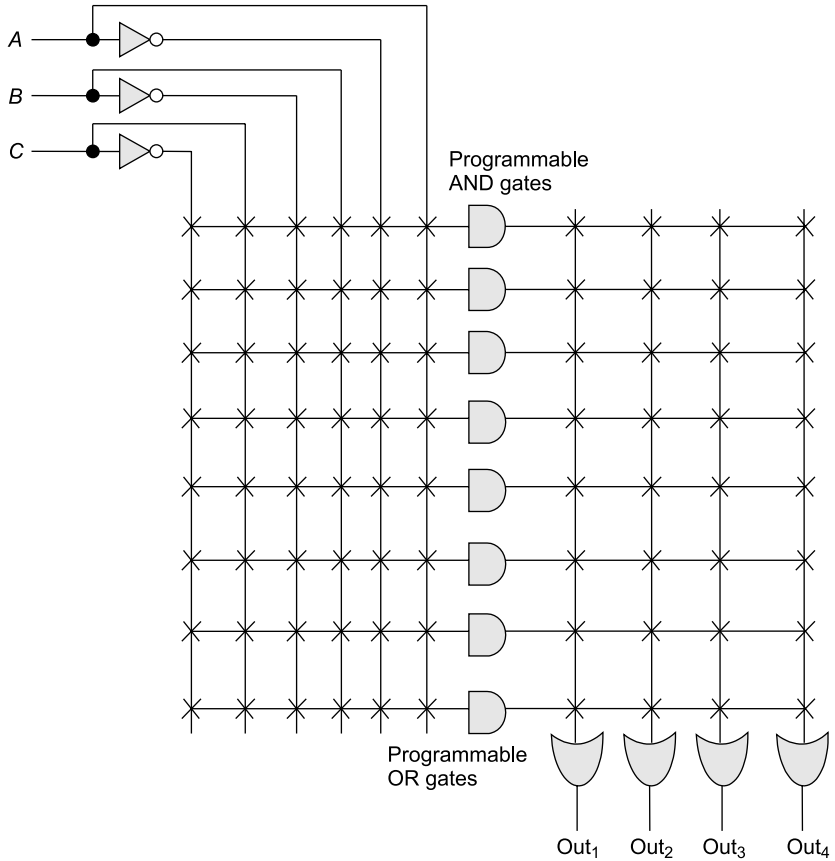


Figure A.31 *Structure of an un-programmed FPLA circuit*

A.4.3 Programmable ROM (PROM)

A programmable ROM can be viewed as a type of programmable logic array and thus used for that purpose. The address inputs to the PROM serve as logic variable inputs and the data output as the node where the output of a logic function is realized. For example, stating that a 1 is stored at address

1001 is the same as stating that the logic function being implemented equals 1 when the input combination is $A\,\overline{B}\,\overline{C}\,D$. In both the cases, the output will be a 1 when the input is 1001. When we regard a PROM as a PLA, we realize that the AND gates are not programmed. In effect, an AND gate is already in place for every possible combination of the inputs corresponding to every possible address of the PROM. Therefore, to program a PROM as a PLA, we must have a truth table that specifies the value of the function being implemented for every possible combination of the inputs. For each combination where F = 1, we leave the output of the corresponding AND gate connected to the output OR gate. For each combination where F = 0, we burn open the connection to the OR gate. We see that a PROM is a PLA with fixed AND gates and a programmable OR gate. An M × N PROM can be regarded as a PLA having N programmable OR gates, capable of implementing N different logic functions of M variables. A PROM is ideally suited for implementing a logic function directly from a truth table.

An example of an actual PROM that is often used as a PLD is AM27S13, which is a 512 × 4 PROM manufactured using high speed Schottky TTL technology. Since $512 = 2^9$, this PROM has nine address inputs and four data outputs. Thus, the AM27S13 can be programmed to generate four outputs each of which can be any logic function of the nine different inputs.

Example Show how an 8 × 1 PROM can be programmed to implement the logic function whose truth table is shown in Figure A.32.

Solution Figure A.32 shows the programmed PROM in the simplified connection format of a PLA. A logic 1 or a 0 is stored at every address combination corresponding to a combination of input variables for which the function equals a 1 or a 0.

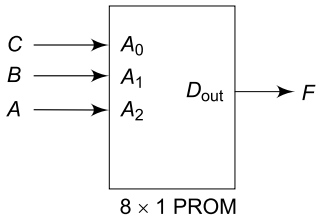
The PROM can generate any possible logic function of the input variables because it generates every possible AND product term. In general, any application that requires every input combination to be available is a good candidate for a PROM. However, PROMs become impractical when a large number of input variables have to be accommodated, because the number of fuses doubles for each added input variable.

A.4.4 Other PLD Features

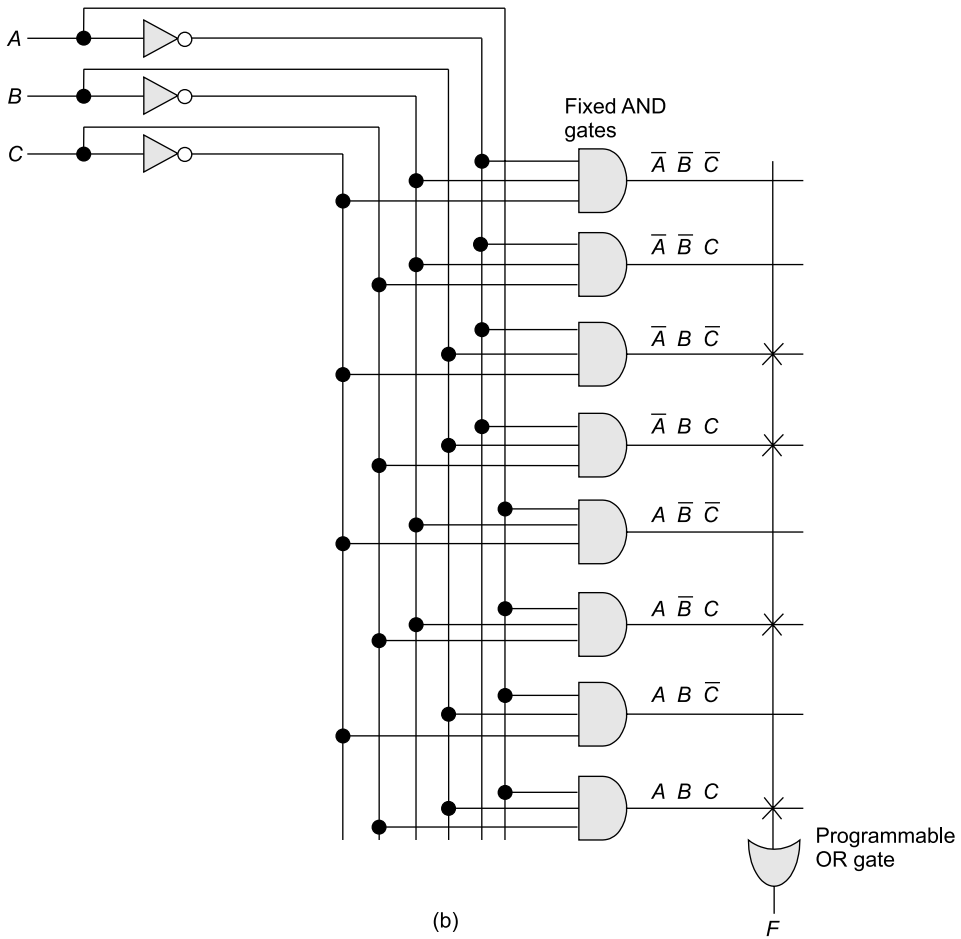
Many PLDs include one or more of the following as part of their architecture: FFs, latches, input registers, and output registers. Very often, the operating characteristics of these devices are programmable, as are the connections to other devices on the chip. This gives the logic designer a great deal of flexibility in designing counters and other sequential logic circuits. This type of PLD is sometimes called a *programmable logic sequencer.*

Programming When PLDs were first introduced, the logic designer would develop a *fuse map* that showed which fuses to blow. The manufacturer would then program the device according to the fuse map, test it, and return it the designer. In recent years, the availability of relatively inexpensive programming equipment has made it convenient for users to program their own PLDs. There are universal programmers in the market that can program the most common PROMs, PALs and FPLAs. The device to be programmed is plugged into a socket on the programmer and the programmer programs and tests the device according to data that have been supplied by the user.

Inputs			Output
A_2	A_1	A_0	D_{out}
A	*B*	*C*	*F*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



(a)



(b)

Figure A.32 *Programming a PROM to implement a truth table*

The programming and test data are typically developed by using the commonly available software that will run on standard PCs. Using this software, the user enters the data into the computer describing the logic functions to be programmed into the PLD, as well as information on how the device is to be tested. The software then generates a fuse map and the test data in a form that can be sent over a

cable to the PLD programmer's memory. Once the programmer has the data, it can proceed to program and test the device. When finished, the programmer will indicate whether the device has passed or failed the test procedure. If it passes, it can be removed from the programmer's socket and placed in the prototype circuit for further testing.

Erasable PLDs The PLDs we have been talking about are programmed by blowing fuses. Once a fuse is blown, it cannot be reconnected. Thus, if you make a mistake in programming or if you want to change the design, the device will no longer be useful. This drawback has been addressed by several manufacturers who have developed PLDs that can be erased and programmed over and over. These are called *erasable programmable logic devices* (EPLDs). These devices are programmed and erased much like EPROMs and EEPROMs.

A.5 INTRODUCTION TO ASSEMBLY-LANGUAGE PROGRAMMING

A set of instructions written for a computer to perform a task is called a *program* and a group of programs is called *software*. The physical components of a computer system are called *hardware*. It is possible to design parts of the hardware without a knowledge of its software capabilities. It is also possible to be familiar with various aspects of computer software without being concerned with details of how the computer hardware operates. However, those concerned with computer architecture should have a knowledge of both hardware and software because the two branches influence each other.

Writing a program for a computer consists of specifying, directly or indirectly, a sequence of machine instructions. Machine instructions are represented by patterns of 0s and 1s. Such patterns are difficult for people to work with and understand. It is preferable to write programs with the more familiar symbols of the alphanumeric character set. Such a symbolic program is referred to as an *assembly-language program*. As a consequence, there is a need for translating this assembly-language program into equivalent binary program (machine language) recognized by the hardware. This translation is done by a special program called an *assembler.*

To design programs using the instruction sets, the symbolic user-oriented format called assembly language can be used. This section discusses the basic features of assembly language and their relationship both to the computer organization and to the machine-language programs that are actually executed by the computer. Most computer programming is now done using high-languages such as C, which, like assembly language, must be translated (compiled) into machine language prior to execution.

A.5.1 Assembly Language

A complete set of symbolic operation names known as *mnemonics* and rules for their use constitute programming language, generally referred to as an *assembly language.* The set of rules for using the mnemonics in the specification of complete instructions and programs called the *syntax* of the language.

Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an *assembler.* The assembler program is one of a collection of utility programs that are a part of the system software. The assembler, like any other program, is

stored as a sequence of machine instructions in the memory of the computer. A user (application) program is usually entered into the computer through a keyboard and stored either in the memory or on a magnetic disk. At this point, the user program is simply a set of lines of alphanumeric characters. When the assembler program is executed, it reads the user program, analyzes it, and then generates the desired machine language program. The machine language program contains patterns of 0s and 1s specifying instructions that will be executed by the computer. The user program in its original alphanumeric text format is called a *source program,* while the assembled machine language program is called an *object program.*

The assembly language for a given computer may or may not be case sensitive, that is, it may or may not distinguish between capital and lower case letters. We will use capital letters to denote all names and labels in our examples in order to improve the readability of the text. For example, we will write a Move (transfer) instruction as:

```
MOV R0, R1
```

The mnemonic MOV represents the binary pattern, or *op-code,* for the operation performed by the instruction. The assembler translates this mnemonic into the binary op-code that the computer understands.

The op-code mnemonic is followed by at least one blank space character. Then the information that specifies the operands is given. In our example, the source operand is in register R1, which is transferred to destination register R0.

Since there are several possible addressing modes for specifying operand locations, the assembly language must indicate which mode is being used. For example, a numerical value or a name used by itself, such as registers R0, R1 in the preceding instruction, may be used to denote the register mode. The sharp (#) sign usually denotes an immediate operand. Thus, the instruction

```
ADD R1, #5
```

adds the number 5 to the contents of register R1 and puts the result back into register R1. The sharp sign is not the only way to denote the immediate addressing mode. In some assembly languages, the intended addressing mode is indicated in the op-code mnemonic. In this case, a given instruction has different op-code mnemonics for different addressing modes. For example, the previous Add instruction may be written as

```
ADI R1, 5
```

The suffix I in the mnemonic ADI states that the source operand is given in the immediate addressing mode.

The indirect addressing is usually specified by putting parentheses around the name or symbol denoting the pointer to the operand. For example, if content of a location whose address is given in X is to be loaded into register R1, the desired action can be specified as

```
MOV R1, (X)
```

A.5.2 Assembler Directives

In addition to providing a mechanism for representing instructions in a program, the assembly language allows the programmer to specify other information needed to translate the source program into the object program. The assembly language allows symbolic names to be assigned to user-defined

constants and variables, such as the immediate operand appearing in earlier examples. For example, many assembly languages use the statement

```
SUB   EQU   2001
```

to indicate that the symbol SUB is to be equivalent (EQU) to the decimal number 2001. This statement does not denote an instruction that will be executed when the object program is run; in fact, it will not even appear in the object program. It simply informs the assembler that the name SUB should be replaced by the value 2001 wherever it appears in the program. Such type of non-executable assembly language instruction, called *assembler directive* (or *pseudo-instruction*), is used by the assembler while it translates a source program into an object program.

Table A.2 lists a representative set of the directives found in most assembly languages. Other assembly language programs may recognize many more directives. The EQU (Equivalent) directive tells the assembler to equate two different names for the same thing, as we have already discussed. The ORG (origin) pseudo-instruction informs the assembler that the instruction or operand in the following line is to be placed in a memory location specified by the number next to ORG. For example, consider the following program segment

```
ORG        100
LDA        SUB
```

Here, the first line has the pseudo-instruction ORG to define the origin of the program at memory location 100. In other words, the LDA instruction is to be assigned to memory location 100. It is also possible to use ORG more than once in a program to specify more than one segment of memory.

Table A.2 *List of representative assembly language directives*

Type	*Op-code*	*Description*
Symbol definition	EQU	Equate symbolic name (in label position) to operand value.
Memory assignment	ORG	Origin: use operand value as starting address for subsequent instructions.
	DS	Define storage: reserve the specified number of consecutive locations (bytes) in memory.
	DC	Define constant: store the operand values as constants.
Miscellaneous	END	End of program(s) to be assembled.
	RET	Return the control from the current sub-rou tine to calling (main) routine.
	IF	Start of conditional block of instructions to be assembled only if a specified condition is met.
	ENDIF	End of conditional block.

Sometimes it is useful to reserve a block of memory for future use, for example, as a buffer storage area for I/O data, without specifying its contents. The DS (define storage) instruction is provided for this purpose. Thus the directive

```
L2   DS   200
```

states that a block of 200 memory bytes should be reserved, beginning at the current location L2. If it is desired to actually define data to be placed in a program, the DC (define constant) directive is used. The END symbol is placed at the end of the program to inform the assembler that the program is terminated. The RET assembler directive identifies the point at which execution of the program should be terminated. It causes the assembler to insert an appropriate machine instruction that returns control to the calling program or to the operating system of the computer.

A.5.3 Rule of Assembly Language

A programming language is defined by a set of rules. Users must conform with all format rules of the language if they want their programs to be translated correctly. Almost every commercial computer has its own particular assembly language. In other words, the assembly language differs in detail and complexity from one computer to another. The rules for writing assembly-language programs are documented and published in manuals which are usually available from the computer manufacturer.

The basic unit of an assembly-language program is a line of code. The specific language is defined by a set of rules that specify the symbols that can be used and how they may be combined to form a line of code. We will now formulate the rules of an assembly language for writing symbolic programs for the basic computer.

Each line of an assembly language program to be written in the form:

Label Operation Operand(s) Comment

These four *fields* are separated by an appropriate delimiter, typically one or more blank characters. The *Label* is an optional field associated with the memory address where the machine language instruction produced from the statement will be loaded. Labels may also be associated with addresses of data items. The *Operation* field contains the op-code mnemonic (symbolic word such as MOV, ADD) of the desired instruction or assembler directive (non-executable assembly-language instruction such as the EQU). The *Operand* field contains addressing information for accessing one or more operands, depending on the type of instruction. The *Comment* field is ignored by the assembler program and each comment is generally preceded by a semi-colon (;). It is used for documentation purposes to make the program easier to understand.

An Example Let us consider an example to perform a subtraction of two numbers. The subtraction is performed by adding the minuend to the 2's complement of the subtrahend. The negative numbers are represented in signed-2's complement forms. Also assume that the subtraction is: 85 – (–20) = 105. Since the subtrahend is negative (–20), we take 2's complement of it (1's complement and increment of the AC).

Table A.3 gives an assembly language program for this task. The first line has the pseudo-instruction ORG to define the origin of the program at memory location 200. The next six lines define machine instructions, and the last four have pseudo-instructions. Three symbolic addresses have been used as Labels and in second field as an address of a memory-reference instruction. Three of the pseudo-instructions specify operands, and the last one signifies the END of the program.

Table A.3 *An assembly-language program to subtract two numbers*

	ORG 200	; Origin of program is location 200
	LDA SUB	; Load subtrahend to AC
	CMA	; Complement AC
	INC	; Increment AC
	ADD MIN	; add minuend to AC
	STA RES	; Store difference (result)
	H LT	; Halt computer
MIN	DC 85	; Minuend
SUB	DC -20	; Subtrahend
RES	DS 0	; Result stored here
	END	; End of symbolic program

To translate this assembly language program into an equivalent binary code, we assume six instructions of the basic computer in Table A.4, to provide an easy reference.

Table A.4 *A simple computer instructions (M refers to memory word)*

Symbol	*Hexadecimal code*	*Description*
ADD	1	Add M to AC
LDA	2	Load AC from M
STA	3	Store AC in M
CMA	7200	Complement AC
INC	7020	Increment AC
HIT	7001	Halt computer

The translation of the assembly program into machine codes is done by a special program called an *assembler.* The translation of the assembly program of Table A.3 into an equivalent binary code may be done by scanning the program and replacing the symbols by their machine code binary equivalent. Starting from the first line, we encounter an ORG pseudo-instruction. This states to start the binary program from the location 200. The second line has two symbols. It is a memory-reference instruction to be placed in location 200. The mnemonic of the operation is LDA. Checking instruction set of the computer, suppose we find that the first hexadecimal digit of the instruction should be 2 (which is equivalent numeric op-code of LDA). The binary value of the address part must be obtained from the address symbol SUB. We scan the Label field and find this symbol in 9^{th} line. To determine its hexadecimal value we note that the 2^{nd} line contains an instruction for location 200 and every other line specifies a machine instruction or an operand for sequential memory locations. Counting lines, we find that Label SUB in 9^{th} line corresponds to memory location 207. Therefore, the hexadecimal address of the instruction LDA must be 207. When the two parts of the instruction are assembled, we obtain the hexadecimal code as 2207. The other lines representing machine instructions are translated in a similar manner and their hexadecimal code is listed in Table A.5.

Two lines in the assembly program specify decimal operands with the pseudo-instruction DC. A third specifies a zero means it reserves one 16-bit memory space to store the result. Decimal 85 is converted to binary and placed in location 206 in its hexadecimal equivalent. Decimal -20 is a negative number and must be converted into binary in signed-2's complement form. The hexadecimal equivalent of the binary number is placed in location 207. The END directive indicates the end of the assembly program.

A.5.4 Assembler

A source program written in an assembly language must be assembled into a machine language object program before it can be executed. This is done by the assembler program, which replaces all symbols denoting operations and addressing modes with the binary codes used in machine instructions, and replaces all names and labels with their actual values.

Table A.5 *Listing of translated program of Table A.4*

Hexadecimal code			
Location	*Content*		*Assembly program*
			ORG 200
200	2207		LDA SUB
201	7200		CMA
202	7020		INC
203	1206		ADD MIN
204	3208		STA RES
205	7001		HLT
206	0055	MIN	DC 85
207	FFEC	SUB	DC -20
208	0000	RES	DS 0
			END

The assembler assigns addresses to instructions and data blocks, starting at the address given in the ORG assembler directives. It also inserts constants that may be given in DC directives and reserves memory space as requested by DS directives.

The main part of the assembly process is determining the values that replace the names. In some cases, where the value of a name is specified by an EQU directive, this is a simple task. In other cases, where a name is defined in the Label field of a given instruction, the value represented by the name is determined by the location of this instruction in the assembled object program. Hence, the assembler must keep track of addresses as it generates the machine code for successive instructions. For example, the names MIN and SUB will be assigned the values 206 and 207, respectively.

In some cases, the assembler does not directly replace a name representing an address with the actual value of this address. For example, in a branch instruction, the name that specifies the location to which a branch is to be made (the branch target) is not replaced by the actual address. A branch instruction is usually implemented in machine code by specifying the branch target using the relative addressing mode. The assembler computes the *branch offset,* which is the distance to the target and puts it into the machine instruction.

As the assembler scans through a source program, it keeps track of all names and the numerical values that correspond to them in a *symbol table.* Thus, when a name appears a second time, it is replaced with its value from the table. A problem arises when a name appears as an operand before it is given a value. For example, this happens if a forward branch is required. The assembler will not be able to determine the branch target, because the name referred to has not yet been recorded in the symbol table. A simple solution to this problem is to have the assembler scan through the source program twice. During the first pass, it creates a complete symbol table. At the end of this pass, all

names will have been assigned numerical values. The assembler then goes through the source program a second time and substitutes values for all names from the symbol table. Such an assembler is called a *two-pass assembler.*

The assembler stores the object program on a magnetic disk. The object program must be loaded into the memory of the computer before it is executed. For this to happen, another utility program called a *loader* must already be in the memory. Executing the loader performs a sequence of input operations needed to transfer the machine language program from the disk into a specified place in the memory. The loader must know the length of the program and the address in the memory where it will be stored. The assembler usually places this information in a header preceding the object code. Having 1oaded the object code, the loader starts execution of the object program by branching to the first instruction to be executed. Recall that the address of this instruction has been included in the assembly language program as the operand of the END assembler directive. The assembler includes this address in the header that precedes the object code on the disk.

When the object program begins executing, it proceeds to completion unless there are logical errors in the program. The user must be able to find errors easily. The assembler can detect and report syntax errors. To help the user find other programming errors, the system software usually includes a *debugger* program. This program enables the user to stop execution of the object program at some points of interest and to examine the contents of various processor registers and memory locations.

2007
Computer Organization
(CS-303)

Semester: 3rd

Time Allotted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) When signed numbers are used in binary arithmetic, then which one of the following notations would have unique representation for zero?

(a) magnitude (b) 1's complement (c) 2's complement (d) none.

Answer

(c) 2's complement

(ii) How many address bits are required for a 1024×8 memory?

(a) 1024 (b) 5 (c) 10 (d) none.

Answer

(c) 10

(iii) The principle of locality justifies the use of

(a) interrupt (b) polling (c) DMA (d) cache memory.

Answer

(d) cache memory

(iv) A major advantage of direct mapping of a cache is its simplicity. The main disadvantage of this organization that

(a) it does not allow simultaneous access to the intended data and its tag

(b) it is more expensive than other types of cache organizations

(c) the cache hit ratio is degraded if two or more blocks used alternatively map onto the same block frame in the cache

(d) its access time is greater than that of other cache organizations.

Answer

(c) the cache hit ratio is degraded if two or more blocks used alternatively map onto the same block frame in the cache

(v) A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (op-code) and address part (allowing for only one address). Each instruction is stored in one word of memory. Bits are needed for op-code

(a) 6 (b) 7 (c) 8 (d) 9

Answer

(c) 8

(vi) Maximum n bit 2's complement number is

(a) 2^n (b) $2^n - 1$ (c) $2^{n-1} - 1$ (d) cannot be said.

Answer

(c) $2^{n-1} - 1$

(vii) State true or false:

Adding $0110\ 1101_2$ to $1010\ 0010_2$ in 8-bit 2's complement binary will cause an overflow:

(a) true (b) false.

Answer

(a) true

(viii) Micro instructions are kept in

(a) main memory (b) control memory (c) cache memory (d) none.

Answer

(b) control memory

(ix) Instruction cycle is

(a) fetch-decode-execution (b) decode-fetch-execute

(c) fetch-execution-decode (d) none.

Answer

(a) fetch-decode-execution

(x) The basic principle of the von Neumann computer is

(a) storing program and data in separate memory

(b) using pipe line concept

(c) storing both program and data in the same memory

(d) using a large number of registers.

Answer

(c) storing both program and data in the same memory

Group-B
(Short-Answer Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. (a) Explain the difference between three-address, two-address, one-address instructions & zero-address instruction with suitable examples.

(b) Give an example and explain Base-Index Addressing. 3 + 2

Answer:

(a) The size of programs consisting of all three-address instructions is small, whereas that of programs using zero-address instructions is large. Three- and two-address instructions are generally used in general-register organized processors, one-address instructions used in single accumulator based processors and zero-address instructions are used in stack based CPU organizations.

Suppose we have to evaluate the arithmetic statement

X = (A + B) * C

using zero, one, two or three address instructions. For this, LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations addition and multiplication respectively. Assume that the respective operands are in memory addresses A, B and C and the result must be stored in the memory at address X.

Using three-address instructions, the program code in assembly language is as:

```
ADD R1, A, B     ; R1← M[A] + M[B]
MULT X, C, R1    ; X ← M[C] + R1
```

Using two-address instructions, the program code in assembly language is as:

```
LOAD R1, A       ; R1← M[A]
ADD R1, B        ; R1 ← R1 + M[B]
LOAD R2, C       ; R2← M[C]
MULT R1, R2      ; R1 ← R1 * R2
STORE X, R1      ; X ← R1
```

Using one-address instructions, the program code in assembly language is as:

```
LOAD A           ; AC ← M[A]
ADD B            ; AC ← AC + M[B]
STORE T          ; T ← AC
LOAD C           ; AC ← M[C]
MULT T           ; AC ← AC * M[T]
STORE X          ; X ← AC
```

Using zero-address instructions, the program code in assembly language is as:

```
PUSH A           ; TOS ← A [TOS means top of the stack]
PUSH B           ; TOS ← B
ADD              ; TOS ← (A + B)
PUSH C           ; TOS ← C
MULT             ; TOS ← ((A + B) * C)
POP X            ; X ← TOS
```

(b) In this mode the content of the base register (BR) is added to the address part of the instruction to obtain the effective address. This mode is similar to the indexed addressing mode, but exception is in the way they are used. A base register holds the starting address of a memory array of operands and the address part of the instruction gives a displacement or offset relative to this starting address. This mode is used for relocation of the programs in the memory.

For example, an operand array starts at memory address 1000 and thus the base register BR contains the value 1000. Now consider load instruction

LDA 0002

The effective address of the operand is calculated as:

Effective address = 0002 + content of BR
= 1002.

3. (a) Explain the difference between full associative and direct mapped cache mapping approaches.
   (b) What are "write through" and "write back" policies in cache? 3 + 2

Answer

(a) The fully associative cache memory uses the fastest and most flexible mapping method, in which both address and data of the memory word are stored. This memory is expensive because of additional storage of addresses with data in the cache memory.

In the direct cache mapping, instead of storing total address information with data in cache, only part of address bits is stored along with data. Suppose the cache memory can hold 2^m words and main memory can hold 2^n words. The n-bit address generated by the CPU is divided into two fields: lower-order m bits for the index field and the remaining higher-order (n-m) bits for the tag field. The direct mapping cache organization uses the m-bit index to access the cache and higher-order (n-m) bits of tag are stored along side the data in cache. This is the simplest type of cache mapping, since only tag field is required to match. That's why it is one of the fastest caches. Also, it is less expensive cache relative to the associative cache.

(b) There are two policies in writing into cache memory: (i) write-through (ii) write-back.

Write-Through Policy: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus, the main memory always contains the same data as the cache.

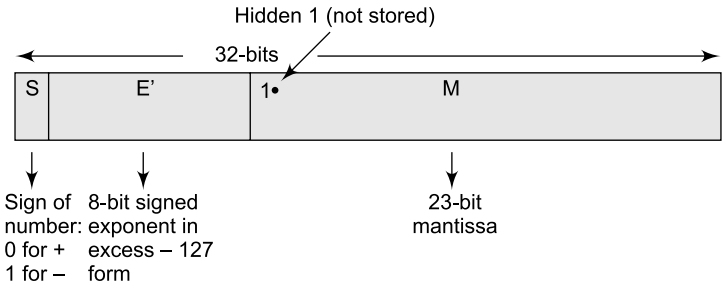
Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called modified or dirty bit. When the word is replaced from cache, it is written into main memory if its flag bit is set.

4. (a) Briefly explain the IEEE 754 standard format for floating point representation.
   (b) How NaN (Not a Number) and Infinity are represented in this standard. 3 + 2

Answer

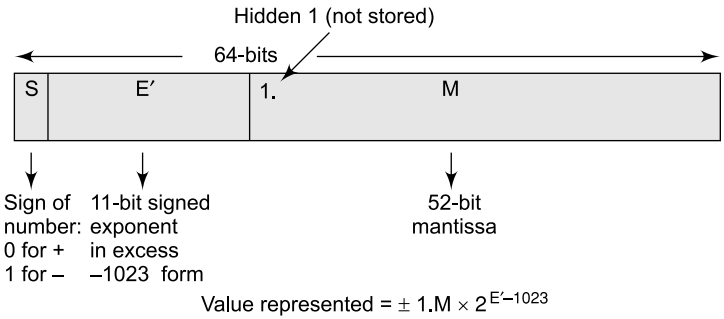
(a) The IEEE 754 has two similar formats as follows:

1. Single precision format: It is 32-bit format, in which 8-bit is for exponent, 23-bit for mantissa, 1-bit for sign of the number, as shown in the following figure.



Here, the implied base 2 and original signed exponent E are not stored in register. The value actually stored in the exponent field is an unsigned integer E′ called *biased exponent*, which is calculated by the relation E′ = E + 127. This is referred to as the *excess–127* format. Thus, E′ is in the range $0 \le E' \le 255$. The end values of this range, 0 and 255, are used to represent special values. Therefore, the range of E′ is $1 \le E' \le 254$, for normal values. This means that the actual exponent (E) is in the range $-126 \le E \le 127$.

2. Double precision format: This is 64-bit format in which 11-bit is for biased exponent E′, 52-bit for mantissa M and 1-bit for sign of the number, as shown in figure next. The representation is same as single precision format, except the size and thus other related parameters.



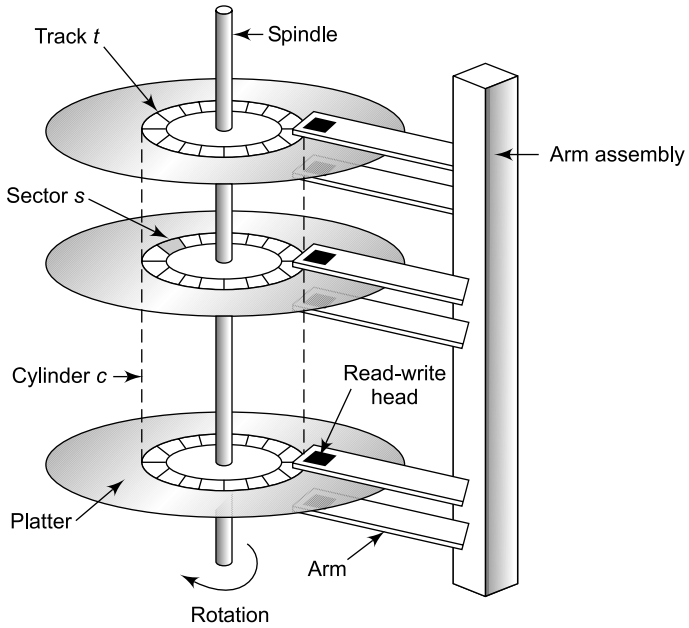
(b) Not a Number (NaN) is represented when E′ = 255 and M ≠ 0. NaN is a result of performing an invalid operation such as 0/0 and $\sqrt{-1}$.

Infinity is represented when E′ = 255 and M = 0. The infinity is the result of dividing a normal number by 0.

5. (a) Draw and explain the basic structure of a Hard-Disk and explain seek-time and latency time associated with it.
   (b) Compare restoring and non-restoring division algorithms. 3 + 2

Answer

(a) Disks that are permanently attached to the unit assembly and cannot be removed by the general user are called hard disks. A hard disk consists of some number of circular magnetic disks. The magnetic disk is made of either aluminum or plastic coated with a magnetic material so that information can be stored on it. The recording surface is divided into a number of concentric circles called *tracks*. The tracks are commonly divided into sections called *sectors*. To distinguish between two consecutive sectors, there is a small *inter-sector gap*. In most systems, the minimum quantity of information transfer is a sector. The information is accessed onto the tracks using movable read-write heads that move from the innermost to the outmost tracks and vice-versa. There is one read-write head per surface. During normal operation, disks are rotated continuously at a constant angular velocity. Same radius tracks on different surfaces of disks form a logical *cylinder*.



To access data, the read-write head must be placed on the proper track based on the given cylinder address. The time required to position the read-write head over the desired track is known as the `seek time`, t_s. After positioning the read-write head on the desired track, the disk controller has to wait until the desired sector is under the read-write head. This waiting time is known as `rotational latency`, t_l. The *access time* of the disk is the sum of t_s and t_l.

(b) Both algorithms are directly used for division of two unsigned numbers. The restoring algorithm requires more number of additions compared to non-restoring algorithm. Because in first algorithm, when the accumulator register becomes negative after the subtraction of divisor from accumulator content, the content of accumulator is restored to previous value by adding divisor to it. This restoration step is not required in non-restoring algorithm. However, the non-restoring algorithm may require a restoration step at the end of algorithm, if content of accumulator is negative.

6. (a) Explain the difference between Instruction pipeline and arithmetic Pipeline.
   (b) What are the different hazards in pipeline? 3 + 2

Answer

(a) 1. Instruction pipeline is used to process all instructions, whereas arithmetic pipeline is used to process arithmetic type instructions such as addition, subtraction, multiplication, etc.
2. In instruction pipeline, the execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions. An arithmetic pipeline divides an arithmetic operation, such as a multiply, into multiple arithmetic steps each of which is executed one-by-one in different arithmetic stages in the ALU.
3. All high-performance computers are now equipped with instruction pipeline. The number of arithmetic pipelines varies from processors to processors.

(b) Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. There are three types of pipeline hazards:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards:
They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Structural Hazards:
Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Data Hazards:
Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard.*

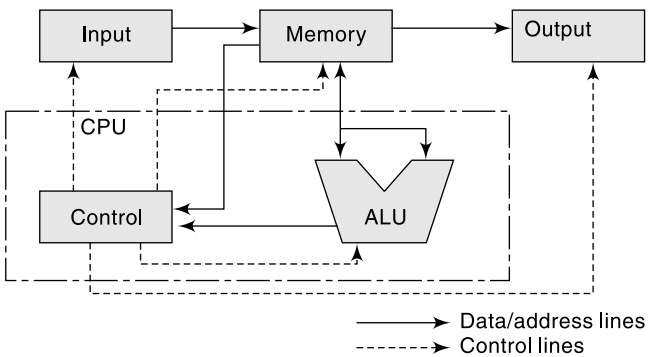
Group-C
(Long-Answer Questions)

Answer any three of the following questions. $3 \times 15 = 45$

7. (a) Describe the function of major components of a digital computer with neat sketch.
   (b) Explain the reading and writing operations of a basic static MOS cell.
   (c) How many 128×16 RAM chips are needed to construct a memory capacity of 4096 words (16 bit is one word)? How many lines of the address bus must be used to access a memory of 4096 words? For chip select, how many lines must be decoded? 5 + 6 + 4

Answer

(a)



Block diagram of a computer

The major units of a computer are described next:

(i) *Arithmetic and Logic Unit (ALU):* It is the main processing unit which performs arithmetic and other data processing tasks as specified by the control unit. The ALU and control unit are the main constituent parts of the Central Processing Unit (CPU). Another component of the CPU is register unit-collection of different registers, used to hold the data or instruction temporarily.

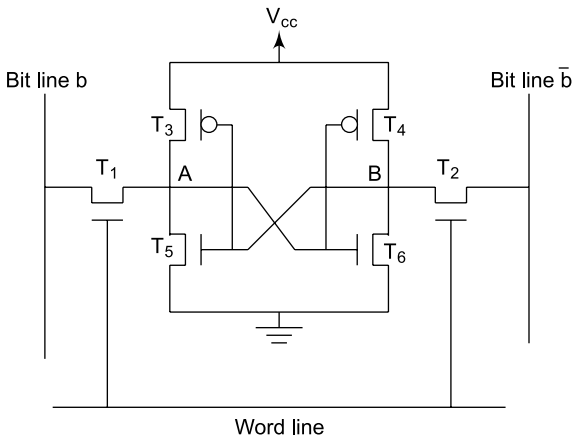
(ii) *Control Unit:* This is the unit that supervises the flow of information between various units. The control unit retrieves the instructions using registers one by one from the program, which is stored in the memory. The instructions are interpreted (or decoded) by the control unit itself and then the decoded instructions are sent to the ALU for processing.

(iii) *Memory:* The memory unit stores programs as well as data. Generally three types of memories are used: secondary, main and cache memories.

(iv) *Input Unit:* This unit transfers the information as provided by the users into memory. Examples include keyboard, mouse, scanner, etc.

(v) *Output Unit:* The output units receive the result of the computation and displayed to the monitor or the user gets the printed results by means of a printer.

(b) One SRAM cell using CMOS is shown in figure next. Four transistors (T_3, T_4, T_5 and T_6) are cross connected in such a way that they produce a stable state. In state 1, the voltage at point A is maintained high and voltage at point at B is low by keeping transistors T_3 and T_6 on (i.e. closed), while T_4 and T_5 off (i.e. open). Thus, for state 1, if T_1 and T_2 are turned on (closed), bit lines b and b′ will have high and low signals, respectively.



A CMOS SRAM cell

Read Operation: For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation: Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.

(c) The number of 128 × 16 RAM chips needed to construct a memory capacity of 4096 × 16 is

= (4096/128) * (16/16) = 32.

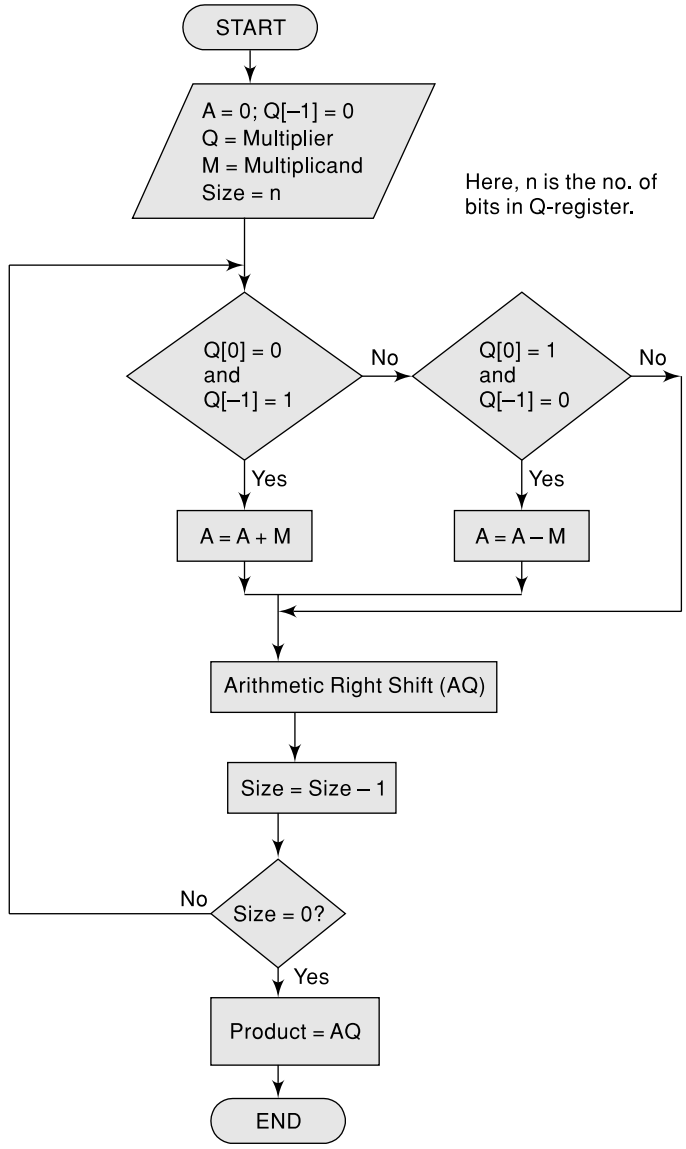
Since the large memory size is 4096 words. The number of lines in the address bus used to access the memory is 12.

Since the 32 rows (= 4096/128) of 128 × 16 RAMs are used to construct the large memory of size 4096 × 16. The number of lines decoded to select a RAM is 5. Therefore the address decoder size is 5-to-32.

8. (a) Give the Booth's algorithm for multiplication of signed 2's complement numbers in flow-chart and explain.
   (b) Multiply –5 by –3 using Booth's algorithm.
   (c) What is von Neumann architecture? What is von Neumann bottleneck?
   (d) What is virtual memory? 5 + 4 + 4 + 2

Answer

(a)



Booth's multiplication algorithm

The algorithm inspects two lower-order multiplier bits at time to take the next step of action. The algorithm is described by the flowchart shown above. A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contain the product. Ignore the right end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier.

(b) M = –5 = 1011 and Q = –3 = 1101.

	M	A	Q	Size
Initial Configuration	1011	0000	1101 0	4
Step-1				
As Q[0]=1and Q[–1]=0				
A=A – M	1011	0101	1101 0	-
And ARS(AQ)	1011	0010	1110 1	3
Step-2				
As Q[0]=0 and Q[–1]=1				
A = A + M	1011	1101	1110 1	-
ARS(AQ)	1011	1110	1111 0	2
Step-3				
As Q[0]=1 and Q[–1]=0				
A = A – M	1011	0011	1111 0	-
ARS(AQ)	1011	0001	1111 1	1
Step-4				
As Q[0]=1 and Q[–1]=1				
ARS(AQ)	1011	0000	1111 1	0

Since, the size register becomes 0, the algorithm is terminated and the product is = AQ = 0000 1111, which shows that the product is a positive number. The result is 15 in decimal.

(c) *Von Neumann architecture:*

The architecture uses a concept, known as stored-program concept, and has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

Each instruction contains only one memory address and has the format:

OPCODE ADDRESS

The 8-bit op-code specifies the operation to be performed by the CPU and 12-bit address specifies the operand's memory address. Thus, length of each instruction is 20-bit.

Von-Neumann bottleneck:
One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck.

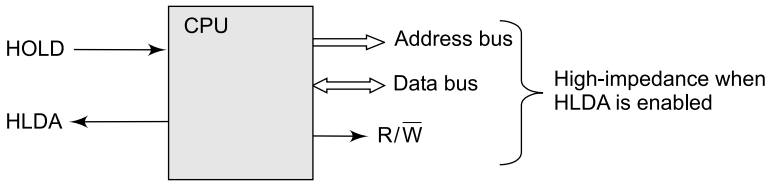
(d) Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, although which may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual (logical) address is mapped to a physical address in main memory. This mapping is done during run-time and is performed by a hardware device called *memory-management unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

9. (a) Explain the basic Direct Memory Access (DMA) operation for transfer of data bytes between memory and peripherals.
   (b) Give the main reason why DMA based I/O is better in some circumstances than interrupt driven I/O?
   (c) What is programmed I/O technique? Why is it not very useful?
   (d) According to the following information, determine size of the subfields (in bits) in the address for Direct Mapping and Set Associative Mapping cache schemes:
   - We have 256 MB main memory and 1 MB cache memory
   - The address space of the processor is 256 MB
   - The block size is 128 bytes
   - There are 8 blocks in a cache set. 5 + 3 + 3 + 4

Answer

(a) DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). The following figure shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.



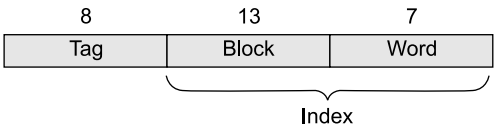
CPU bus signals for DMA transfer

(b) To transfer large blocks of data at high speed, DMA method is used. A special DMA controller is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. The data transmission cannot be stopped or slowed down until an entire block is transferred. This mode of DMA transfer is known as burst transfer.

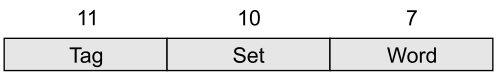
(c) This is the software method where CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU.

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. Generally the CPU is 5-7 times faster than an I/O device. Thus, the difference in data transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

(d) The address space of the processor is 256 MB. So, the processor generates an address of 28-bit.
The cache memory size = 1MB
Therefore, the size of index field of cache = 20-bit (1MB = 2^{20})
The tag-field uses 28 – 20 = 8 bits.
The number of blocks in cache = Size of cache/size of a block = $2^{20}/2^7$ = 8192.
Therefore the number of bits required to select each block = 13 (since 8192 = 2^{13})
The size of each block is 128 bytes.
So, the number of bits required to select a word (byte) in a block = 7.
Thus, the address format for direct mapped cache is as follows:



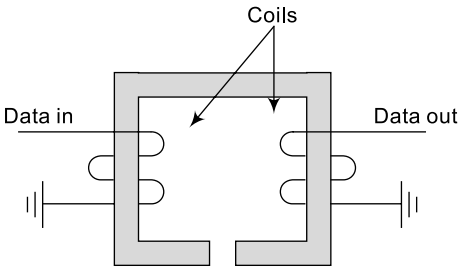
The number of blocks in a set is = 8
Number of bits required to select a block in a set is = 3 (because 8 = 2^3).
Number of sets in the set-associative cache is = 8192/8 = 1024.
To select each set, number of bits required is = 10 (because 1024 = 2^{10}).
Therefore, tag field requires (28 – (10 + 7)) = 11 bits.
Thus, the address format for set-associative cache is as follows:



10. Write short notes on any *three* of the following: $3 \times 5 = 15$
    (a) Magnetic recording
    (b) Adder-subtractor circuit
    (c) Addressing modes
    (d) Stack organization
    (e) Bus organization using tristate buffer.

Answer

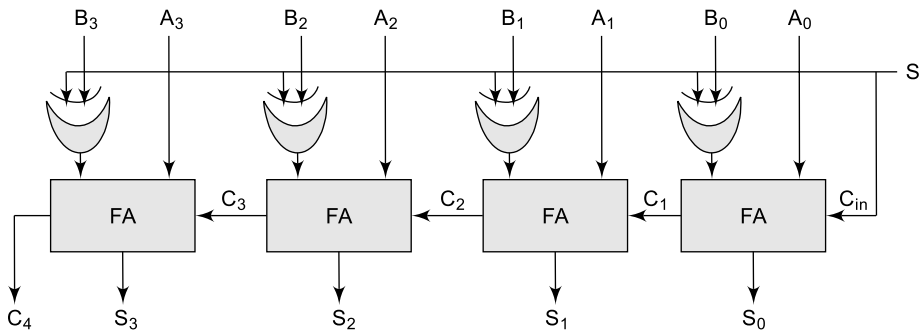
(a) A conducting coil called head does the data recording and retrieval from the disk. During read or write operation, the head is stationary while the disk platter rotates beneath it. The electricity flowing through the write coil that produces a magnetic field causes the write operation. Pulses are sent to the write head and magnetic patterns are recorded on the surface below, with different patterns for positive and negative currents. A head is shown in the figure below.



A read-wrtie head

The read and write signals pass through coils around a ring of soft magnetic material. A very narrow gap separates the ring from a storage cell on a track so that their respective magnetic field can induct. This induction permits information transfer between the head and the storage medium.

(b) The subtraction A – B is equivalent to A+ 2's complement of B (i.e. 1's complement of B +1). The addition and subtraction can be combined to a single circuit by using exclusive-OR (XOR) gate with each full adder. The 4-bit adder-subtractor circuit is shown in figure next. The selection input S determines the operation. When S = 0, this circuit performs the addition operation and when S = 1, this circuit performs subtraction operation.



4-bit Binary Adder-Subtractor

(c) The ALU of the CPU executes the instructions as dictated by the op-code field of instructions. The instructions are executed on some data stored in registers or memory. The different ways in which the location of an operand is specified in an instruction are referred to as *addressing modes*. A computer uses variety of addressing modes; some of them are described below:

1. *Implied (or Inherent) Mode:* In this mode the operands are indicated implicitly by the instruction. The accumulator register is generally used to hold the operand and after the instruction execution the result is stored in the same register. For example,

   RAL; Rotates the content of the accumulator left through carry.

2. *Immediate Mode:* In this mode the operand is mentioned explicitly in the instruction. In other words, an immediate-mode instruction contains an operand value rather than an address of it in the address field. To initialize registers to a constant value, this mode of instructions is useful. For example:

   MVI A, 06; Loads equivalent binary value of 06 to the accumulator

3. *Register (Direct) Mode:* In this mode the processor registers hold the operands. In other words, the address field is now register field, which contains the operands required for the instruction.
   For example:

   ADD R1, R2; Adds contents of registers R1 and R2 and stores the result in R1.

4. *Register Indirect Mode:* In this mode the instruction specifies an address of CPU register that holds the address of the operand in memory.
5. *Direct (or Absolute) Address Mode:* In this mode the instruction contains the memory address of the operand explicitly. Example of direct addressing is:

   STA 2500H ; Stores the content of the accumulator in the memory location 2500H.

6. *Indirect Address Mode:* In this mode the instruction gives a memory address in its address field which holds the address of the operand.
   For example:

   MOV R1, (X) ; Content of the location whose address is given in X is loaded into register R1.

7. *Relative Address Mode or PC-relative Address Mode:* In this mode the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction.

(d) Stack based computer operates instructions, based on a data structure called stack. A stack is a list of data words with a Last-In, First-Out (LIFO) access method that is included in the CPU of most computers. A portion of memory unit used to store operands in successive locations can be considered as a stack in computers. The register that holds the address for the top most operand in the stack is called a stack pointer (SP). The two operations performed on the operands stored in a stack are the *PUSH* and *POP*. From one end only, operands are pushed or popped. The PUSH operation results in inserting one operand at the top of stack and it decreases the stack pointer register. The POP operation results in deleting one operand from the top of stack and it increases the stack pointer register.

For example, figure below shows a stack of four data words in the memory. PUSH and POP instructions which require an address field. The PUSH instruction has the format:

PUSH <memory address>

The PUSH instruction inserts the data word at specified address to the top of the stack. The POP instruction has the format:

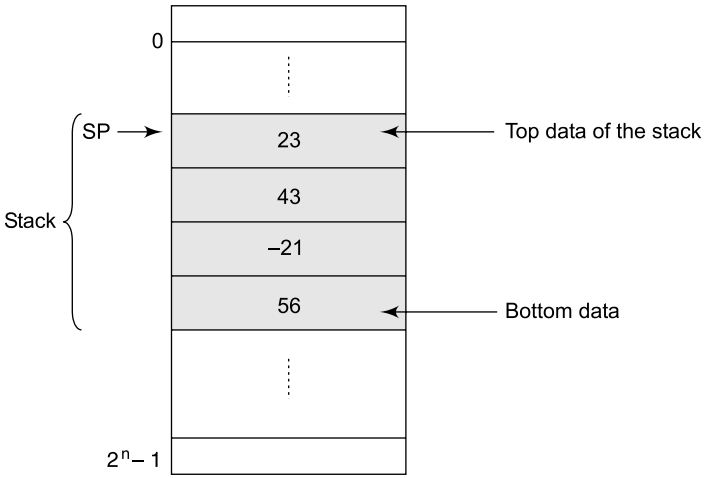
POP <memory address>

The POP instruction deletes the data word at the top of the stack to the specified address. The stack pointer is updated automatically in either case. The PUSH operation can be implemented as

SP ← SP – 1 ; decrement the SP by 1
SP ← <memory address> ; store the content of specified memory address into SP, i.e. at top of stack

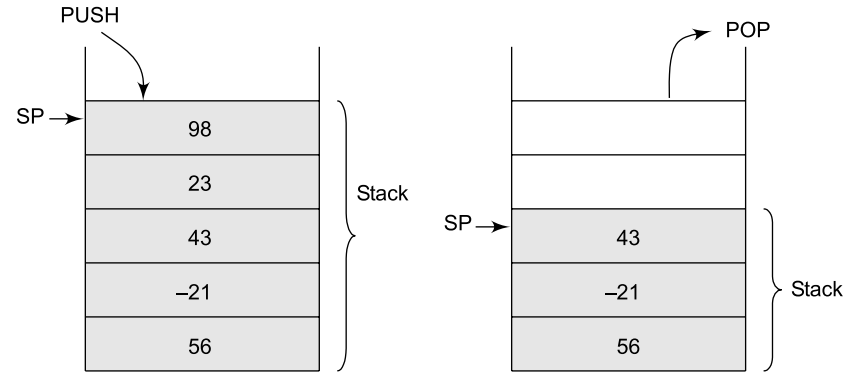
The POP operation can be implemented as

<memory address> ← SP ; transfer the content of SP (i.e. top most data) into specified memory location
SP ← SP + 1 ; increment the SP by 1



A stack of words in memory

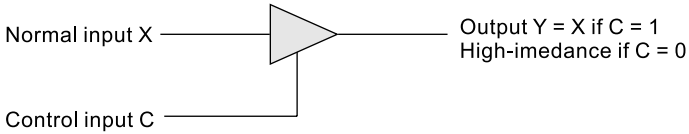
The figure next shows the effects of these two operations on the stack in figure above.



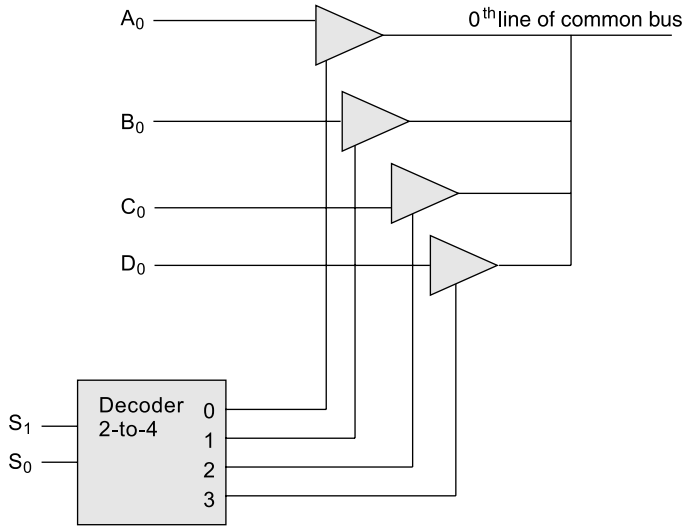
After PUSH of data (98) at top of stack

After POP of data (23) at top of stack

(e) A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The gate is controlled by one separate control input C. If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low the gate does not produce any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown below.



The common bus is used to transfer a register's content to other register or memory at a single time. A common bus system with tri-state buffers is described in the figure next, where one line of the common bus is shown.



A single line of bus system with tri-state buffers

Assume that there are four registers A, B, C and D. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

11. (a) Classify memory system in a digital computer according to their use.
    (b) A random access memory module of capacity 2048 bytes is to be used in a computer and mapped between 2000_H and $27FF_H$. Explain with the help of a block diagram the address decoding schema assuming 16 bit address bus.
    (c) How do the following influence the performance of a virtual memory system?
        (i) Size of page
        (ii) Replacement policy. 3 + 7 + 5

Answer

(a) The memory system consists of all storage devices used in a computer system and are broadly divided into following four groups:
- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory

Secondary Memory: The slow-speed and low-cost devices that provide backup storage are called secondary memory. The most commonly used secondary memories are magnetic disks, such as hard disk, floppy disk and magnetic tapes. This type of memory is used for storing all programs and data, as this is used in bulk size. When a program not residing in main memory is needed to execute, it is transferred from secondary memory to main memory. Programs not currently needed in main memory (in other words, the programs are not currently executed by the processor) are transferred into secondary memory to provide space for currently used programs and data.

Main Memory: This is the memory that communicates directly with CPU. Only programs and data currently needed by the CPU for execution reside in the main memory. Main memory occupies central position in hierarchy by being able to communicate directly with CPU and with secondary memory devices through an I/O processor.

Cache Memory: This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time.

Internal memory: This memory refers to the high-speed registers used inside the CPU. These registers hold temporary results when a computation is in progress. There is no speed disparity between these registers and the CPU because they are fabricated with the same technology. However, since registers are very expensive, only a few registers are used as internal memory.

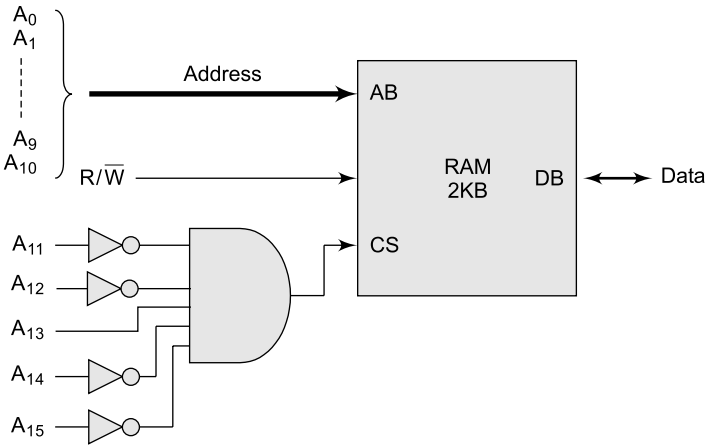
(b) Since the capacity of RAM memory is 2048 bytes i.e. 2 KB, the memory uses 11 ($2\ KB = 2^{11}$) address lines, say namely $A_{10} - A_0$, to select one word. Thus, memory's internal address decoder uses 11 lines $A_{10} - A_0$ to select one word.

To select this memory module, remaining 5 (i.e. 16 – 11) address lines $A_{15} - A_{11}$ are used. Thus, an external decoding scheme is employed on these higher-order five address bits of processor's address.

The address space of the memory is 2000_H to $27FF_H$.
Therefore, the starting address (2000_H) in memory is as:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Based on the higher-order five bits (00100), external decoding scheme performs a logical AND operation on address values: $\overline{A_{15}}$, $\overline{A_{14}}$, A_{13}, $\overline{A_{12}}$ and $\overline{A_{11}}$. The output of AND gate acts as chip select (CS) line. The address decoding scheme is shown in the figure below.



(c) (i) *Page size:* This is very important issue in designing virtual memory. For a given virtual-memory space, if the page size decreases, the number of pages increases and thus the size of page table increases. For example, a virtual memory of 2 MB, there will be 2048 pages of 1 KB, but only 1024 pages of 2 KB. Since each executing program must have its own copy of the page table, a large page size is advantageous.

On contrary, memory is better utilized with smaller pages. In other words, no or little space will be wasted inside a page, if it smaller. With smaller page size, total I/O time should be reduced, since locality of reference will be improved.

(ii) *Replacement policy:* When a page fault occurs, a page replacement is needed to select one of the existing pages in main memory to make the room for the required page. Designing appropriate replacement algorithms is an important task. We want one with the lowest page fault rate. There are several replacement algorithms such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement* algorithm available. Among these, optimal algorithm generally has the lowest page fault rate.

2007
Computer Architecture and Organization (CS-404)

Semester: 4th

Time Alloted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) With 2's complement representation, the range of values that can be represented on the data bus of an 8 bit microprocessor is given by
(a) –128 to + 127 (b) –128 to + 128 (c) –127 to + 128 (d) –256 to + 128.

Answer
(a) –128 to + 127

(ii) Booth's algorithm for computer arithmetic is used for
(a) multiplication of number in sign magnitude form
(b) multiplication of number in 2's complement form
(c) division of number in sign magnitude form
(d) division of number in 2's complement form.

Answer
(b) multiplication of number in 2's complement form

(iii) Micro instructions are kept in
(a) main memory (b) control store (c) cache (d) none of these.

Answer
(b) Control store

(iv) What is the 2's complement representation of –24 in a 16-bit microcomputer?
(a) 0000 0000 0001 1000 (b) 1111 1111 1110 1000
(c) 1111 1111 1110 0111 (d) 0001 0001 1111 0011.

Answer
(b) 1111 1111 1110 1000

(v) Associative memory is a
(a) pointer addressable memory
(b) very cheap memory
(c) content addressable memory
(d) slow memory.

Answer
(c) content addressable memory

(vi) The principle of locality justifies the use of
(a) interrupts
(b) polling
(c) DMA
(d) cache Memory.

Answer
(d) cache memory

(vii) In a microprocessor the address of the next instruction to be executed is stored in
(a) stack pointer
(b) address latch
(c) program counter
(d) general purpose register.

Answer
(c) program counter

(viii) A system has 48-bit virtual address, 36-bit physical address and 128 MB main memory; how many virtual and physical pages can the address spaces support?
(a) 2^{36}, 2^{24}
(b) 2^{12}, 2^{36}
(c) 2^{24}, 2^{34}
(d) 2^{34}, 2^{36}

Answer
(a) 2^{36}, 2^{24}

(ix) The basic principle of the von Neumann computer is
(a) storing program and data in separate memory
(b) using pipeline concept
(c) storing both program and data in the same memory
(d) using a large number of registers.

Answer
(c) storing both program and data in the same memory

(x) Physical memory broken down into groups of equal size is called
(a) page
(b) tag
(c) block
(d) index.

Answer
(c) block

Group-B
(Short-Answer Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. What is virtual memory? Why is it called virtual? Write the advantage of virtual memory.

Answer
Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, although which may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual (logical) address is mapped to a physical address in main memory. This mapping is done during run-time and is performed by a hardware device called *memory-management unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

Virtual memory is not a physical memory, is actually a technique. That is why it is called virtual memory.

The advantage of virtual memory is efficient utilization of main memory, because the larger size program is divided into blocks and partially each block is loaded in the main memory whenever it is required. Thus multiple programs can be executed simultaneously. The technique of virtual memory has other advantages of efficient CPU utilization and improved throughput.

3. What is meant by parallel processing? What is the basic objective of parallel processing?

Answer

Parallel processing is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. Parallel processing is the basis of parallel computers. The basic objective of the parallel processing is to improve the performance of a computer system by carrying out several tasks simultaneously.

4. What do you mean by instruction cycle, machine cycle and T states?

Answer

Instruction cycle: The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

Machine cycle: A machine cycle consists of necessary steps carried out to perform the memory access operation. Each of the basic operations such as fetch or read or write operation constitutes a machine cycle. An instruction cycle consists of several machine cycles.

T-states: One clock cycle of the system clock is referred to as T-state.

5. Distinguish between vectored interrupt and non-vectored interrupt.

Answer:

Interrupt is a special signal to the CPU generated by an external device that causes the CPU to suspend the execution of one program and the execution of another.

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially, the set-up time is quite large.

6. Compare RISC with CISC.

Answer

CISC	*RISC*
1. A large number of instruction types used - typically from 100 to 250 instructions.	1. Relatively few number of instruction types-typically less than100 instructions.
2. A large number of addressing modes used - typically from 5 to 15 different modes.	2. Relatively few addressing modes - typically less than or equal to 5.
3. Variable-length instruction formats.	3. Fixed-length, easily decoded instruction formats.
4. Small number of general-purpose registers (GPRs) - typically 8-24 GPRs.	4. Large number of general-purpose registers (GPRs)-typically 32-192 GPRs.
5. Clock per instruction (CPI) lies between 2 and 15.	5. Clock per instruction (CPI) lies between 1 and 2.
6. Mostly micro-programmed control units.	6. Mostly hardwired control units.
7. Most instructions manipulate operands in memory.	7. All operations are executed within registers of the CPU.

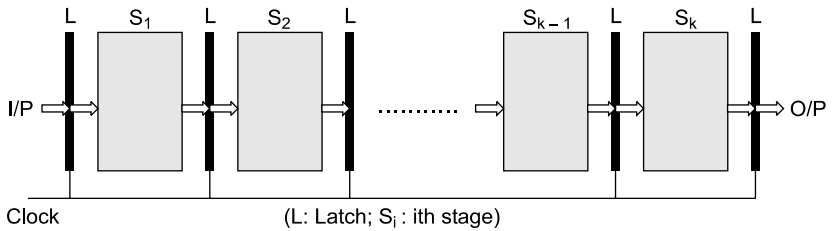
Group-B
(Long-Answer Questions)

Answer any *three* questions. $3 \times 15 = 45$

7. (a) What is pipelining ?
   (b) What are speedup, throughput and efficiency of a pipelined architecture?
   (c) Describe pipeline hazards.
   (d) Compare between centralized and distributed architecture. Which is the best architecture among them and why? $2 + 3 + 5 + 3 + 2$

Answer

(a) *Pipelining* is a technique of decomposing a sequential task into subtasks, with each subtask being executed in a special dedicated stage (segment) that operates concurrently with all other stages. Each stage performs partial processing dictated by the way the task is partitioned. Result obtained from a stage is transferred to the next stage in the pipeline. The final result is obtained after the instruction has passed through all the stages. All stages are synchronized by a common clock. Stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface latches (i.e. collection of registers). Figure below shows the pipeline concept with k stages.



Concept of pipelining

(b) *Speed-up*: It is defined as

$$S_k = \frac{\text{Time to execute n tasks in k-stage non-pipeline processor}}{\text{Time to execute n tasks in k-stage pipeline processor}}$$

$$= \frac{n \cdot k \cdot \tau}{\tau[k + (n-1)]}$$ where, τ = clock period of the pipeline processor.

Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n–1)]$ units, where k clock periods (cycles) are needed to complete the execution of the first task and remaining (n–1) tasks require (n–1) cycles. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can enter the pipeline until the previous task finishes.

It can be noted that the maximum speed-up is k, for $n >> k$. But this maximum speed-up is never fully achievable because of data dependencies between instructions, interrupts, program branches, etc.

Efficiency: To define it, we need to define another term "time-space span". It is the product (area) of a time interval and a stage space in the space-time diagram. A given time-space span can be in either a busy state or an idle state, but not both.

The *efficiency* of a linear pipeline is measured by the percentage of busy time-space spans over the total time-space spans, which equal the sum of all busy and idle time-space spans. Let n, k, τ be the number of tasks (instructions), the number of stages and the clock period of a linear pipeline, respectively. Then the efficiency is defined by

$$\eta = \frac{n \cdot k \cdot \tau}{k \cdot [k \cdot \tau + (n-1) \cdot \tau]} = \frac{n}{k + (n-1)}$$

Note that $\eta \to 1$ (i.e., 100%) as $n \to \infty$. This means that the larger the number of tasks flowing through the pipeline, the better is its efficiency. For the same reason as speed-up, this ideal efficiency is not achievable.

Throughput: The number of tasks that can be completed by a pipeline per unit time is called its throughput. Mathematically, it is defined as

$$\omega = \frac{n}{k \cdot \tau + (n-1) \cdot \tau} = \frac{\eta}{\tau}$$

Note that in ideal case, $\omega = 1/\tau = f$, frequency, when $\eta \to 1$. This means that the maximum throughput of a linear pipeline is equal to its frequency, which corresponds to one output result per clock period.

(c) Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. There are three types of pipeline hazards:

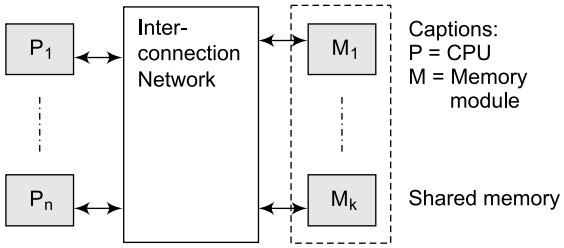
1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards: They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Structural hazards: Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

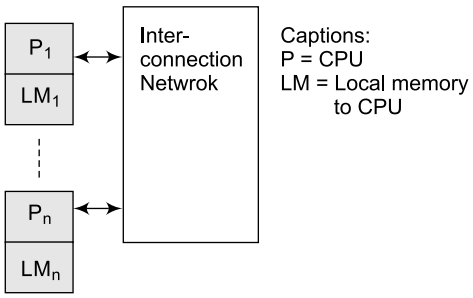
Data hazards: Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard*.

(d) In centralized architecture, all the processors access the physical main memory uniformly. All processors have equal access time to all memory words. The architecture is shown in the following figure.



Centralized system

In distributed system, a local memory is attached with each processor. All local memories distributed throughout the system form a global shared memory accessible by all processors. A memory word access time varies with the location of the memory word in the shared memory. The distributed system is depicted in the figure.



Distributed system

It is faster to access a local memory with a local processor. The access of remote memory attached to other processor takes longer due to the added delay through the interconnection network. Therefore, the distributed system is faster and in this regard, it is better.

8. (a) What is meant by DMA? Why is it useful? Briefly explain with suitable diagram, the DMA operation in association with CPU.
   (b) Draw the schematic diagram for daisy chain polling arrangement in case of vectored interrupt for three devices. 2 + 2 + 6 + 5

Answer

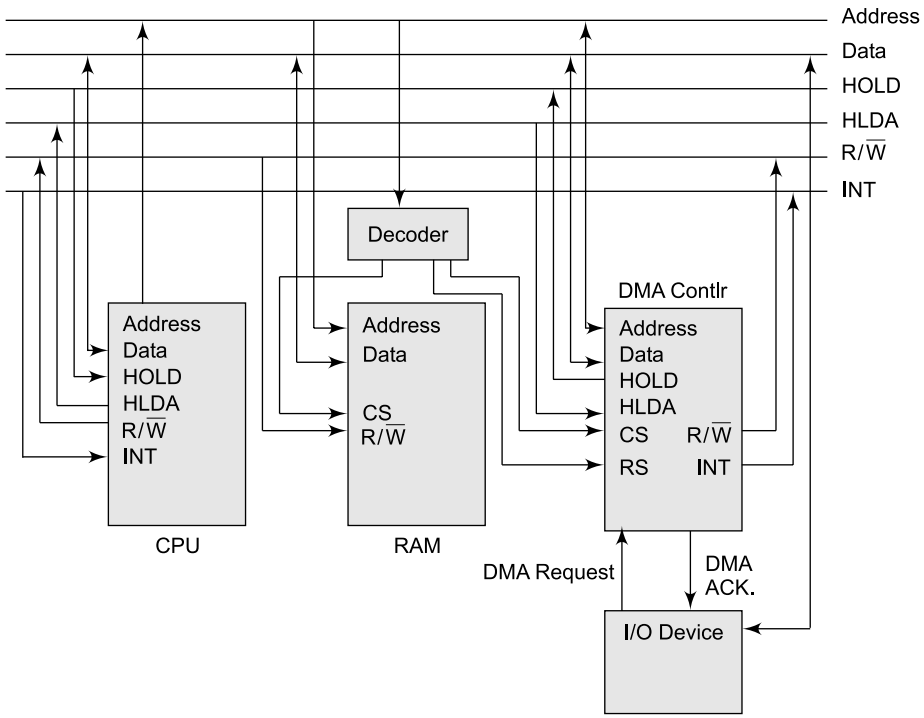
(a) A special controlling unit called DMA controller is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA).*

DMA is useful, because it has following advantages:

1. High speed data transfer is possible, since CPU is not involved during actual transfer, which occurs between I/O device and the main memory.

2. Parallel processing can be achieved between CPU processing and DMA controller's I/O operation.

In DMA transfer, I/O devices can directly access the main memory without intervention by the processor. The following figure shows a typical DMA system.

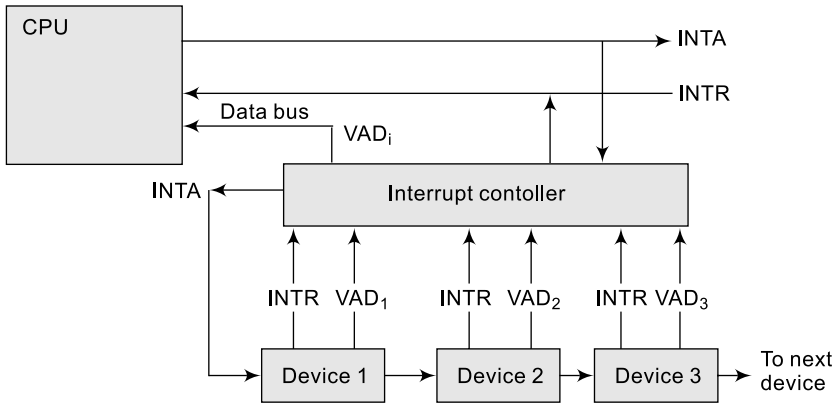


The sequences of events involved in a DMA transfer between an I/O device and the main memory are as follows:

A DMA request signal from an I/O device starts the DMA sequence. DMA controller activates the HOLD line. It then waits for the HLDA signal from the CPU. On receipt of HLDA, the controller sends a DMA ACK (acknowledgement) signal to the I/O device. The DMA controller takes the control of the memory buses from the CPU. Before releasing the control of the buses to the controller, the CPU initializes the address register for starting memory address of the block of data, word-count register for number of words to be transferred and the operation type (read or write). The I/O device can then communicate with memory through the data bus for direct data transfer. For each word transferred, the DMA controller increments its address-register and decrements its word count register. After each word transfer, the controller checks the DMA request line. If this line is high, next word of the block transfer is initiated and the process continues until word count register reaches zero (i.e., the entire block is transferred). If the word count register reaches zero, the DMA controller stops any further transfer and removes its HOLD signal. It also informs the CPU of the termination by means of an interrupt through INT line. The CPU then gains the control of the memory buses and resumes the operations on the program which initiated the I/O operations.

(b) To implement interrupts, the CPU uses a signal, known as an *interrupt request (INTR)* signal to the interrupt controller hardware, which is connected to each I/O device that can issue an

interrupt to it. Here, interrupt controller makes liaison with the CPU on behalf of I/O devices. Typically, interrupt controller is also assigned an *interrupt acknowledge (INTA)* line that the CPU uses to signal the controller that it has received and begun to process the interrupt request by employing an ISR (interrupt service routine). Devices are connected in daisy chain fashion, as shown in figure below, to set up priority interrupt system.



The devices are placed in a chain-fashion with highest priority device in the first place (device 1), followed by lower priority devices. The priorities are assigned by the interrupt controller. When one or more devices send interrupt signal through the interrupt controller to the CPU, the CPU then sets interrupt acknowledge (INTA) to the controller, which in turns sends it to the highest priority device. If this device has generated the interrupt INTR, it will accept the INTA; otherwise it will pass the INTA signal to the next device until the INTA is accepted by one requestor device. When the INTA is accepted by a device, device puts its own interrupt vector address (VAD) to the data bus using interrupt controller.

9. (a) Discuss the principle of carry look ahead adder and design a 4-bit CLA adder and estimate the speed enhancement with respect to ripple carry adder.
   (b) Briefly state the relative advantages and disadvantages of parallel adder over serial adder.
   (c) $X = (A + B) \times C$
   Write down the zero address, one address and three address instructions for the expression.
   $(4 + 3) + 2 + 6$

Answer

(a) Principle of CLA: A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

Design: In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1} = A_iB_i + (A_i+B_i)C_i$$

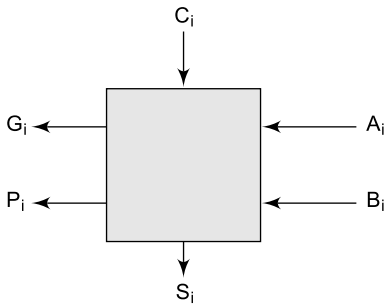
This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both

A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in the following figure. The sum bit S_i is = $A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.



Basic Adder (BA)

Now, to design a 4-bit CLA, four carries C_1, C_2, C_3 and C_4 are to be generated. Using Eq. (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2$$
$$C_4 = G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$

$$\begin{aligned} C_2 &= G_1 + P_1C_1 \\ &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P_1P_0C_0 \end{aligned} \quad (3)$$

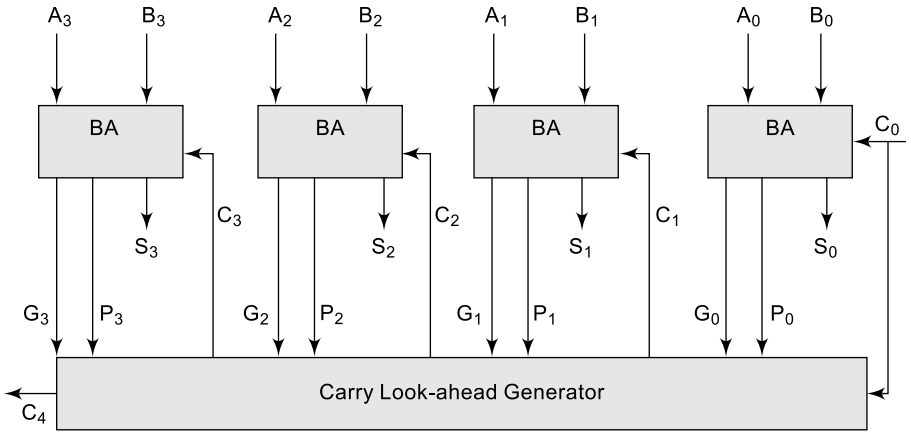
$$\begin{aligned} C_3 &= G_2 + P_2C_2 \\ &= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\ &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \end{aligned} \quad (4)$$

$$\begin{aligned} C_4 &= G_3 + P_3C_3 \\ &= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) \\ &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \end{aligned} \quad (5)$$

The Eqs (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in the figure next.

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay =Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It

depends on the number of levels of gates used to generate the sum and the carry bits. Whereas, the maximum propagation delay for CPA depends on size of inputs and for n-bit CPA it is $\Delta \times n$, where Δ is the time delay for each full adder stage and n is the number of bits in each operand.



4-bit Carry Look-ahead Adder (CLA)

(b) *Advantage of parallel adders over serial adders*: The parallel adder, being a combinational circuit, is faster than serial adder. In one clock period all bits of two numbers are added, whereas a serial adder requires n clock periods to add two n-bit numbers.

Disadvantages of parallel adders over serial adders:

1. The addition delay becomes large, if the size of numbers to be added is increased. But this remains same for serial adders.
2. The hardware cost is more than that of serial adder. Because, number of full adders needed is equal to the number of bits in operands.

(c) To evaluate this arithmetic expression, we use some op-codes as: LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations addition and multiplication respectively. Assume that the respective operands are in memory addresses A, B and C and the result must be stored in the memory at address X.

Using three-address instructions, the program code in assembly language is as:

```
ADD R1, A, B        ; R1 ← M[A] + M[B]
MULT X, C, R1       ; X ← M[C] + R1
```

Using two-address instructions, the program code in assembly language is as:

```
LOAD R1, A          ; R1 ← M[A]
ADD R1, B           ; R1 ← R1 + M[B]
LOAD R2, C          ; R2 ← M[C]
MULT R1, R2         ; R1 ← R1 * R2
STORE X, R1         ; X ← R1
```

Using one-address instructions, the program code in assembly language is as:

```
LOAD A              ; AC ← M[A]
ADD B               ; AC ← AC + M[B]
```

```
STORE T          ; T ← AC
LOAD C           ; AC ← M[C]
MULT T           ; AC ← AC * M[T]
STORE X          ; X ← AC
```

Using zero-address instructions, the program code in assembly language is as:

```
PUSH A           ; TOS ← A [TOS means top of the stack]
PUSH B           ; TOS ← B
ADD              ; TOS ← (A + B)
PUSH C           ; TOS ← C
MULT             ; TOS ← ((A + B) * C)
POP X            ; X ← TOS
```

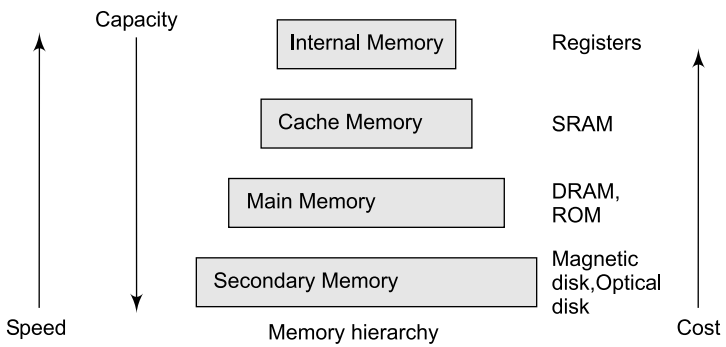
10. (a) Why do we require memory hierarchy ? Show the memory hierarchy diagram indicating the speed and cost.
    (b) Distinguish between SRAM and DRAM.
    (c) How many 256 × 4 RAM chips are needed to provide a memory capacity of 2048 bytes? Show also the corresponding interconnection diagram.
    (d) A disk drive has 20 sectors/track, 4000 bytes/sector, 8 surfaces all together. Outer diameter of the disk is 12 cm and inner diameter is 4 cm. Inter-track space is 0.1 mm. What is the no. of tracks, storage capacity of the disk drive and data transfer rate there from each surface? The disk rotates at 3600 rpm. (2 + 1) + 3 + (2 + 2) + 5

Answer

(a) Ideally, we would like to have the memory which would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three requirements simultaneously. If we increase the speed and capacity, then cost will increase. We can achieve these goals at optimum level by using several types of memories, which collectively give a memory hierarchy.

A memory hierarchy system broadly divided into following four groups, shown in figure below.

- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory



(b) Distinguish between SRAM and DRAM:

1. The SRAM has lower access time, which means it is faster compared to the DRAM.
2. The SRAM requires constant power supply, which means this type of memory consumes more power; whereas, the DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
3. Due to the relatively small internal circuitry in the one-bit memory cell of DRAMs, the large storage capacity in a single DRAM memory chip is available compared to the same physical size SRAM memory chip. In other words, DRAM has high packaging density compared to the SRAM.
4. SRAM is costlier than DRAM.

(c) The given RAM memory size is 256 × 4. This memory chip requires 8 (because $256 = 2^8$) address lines and 4 data lines.

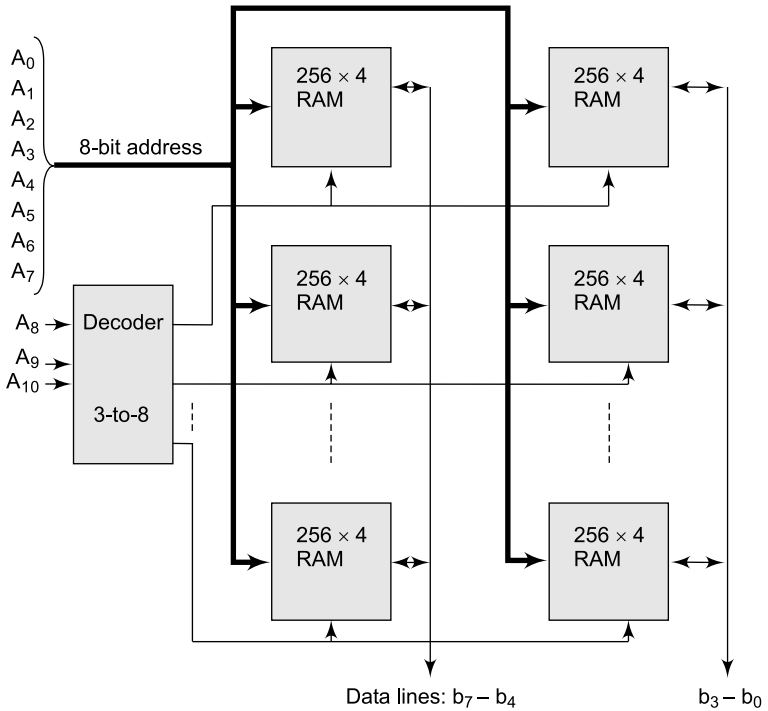
Size of memory to be constructed is 2048 bytes, which is equivalent to 2048 × 8. Thus, it requires 11 (because $2048 = 2^{11}$) address lines and 8 data lines.

In the interconnection diagram:

The number of rows required = 2048/256 = 8.

The number of columns required = 8/4 = 2.

Thus, total number of RAMs each of size 256 × 4 required = 8 * 2 = 16.



(d) Given

No. of sectors per track = 20

No. of bytes in each sector = 4000

No. of surfaces in disk pack = 8.

Outer diameter of disk = 12 cm.

Inner diameter = 4 cm.

Inter-track gap = 0.1 mm.

So, total width of track = (12 – 4)/2 = 4 cm.
No. of tracks per surface = (4 * 10)/0.1 = 400.
Thus, the total storage capacity of the disk drive = no. of surfaces * no. of tracks per surface * no. sectors per track * capacity of each sector = 8 * 400 * 20 * 4000 = 256000000 bytes = 244.14 MB (apprx.).
The rotational speed = 3600 rpm
So, the rotation time = 60/3600 sec = 1/60 sec.
Storage capacity of each track = 20 * 4000 bytes = 80000 bytes.
Thus, the data transfer rate = 80000/(1/ 60) = 4800000 bytes /sec = 4.578 MB/sec.

11. (a) Explain Booth's algorithm. Apply Booth's algorithm to multiply the two numbers $(+14)_{10}$ and $(-12)_{10}$. Assume the multiplier and multiplicand to be of 5 bits each.
    (b) Give the flowchart for division of two binary numbers and explain. 10 + 5

Answer

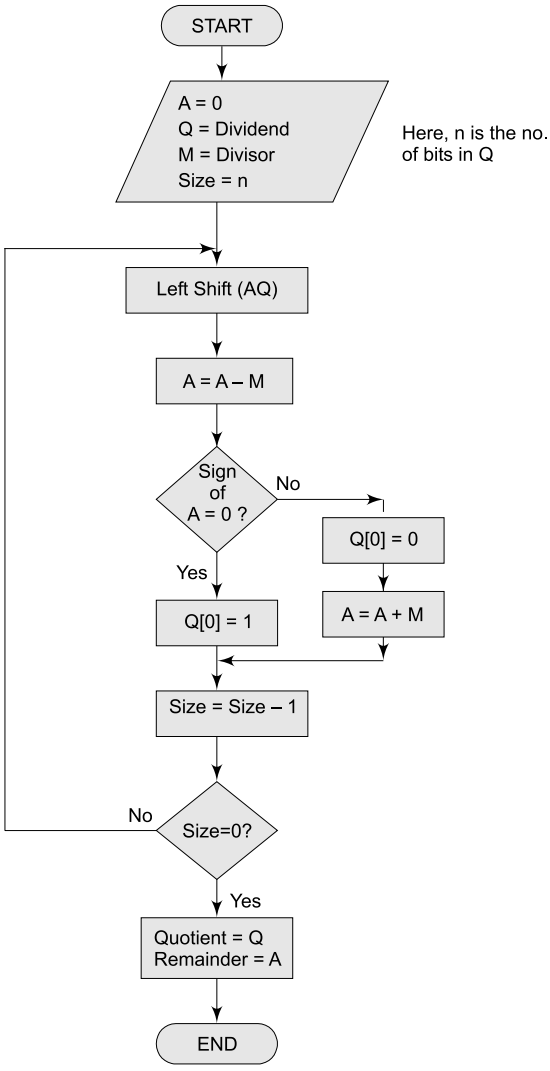
(a) For Booth's algorithm, see the answer of question no. 8(a) of 2007 (CS-303).
Multiplication of numbers $(+14)_{10}$ and $(-12)_{10}$:
Multiplicand, M = + 14 = 01110 and multiplier, Q = –12 = 10100.

	M	A	Q	Size
Initial Configuration	01110	00000	10100 0	5
Step-1				
As Q[0]=0 and Q[–1]=0				
ARS(AQ)	01110	00000	01010 0	4
Step-2				
As Q[0]=0 and Q[–1]=0				
ARS(AQ)	01110	00000	00101 0	3
Step-3				
As Q[0]=1 and Q[–1]=0				
A = A – M	01110	10010	00101 0	–
ARS(AQ)	01110	11001	00010 1	2
Step-4				
As Q[0]=0 and Q[–1]=1				
A = A + M	01110	00111	00010 1	-
ARS(AQ)	01110	00011	10001 0	1
Step -5				
As Q[0] = 1and Q[–1] = 0				
A = A – M	01110	10101	10001 0	-
ARS (AQ)	01110	11010	11000 1	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 1101011000, which shows that the product is a negative number. To get the result in familiar form, take the 2's complement of the magnitude of the number and the result is –168 in decimal.

(b) The restoring division method uses three n-bit registers A, M, Q for dividing two n-bit numbers. The register M is used to hold the divisor. Initially, A contains 0 and Q holds the n-bit dividend. In each iteration, the contents of register-pair AQ are shifted to the left first. The content of M is then subtracted from A. If the result of subtraction is positive, a 1 is placed into the vacant position created in lsb position of Q by the left shift operation; otherwise a 0 is put into this position and before beginning the next iteration, restore the content of A by adding the current content of A register with M. For this step, the algorithm is referred to as a restoring division algorithm. When the algorithm terminates, the A register contains the remainder result and the Q register contains the quotient result.

The restoring division algorithm to divide two n-bit numbers is described using the flowchart shown in figure below.



Restoring division algorithm

The algorithm discussed is used for division of two unsigned integers. This algorithm can be extended to handle signed numbers as well. The sign of the result must be treated separately and the positive magnitudes of the dividend and divisor are performed using this technique for quotient and remainder. The sign of the quotient is determined as $M_n \oplus Q_n$, where M_n, Q_n are the signs of the divisor (M) and the dividend (Q) respectively.

2007
Computer Architecture and Organization (EC-503)

Time Alloted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) What is the 2's complement representation of –24 in a 16-bit microcomputer?

(a) 0000 0000 0001 1000 (b) 1111 1111 1110 1000

(c) 1111 1111 1110 0111 (d) 0001 0001 1111 0011

Answer

(b) 1111 1111 1110 1000

(ii) The basic principle of the von Neumann computer is

(a) storing program and data in separate memory

(b) using pipeline concept

(c) storing both program and data in the same memory

(d) using a large number of registers.

Answer

(c) storing both program and data in the same memory

(iii) In a microprocessor the address of the next instruction to be executed is stored in

(a) stack pointer (b) address latch

(c) program counter (d) general purpose register

Answer

(c) program counter

(iv) For BIOS (Basic Input/Output System) and IOCS (Input/Output Control System), which one of the following is true?

(a) BIOS and IOCS are same

(b) BIOS controls all devices and IOCS controls only certain devices
(c) BIOS is not a part of operating system and IOCS is a part of operating system
(d) BIOS is stored in ROM and IOCS is stored in RAM.

Answer

(c) BIOS is not a part of operating system and IOCS is a part of operating system

(v) The principle of locality justifies the use of
(a) interrupts (b) polling (c) DMA (d) cache memory

Answer

(d) cache memory.

(vi) The performance of a pipelined processor suffers if
(a) the pipeline stages have different delays
(b) consecutive instructions are dependent on each other
(c) the pipeline stages share hardware resources
(d) all of these

Answer

(d) all of these.

(vii) "Delayed Branching" is related to
(a) pipeline hazard (b) pipeline remedy (c) both (a) and (b) (d) none of these

Answer

(b) pipeline remedy

(viii) How many RAM chips of size (256 × 1 bit) are required to build 1 M byte memory?
(a) 8 (b) 10 (c) 24 (d) 32

Answer

(b) pipeline remedy

(ix) The mode field determines
(a) the type of addressing (b) the type of operand
(c) the type of instruction format (d) the type of arithmetic or logic operation.

Answer

(a) the type of addressing

(x) By left-shifting the content of a register once, its content is
(a) doubled (b) halved
(c) both (a) and (b) (d) no such decision can be made

Answer

(a) doubled

Group-B
(Short-Answer Type Questions)

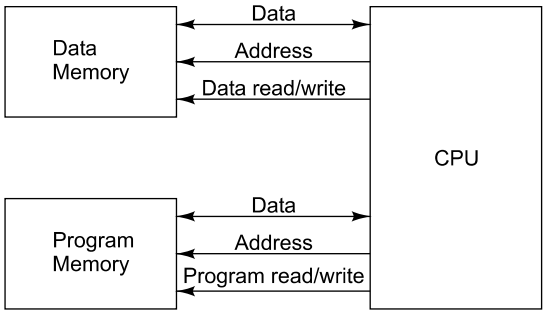
Answer any *three* of the following. 3 × 5 = 15

2. What is Harvard architecture? Explain briefly using a block diagram. 1 + 4

Answer

In Harvard architectures, separate program and data memories are used. Data memory and program memory can be different widths, type, etc. Program and data can be fetched in one

cycle by using separate control signals- 'program memory read' and 'data memory read'. Example includes Harvard Mark 1 computer. The block diagram of Harvard architecture is shown in the figure below.



Harvard architecture

In a Harvard computer, the CPU can both read an instruction and perform a data memory access at the same time, even without a cache. A Harvard architecture computer can thus be faster than a von Neumann computer (which uses common memory for both data and instructions) for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

Modern high performance CPU chip designs incorporate aspects of both Harvard and von Neumann architectures. Harvard architecture is used as the CPU accesses the cache. On-chip cache memory is divided into an instruction cache and a data cache. In the case of a cache miss, however, the data is retrieved from the main memory, which contains both instruction and data collectively. Thus, the von Neumann architecture is used as the CPU accesses the main memory.

The Harvard architectures are used frequently in specialized Digital Signal Processors (DSPs) for vedio and audio applications.

3. Sketch the instruction format of a two address instruction that uses immediate, register direct and indexed addressing mode if size of the memory is 1 MB and size of instruction word is limited to 16 bits with 3 bit op-code field. 5

Answer

Specifications:

Memory size = 1MB; so 20 address bits are used

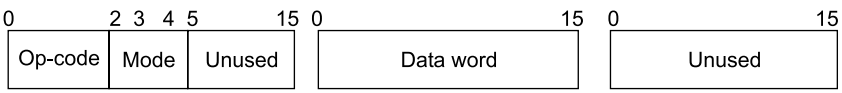
3 bit op-code.

3 addressing modes; so 2 mode bits are used.

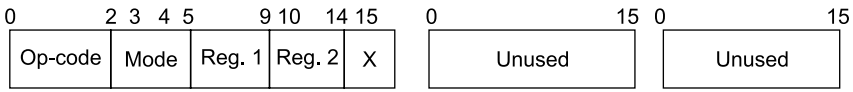
The instruction is 2-address instruction; so a total of 45 bits (i.e. 3 + 2 + 20 + 20) are required for the instruction.

Since the word size is 16-bit, the instruction must be of 3-word.

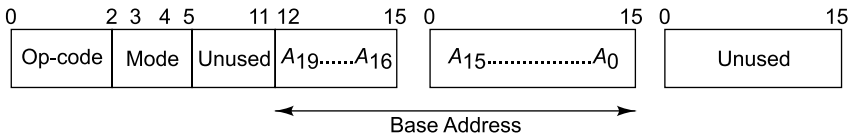
(i) *Immediate Mode:* The data word is kept in the second word and third word is left vacant. The format is shown below:



(ii) *Register Direct Mode:* Assuming 32 CPU registers each of 16-bit, 5 bits are used to specify one register. The format is shown below, where X stands for unused.



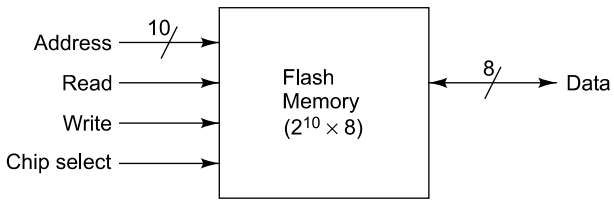
(iii) *Indexed Addressing Mode:* The 20-bit base address is stored as part of first two words and third word is unused. The format is shown below:



4. What is flash memory? Explain with an example. 2 + 3

Answer

A currently popular type of non-volatile EEPROM (Electrically Erasable Programmable ROM), in which erasing is performed in large blocks rather than bit by bit, is known as flash EPROM or flash memory. Erasing in large blocks reduces the overhead circuitry, thus leading to greater density and lower cost. Flash memory gets its name because the memory chip is organized so that a section of memory cells are erased in a single action i.e. 'flash'. The current trend is "memory stick" made of flash memory that is used to Universal Serial Bus (USB) of the personal computer for data exchange between computers. The following figure shows a block diagram of an 1-KB flash memory.

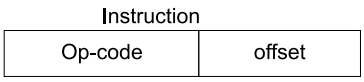


1 KB Flash memory

5. What are the advantages of relative addressing mode over direct addressing mode? 5

Answer

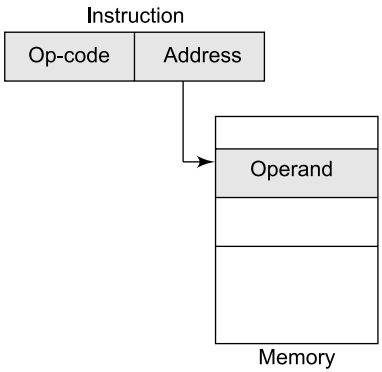
In relative addressing mode, the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. The instruction specifies the memory address of operand as the relative position of the current instruction address. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address.



Operand (i.e. effective) address = content of PC + offset.

Relative addressing mode

In direct addressing mode, the instruction contains the memory address of the operand explicitly. Thus, the address part of the instruction is the effective address.



Direct addressing mode

The advantages of relative addressing mode over direct addressing mode:

- In relative addressing mode, smaller number of bits are used as the address of the operands compared to the direct addressing mode.
- Since the size of relative addressed mode instructions is shorter than that of direct mode instructions, the relative addressed mode instructions occupy lesser memory space, which decreases the memory requirement.
- Due to the smaller size for relative addressing mode instructions, either data bus width is small or instruction fetch takes less time.

6. Distinguish between arithmetic pipeline and instruction pipeline. 5

Answer

See answer of question 6(a) of 2007 (CS-303).

Group-C
(Long-Answer Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. (a) With the help of a block diagram discuss the construction and working of an 8-bit carry-look-ahead adder. Also compute total time needed to perform one addition using gate delay of each gate ä ìs and no delay are involved in the connecting wires. 8 + 3

   (b) What are the advantages of carry-look-ahead adder over ripple-carry adders? Explain. 4

Answer

(a) A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

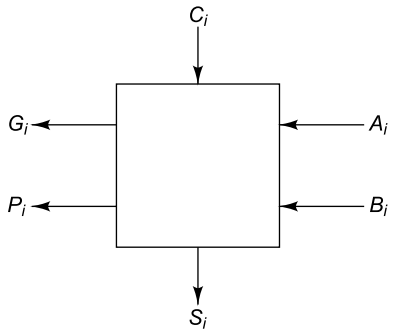
$$C_{i+1} = A_iB_i + (A_i + B_i)\, C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \qquad (1)$$

where $G_i = A_iB_i$ and $Pi = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (*BA*) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in figure below. The sum bit S_i is $= A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.



Basic Adder

Now, for an 8-bit CLA, eight carries C_1, C_2, …, C_8 are to be generated. Using equation number (1); C_1, C_2, …, C_8 can be expressed as follows:

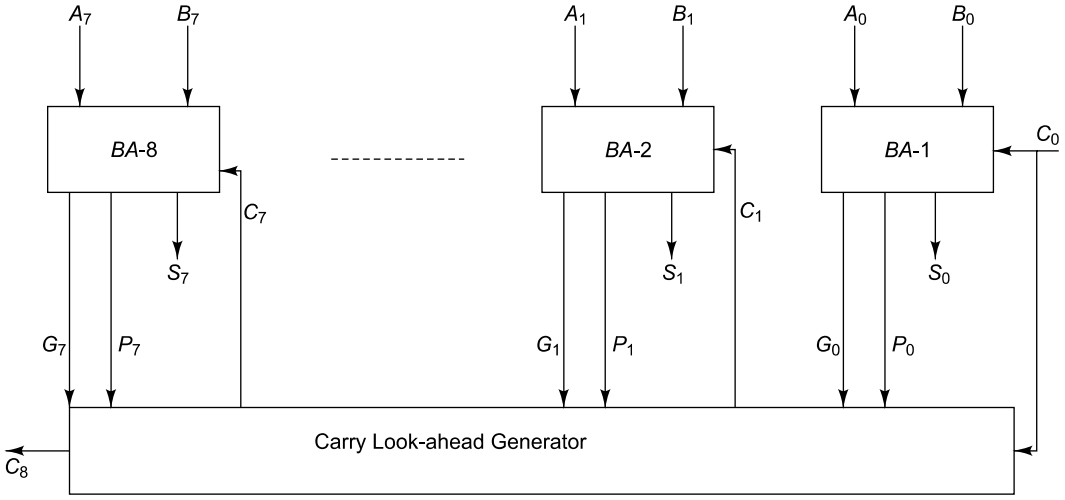
$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_8 = G_7 + P_7C_7$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$
$$= G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$
$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$

Similarly, C_4, C_5, C_6, C_7 and C_8 can be expanded to remove the recursion.

The equations (2), (3), (4) and others, if derived, suggest that C_1, C_2, …, C_8 can be generated directly from C_0. In other words, these eight carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. An 8-bit carry look-ahead adder (CLA) is shown in figure below.



8-bit Carry Look-ahead Adder (CLA)

Total time needed to perform one addition:
The maximum delay of the CLA is $6 \times \delta$ (for G_i and P_i generation, delay = δ, for C_i generation, delay = 2δ and lastly another 3δ for sum bit S_i) where δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

(b) The advantages of carry-look-ahead adder over ripple-carry adders:

- Generally, the carry look-ahead adder (CLA) is faster than the ripple carry adder (RCA). Because, the maximum delay of the CLA is $6 \times \delta$, where δ is the average gate delay and this holds good for any size numbers. However, the maximum propagation delay for n-bit RCA is δ x n, where n is the number of bits in each operand.
- The RCA becomes slow once the sizes of operands are increased, which is not true for CLA. The speed of CLA remains same irrespective of sizes of operands.

8. (a) Using Booth's algorithm, multiply (+14) and (–12) when the numbers are represented in 2's complement form. 9
   (b) Compare and contrast restoring and non-restoring divisions. 6

Answer

(a) See answer of question no. 11(a) of 2007 (CS-404).
(b) See answer of question 5(b) of 2007 (CS-303).

9. (a) Explain Flynn's classification for multi-processor system. 5
   (b) Discuss the advantages of vector processing over scalar processing. 5
   (c) Explain how daisy chaining is used for bus arbitration in a multiprocessor system. 5

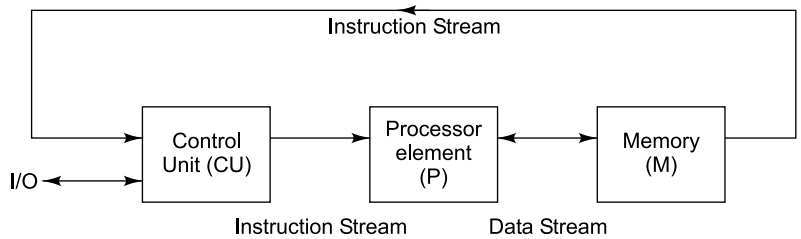
Answer

(a) *Flynn's classification*: Based on the number of simultaneous instruction and data streams used by a CPU during program execution, digital computers can be classified into four categories as:

- Single instruction stream-single data stream (SISD) machine.
- Single instruction stream-multiple data stream (SIMD) machine.
- Multiple instruction stream-single data stream (MISD) machine.
- Multiple instruction stream-multiple data stream (MIMD) machine.

SISD Computer

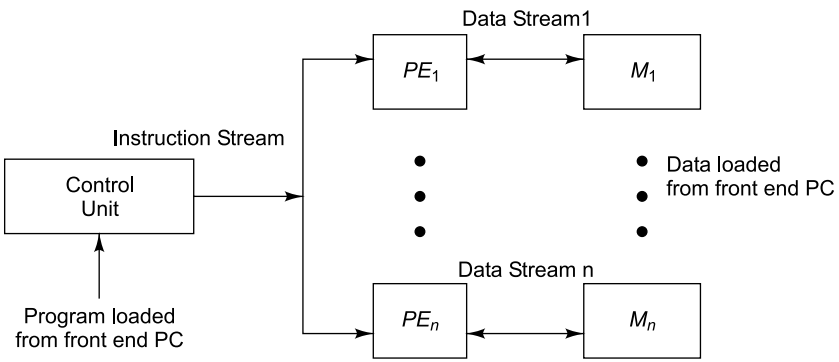
Most serial computers available today falls in this organization as shown in figure next. Instructions are executed sequentially but may be overlapped in their execution stages (In other words the technique of pipelining can be used in the CPU). Modern day SISD uniprocessor systems are mostly pipelined. Examples of SISD computers are IBM 360/91, CDC Star-100 and TI-ASC.



SISD Computer

SIMD Computer

Array processors falls into this class. As illustrated in figure next, there are multiple pro-cessing elements supervised by the common control unit. All PEs (processing elements, which are essentially ALUs) receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams. The shared memory subsystem containing multiple modules is very essential. This machine generally used to process vector type data. Examples of SIMD computers includes Illiac-IV and BSP.

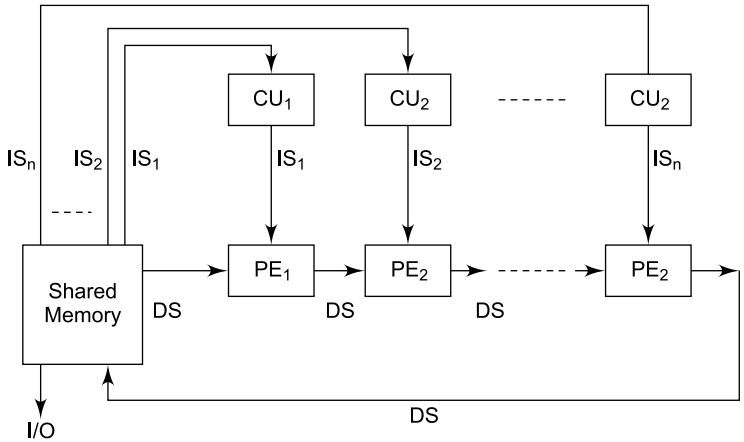


SIMD Computer

MISD Computer

Very few or no parallel computers fit in this organization, which is conceptually illustrated in figure next. There are n processor elements, each receiving distinct instructions to execute on the same data stream and its derivatives. The results (outputs) of one processor

element become the inputs (operands) of the next processor element in the series. This architecture is also known as *systolic arrays*.

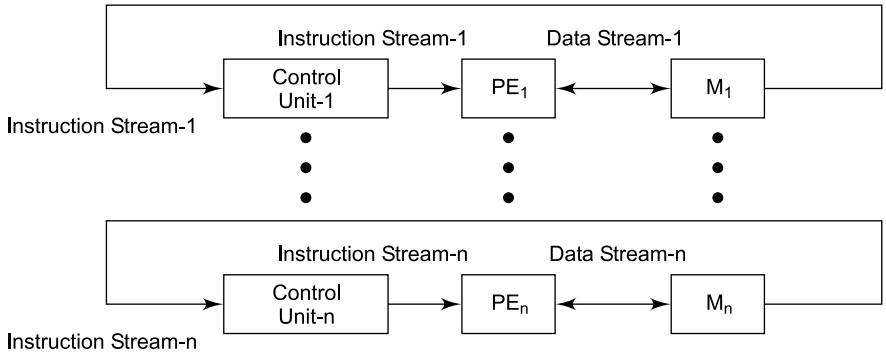


Captions: CU: control unit PE: processing element IS: instruction stream DS: data stream.

MISD computer (Systolic array)

MIMD Computer

This category covers multiprocessor systems and multiple com-puter systems. The structure of MIMD computer is shown in figure next. An MIMD computer is called *tightly coupled (or Uniform Memory Access (UMA))* if the degree of interactions among the processors is high. Otherwise, we consider them *loosely coupled (or Non-Uniform Memory Access (NUMA))*. Most commercial MIMD computers are loosely coupled. Examples of MIMD multiprocessors are C.m*, C.mmp, Cray-3 and S-1.



MIMD computer

(b) To examine the advantages of vector processing over scalar processing, we compare the following two programs to perform the same task, one written for vector processing and the other written for scalar processing.

Example: In a conventional scalar processor, the 'for' loop

```
For I = 1 to N do
    A(I) = B(I)+C(I);
End
```

is implemented by the following sequence of scalar operations:

```
        INITIALIZE I=1;
LABEL:  READ B(I);
        READ C(I);
        ADD B(I)+C(I);
        STORE A(I) = B(I)+C(I);
        INCREMENT I = I+1;
        IF I <= N GO TO LABEL;
        STOP;
```

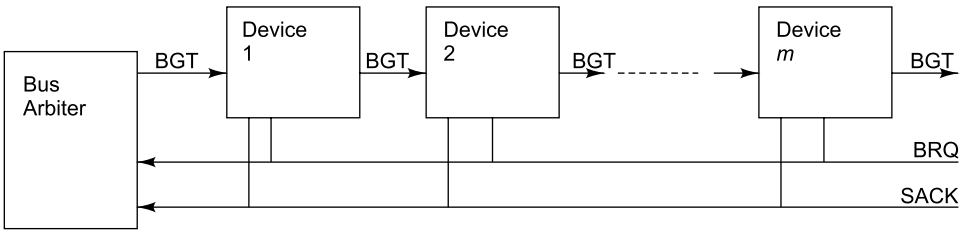
In a vector processor, the above 'for' loop operation can be vectorized into one vector instruction as:

```
A (1: N)=B(1: N)+C(1: N);
```

where A(1: N) refers to the N-element vector consisting of scalar components A(1), A(2),..., A(N). To execute this vector instruction, vector processor uses an adder pipeline. Thus, the advantages of vector processing over scalar processing can be summarized as:

- The execution of the scalar loop repeats the loop-control overhead in each iteration which is eliminated by using hardware or firmware controls in vector processing using pipelines. A vector-length register is used to control the vector operations.
- Due to the usage of various pipelines in vector processors, the overall throughput is much higher than that of scalar processors.
- In vector processing using pipelines, sharing of resources like memory, bus, etc is possible in larger extend.

(c) The daisy chaining method is a centralized bus arbitration method. During any bus cycle, the bus master may be any device - a processor or any DMA controller unit, connected to the bus. Figure below illustrates the daisy chaining method, where devices are processors.



Daisy chained bus arbitration

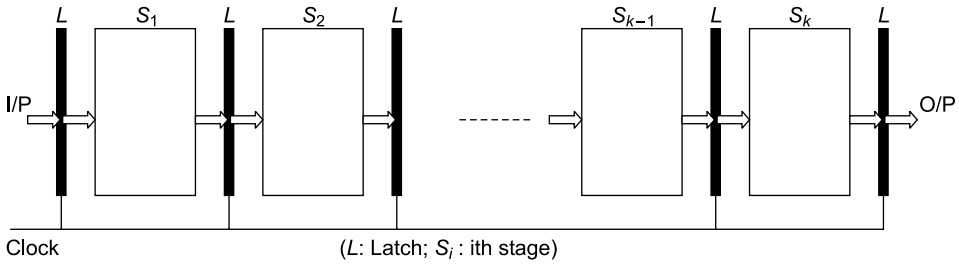
All devices (processors) are effectively assigned static priorities according to their locations along a bus grant control line (BGT). The device (processor) closest to the central bus arbiter is assigned the highest priority. Requests for bus access are made on a common request line, BRQ. Similarly, the common acknowledge signal line (SACK) is used to indicate the use of bus. When no device (processor) is using the bus, the SACK is inactive. The central bus arbiter propagates a bus grant signal (BGT) if the BRQ line is high and acknowledge signal (SACK) indicates that the bus is idle. The first device, which has issued a bus request, receives the BGT signal and stops the latter's propagation. This sets the bus-busy flag in the bus arbiter by activating SACK and the device assumes bus control. On completion, it resets the bus-busy flag in the arbiter and a new BGT signal is generated if other requests are outstanding (i.e., BRQ is still active). The first device simply passes the BGT signal to the next device in the line.

The main advantage of the daisy chaining method is its simplicity. Another advantage is scalability. The user can add more devices anywhere along the chain, up to a certain maximum value.

10. (a) What is meant by "pipeline architecture"? 2
    (b) How does it improve the speed of execution of processor? 5
    (c) What are pipeline hazards? 3
    (d) A non-pipeline system takes 40 ns to process a task. The same task can be processed in a four segment pipeline with a clock cycle of 10 ns. Determine the speed up ratio of the pipeline for 50 tasks. What is the maximum speed up that can be achieved in this case? 5

Answer

(a) Pipeline architecture performs overlapped computations to exploit temporal parallelism; in other words, pipeline architecture uses one parallel processing concept, known as pipelining. *Pipelining* is a technique of decomposing a sequential task into subtasks, with each subtask being executed in a special dedicated stage (segment) that operates concurrently with all other stages. Each stage performs partial processing dictated by the way the task is partitioned. Result obtained from a stage is transferred to the next stage in the pipeline. The final result is obtained after the instruction has passed through all the stages. All stages are synchronized by a common clock. Stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface latches (i.e. collection of registers). Figure below shows the pipeline concept with k stages.



Concept of pipelining

(b) Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n - 1)]$ units, where τ = clock period of the pipeline processor. Since, k clock periods (cycles) are needed to complete the execution of the first task and remaining $(n - 1)$ tasks require $(n - 1)$ cycles, because of overlapped execution of tasks. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can start its execution until the previous task finishes.
Thus, the speed-up (S_k)of pipeline processor over its equivalent non-pipeline processor is defined as:

$$S_k = \frac{\text{Time to execute } n \text{ tasks in k-stage non-pipeline processor}}{\text{Time to execute } n \text{ tasks in k-stage pipeline processor}}$$
$$= \frac{\text{n.k.}\tau}{\tau(k + (n - 1)}$$

(c) See answer of question 6(b) of 2007 (CS-303).

(d) In non-pipeline processor, a task takes 40 ns and there are 50 tasks.
So, the total time to execute all tasks in non-pipeline processor is $40 \times 50 = 2000$ ns.
In case of 4-stage (i.e. $k = 4$) pipeline with a clock cycle of 10 ns (i.e. $\tau = 10$), the total time to execute the same number of tasks $= \tau[k + (n - 1)]$
$= 10\ [\ 4 + 49]$; by putting $n = 50$
$= 530$ ns.
Thus, the speed up ratio of the pipeline = 2000/530 = 3.77
It can be noted, from the speed-up relation in answer of question no. 10(b), that the maximum speed-up is k, for $n >> k$. So, the maximum speed up that can be achieved in this case is $k = 4$.

11. Write short notes on any *three* of the following: 3×5
(a) Microprogramming and microprogrammed control unit
(b) Page replacement policies
(c) Interrupt servicing with priority interrupts
(d) Vector processors and their uses
(e) Architecture of IOP.

Answer

(a) Microprogramming and microprogrammed control unit: Microprogramming is a modern approach to design a control unit. In the microprogrammed approach, all control functions that can be simultaneously activated are grouped to form control words stored in a separate ROM memory called the control memory. From the control memory, the control words are fetched one at a time and the individual control fields are routed to various functional units to activate their appropriate circuits. The desired task is performed by activating these circuits sequentially.

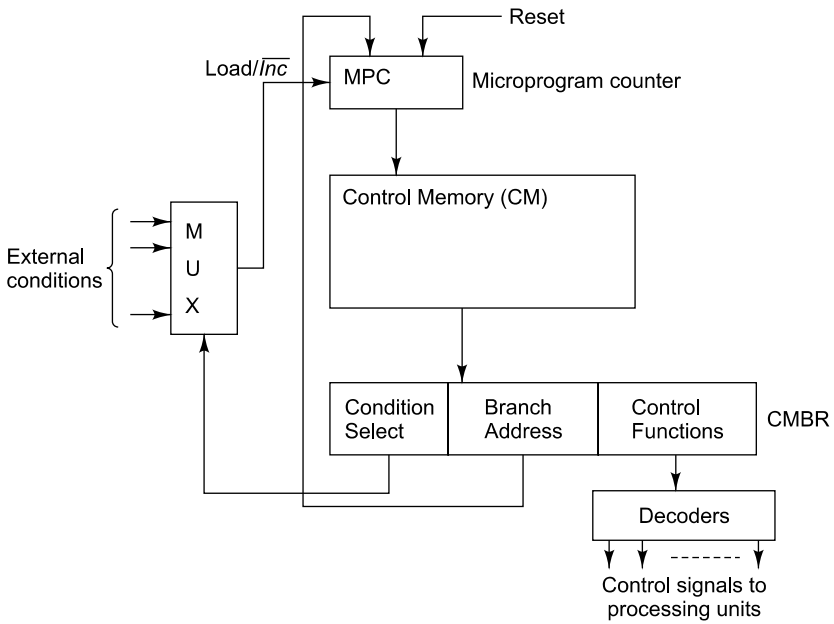
Like conventional program, retrieval and interpretation of the control words are done. The instructions of a CPU are stored in the main memory. They are fetched and executed in a sequence. The CPU can perform different functions simply by changing the instructions stored in the main memory. Similarly, the control unit can execute a different control operation by changing the contents of the CM. Hence, the microprogrammed approach

offers greater flexibility than its hardwired counterpart, since this approach is based on the programming concept giving an easy way for altering the contents of the CM.

Usually, all microinstructions have three important fields:

- Control field
- Next-address field
- Condition for branching.

We now describe the design of a typical microprogrammed control unit. The architecture of a typical modern microprogrammed control unit is shown in the figure below.



General-purpose microprogrammed control unit

The various components used in the figure above are summarized next.

Control memory buffer register (CMBR): The function of CMBR is same as the MBR (memory buffer register) of the main memory. It is basically a latch and acts as a buffer for the microinstructions retrieved from the CM. Typically, each microinstruction has three fields as:

Condition select	Branch address	Control functions

The condition select field selects the external condition to be tested. The output of the MUX will be 1, if the selected condition is true. The MPC will be loaded with the address specified in the branch address field of the microinstruction, because the output of the MUX is connected to the load input of the microprogram counter (MPC). However, the MPC will point to the next microinstruction to be executed, if the selected external condition is false. Thus, this arrangement allows conditional branching. The control function field of

the microinstruction may hold the control information in an encoded form which thus may require decoders.

Microprogram counter (MPC): The task of MPC is same as the PC (program counter) used in the CPU. The address of the next microinstruction to be executed is held by the MPC. Initially, it is loaded from an external source to point to the starting address of the microprogram to be executed. From then on, the MPC is incremented after each microinstruction fetch and the instruction fetched is transferred to the CMBR. However, the MPC will be loaded with the contents of the branch address field of the microinstruction that is held in the CMBR, when a branch instruction is encountered.

External condition select MUX: Based on the contents of the condition select field of the microinstruction, this MUX selects one of the external conditions. Therefore, the condition to be selected must be specified in an encoded form. Any encoding leads to a short microinstruction, which implies a small control memory; hence the cost is reduced. Suppose two external conditions X_1, X_2 are to be tested; then the condition-select and actions taken are summarized next:

Condition select	*Action taken*
00	No branching
01	Branch if X_1 =1
10	Branch if X_2 =1
11	Always branching (unconditional branching)

The multiplexer has four inputs V_0, V_1, V_2, V_3 where V_i is routed to the multiplexer's output when the condition select field has decimal equivalent i. Hence we require $V_0 = 0$, $V_1 = X_1$, $V_2 = X_2$, $V_3 = 1$ to control the loading of microinstruction branch addresses into MPC.

(b) *Page replacement policies:* When a program starts execution, one or more pages are brought to the main memory and the page table is responsible to indicate their positions. When the CPU needs a particular page for execution and that page is not in main (physical) memory (still in the secondary memory), this situation is called *page fault*. When the page fault occurs, the execution of the present program is suspended until the required page is brought into main memory from secondary memory. The required page replaces an existing page in the main memory, when it is brought into main memory. Thus, when a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several replacement algorithms such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement* algorithm available.

The *FIFO algorithm* is simplest and its criterion is "select a page for replacement that has been in main memory for longest period of time".

The *LRU algorithm* states that "select a page for replacement, if the page has not been used often in the past". The LRU algorithm is difficult to implement, because it requires a counter for each page to keep the information about the usage of page.

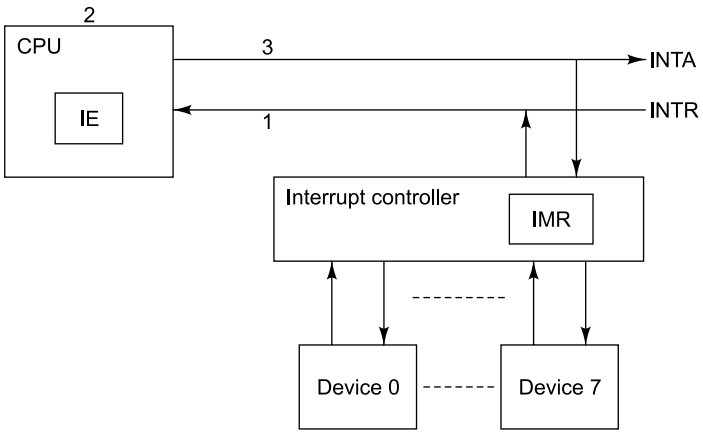
The *optimal algorithm* generally gives the lowest page faults of all algorithms and its criterion is "replace a page that will not be used for the longest period of time". This

algorithm is also difficult to implement, because it requires future knowledge about page references.

An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a *reference string*.

(c) *Interrupt servicing with priority interrupts:* In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt controller is to identify the source of the interrupt. There is also the possibility that several sources may request interrupt service simultaneously. In this case the controller must also decide which to service first. A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. Devices with high-speed transfers such as magnetic disks are usually given high priority and slow devices such as keyboards receive low priority. When two devices interrupt the CPU at the same time, the CPU services the device, with the higher priority first.

The following figure implements the interrupts. The interrupt requests from various sources are connected as input to the interrupt controller. As soon as the interrupt controller senses (using IMR) the presence of any one or more interrupt requests, it immediately issues an interrupt signal through INTR line to the CPU. The CPU uses a flag bit known as *interrupt enable (IE)* in its status register (PS) to process the interrupt. When this flag bit is '1', the CPU responds to the presence of interrupt by enabling INTA line; otherwise not. The interrupt controller assigns a fixed priority for the various interrupt requestor devices. For example, the IRQ0 is assigned the highest priority among the eight different interrupt requestors. Assigning decreasing order of priority from IRQ0 to IRQ7, the IRQ7 is the lowest priority. It (IRQ7) is serviced only when no other interrupt request is present.



1. Interrupt from interrupt controller when data transfer is needed.
2. Using IE flip-flop, CPU detects interrupt.
3. CPU branches to a respective device's ISR after enabling INTA.

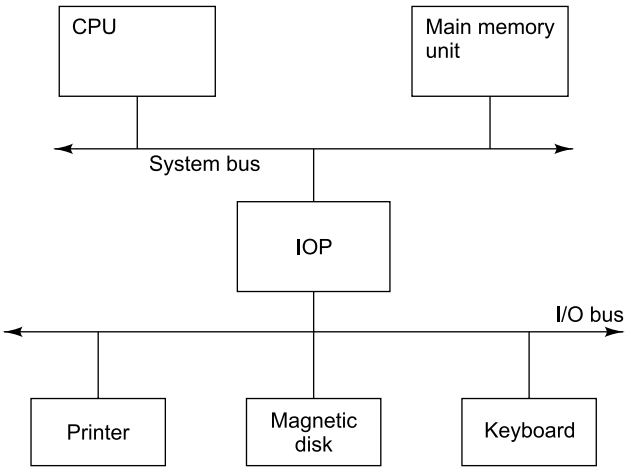
Hardware interrupt

(d) *Vector processors and their uses:* A vector processor consists of a scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operations. Scalar instructions are executed on the scalar processor, whereas vector instructions are executed on the vector unit, which is generally pipelined. When designing the instruction set for a vector unit, one has to choose an ISA (Instruction Set Architecture). Most of today's vector units have an instruction set that generalizes the Load/Store ISA (RISC architecture) of scalar processors. Vector processors are processors, which have special hardware for performing operations on vectors.

Most of the available supercomputers are vector supercomputers. A *supercomputer* is characterized by very high execution speed, large main memory and secondary memory and the use of parallel structured software in large extend. Supercomputers are specifically designed to perform huge vectors or matrix computations in the scientific areas of petroleum exploration, VLSI circuit design, meteorology, nuclear research and artificial intelligence, etc.

(e) *Architecture of IOP:* The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rise to the development of special purpose processor called IO processor (or IO channel).

The IOP is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in a typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O-related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The block diagram of an IOP is shown in figure below. The main memory unit takes the pivotal role. It communicates with processor by means of DMA.



Block diagram of a computer with IOP

2008
Computer Organization and Architecture (CS-404(EI))

This question paper is for EIE, 4th Semester and for new syllabus

Semester: 4th

Time Alloted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct answer from the given alternatives in each of the following:
   (i) Maximum number of directly addressable locations in the memory of a processor having 10-bits wide control bus, 20-bits address bus, and 8-bit data bus is
   (a) 1K (b) 2K (c) 1M (d) none of these

   Answer
   (c) 1M
   (ii) Booth's algorithm for computer arithmetic is used for
   (a) multiplication of numbers in sign magnitude form
   (b) multiplication of numbers in two's complement form
   (c) division of numbers in sign magnitude form
   (d) division of numbers in two's complement form

   Answer
   (b) multiplication of numbers in two's complement form
   (iii) The last statement of any Symbolic Mircroprogram must contain
   (a) NEXT (b) OVER (c) FETCH (d) INDRCT

   Answer
   (c) FETCH
   (iv) Virtual memory system allows the employment of
   (a) more than address space (b) the full address space
   (c) more than hard disk capacity (d) none of these

Answer

(a) more than address space

(v) In fourth generation computers, the main technology used is

(a) Transistor (b) SSI (c) MSI (d) LSI & VLSI

Answer

(d) LSI & VLSI

(vi) The numbers in the range –23 to +31 are represented by the minimum number of bits

(a) 6 (b) 8 (c) 7 (d) 5

Answer

(a) 6

(vii) Bidirectional buses use

(a) tri-state buffers

(b) two tri-state buffers in cascade

(c) two back to back connected tri-state buffers in parallel

(d) two back to back connected buffers

Answer

(c) two back to back connected tri-state buffers in parallel

(viii) CPU gets the address of next instruction to be processed from

(a) Instruction register (b) Memory address register

(c) Index register (d) Program counter

Answer

(d) Program counter

(ix) The first computer used to store a program is

(a) EDSAC (b) ENIAC (c) EDVAC (d) ACE

Answer

(b) ENIAC

(x) A machine using base register addressing method has n base registers and displacement contains k bits; programmer can access any

(a) n regions of k addresses each (b) 2^n regions of 2^k addresses each

(c) n regions of 2^k addresses each (d) none of these

Answer

(c) n regions of 2^k addresses each

Group-B
(Short-Answer Questions)

Answer *any three of* the following. $3 \times 5 = 15$

2. (a) Write the key features of von Neumann architecture of a computer and mention the bottlenecks.

(b) How does Harvard architecture differ from von Neumann architecture? 2 + 2 + 1

Answer

(a) See answer of question number 8 (C) of 2007 (CS-303).

(b) A single memory is used for both program and data storage in Von Neumann computers.

But, Harvard computers are computers with separate programs and data memories. Data memory and program memory can be of different widths, type, etc. Program and data can be fetched in one cycle, by using separate control signals—'program memory read' and 'data memory read'.

3. (a) Write + 7_{10} in IEEE 64-bit format.
   (b) Convert IEEE 32-bit format 40400000_{16} in decimal value.
   (c) Convert IEEE 64-bit format $ABCD000000000000_{16}$ in decimal value. 2 + 1.5 + 1.5

Answer

(a) The decimal number +7 = + 111 in binary = + 1.11×2^2
The 52-bit mantissa M =0.11000 00000 00000 00000 00000 00000 00000 00000 00000 0000000
The biased exponent E′ = E + 1023 = 2 + 1023 = 1025 = 100000 00001, using 11-bit.
Since the number is positive, the sign bit S = 0
Therefore, the IEEE double-precision (64-bit) representation is:

0	100000 00001	11000 00000 00000 00000 00000 00000 00000 00000 00000 0000000

(b) The number 40400000_{16} has equivalent binary representation as:
0100 0000 0100 0000 0000 0000 0000 0000
The sign of the number = 0, biased exponent value = 1000 0000 = 128. So the exponent value = 128 – 127 = 1. The mantissa field = 100 0000 0000 0000 0000 0000.
Therefore, the decimal value of the number = + $(1.1)_2 \times 2^1 = 1.5 \times 2 = 3$.

(c) The number $ABCD000000000000_{16}$ has equivalent binary representation as:
1010 1011 1100 1101 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
The sign of the number = 1, biased exponent value = 010 1011 1100 = 700. So the exponent value = 700 – 1023 = –323. The mantissa field = 1101 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000.
Therefore, the decimal value of the number = $-(1.1101)_2 \times 2^{-323} = -1.8125 \times 2^{-323}$.

4. Evaluate the arithmetic statement X = (A + B) * (C + D) in zero, one, two and three address machines. 5

Answer

To evaluate the statement X = (A + B) * (C + D) in zero, one, two and three address machines, we assume the following assumptions:
LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations—addition and multiplication respectively. Assume that the respective operands are in memory addresses A, B, C and D and the result must be stored in the memory at address X.

For zero address machine:
The assembly language program using zero-address instructions is written next. In the comment field, the symbol TOS is used, which means the top of stack.

```
PUSH  A        ; TOS ← A
PUSH  B        ; TOS ← B
ADD            ; TOS ← (A + B)
PUSH  C        ; TOS ← C
```

```
PUSH  D          ; TOS ← D
ADD              ; TOS ← (C + D)
MULT             ; TOS ← (A + B) * (C + D)
POP X            ; X ← TOS
```

For one address machine:
The assembly language program using one address instructions is written below.

```
LOAD  A          ; AC ← M[A]
ADD   B          ; AC ← AC + M[B]
STORE T          ; T ← AC
LOAD  C          ; AC ← M[C]
ADD   D          ; AC ← AC + M[D]
MULT  T          ; AC ← AC * M[T]
STORE  X         ; X ← AC
```

For two address machine:
The assembly language program using two address instructions is written below.

```
LOAD R1, A       ; R1 ← M[A]
ADD  R1, B       ; R1 ← R1 + M[B]
LOAD R2, C       ; R2 ← M[C]
ADD  R2, D       ; R2 ← R2 + M[D]
MULT  R1, R2     ; R1 ← R1 * R2
STORE X, R1      ; X ← R1
```

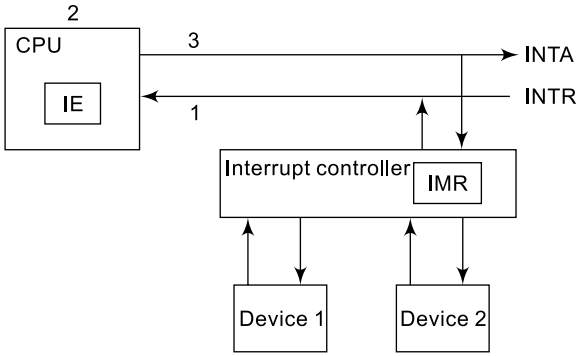
For three address machine:
The assembly language program using three address instructions is written below.

```
ADD   R1, A, B   ; R1 ← M[A] + M[B]
ADD   R2, C, D   ; R2 ← M[C] + M[D]
MULT   X, R1, R2 ; X ← R1 * R2
```

5. What are vectored interrupts? How are they used in implementing hardware interrupts? 5

Answer
In a vectored interrupt I/O method, the source device that interrupts, supplies the branch information (i.e. the starting address of interrupt service routine (ISR)) to the CPU. This information is called the *interrupt vector*, which is not any fixed memory location.

To implement interrupts, the CPU uses a signal, known as an *interrupt request (INTR)* signal to the interrupt handler or controller hardware, which is connected to each I/O device that can issue an interrupt to it. Here, interrupt controller makes liaison with the CPU on behalf of I/O devices. Typically, interrupt controller is also assigned an *interrupt acknowledge (INTA)* line that the CPU uses to signal the controller that it has received and begun to process the interrupt request by employing an ISR. The following figure below shows the hardware lines for implementing interrupts.
2



1. Interrupt from interrupt controller when data transfer is needed.
2. Using IE flip-flop, CPU detects interrupt.
3. CPU branches to a respective device's ISR after enabling INTA.

The interrupt controller uses a register called *interrupt-request mask register (IMR)* to detect any interrupt from the I/O devices. If there is n number of I/O devices in the system, then IMR is *n*-bit register and each bit indicates the status of one I/O device. Let IMR's content be denoted as E_0 E_1 E_2 ….. E_{n-1}. When $E_0 = 1$ then device 0 interrupt is recognized; When $E_1 = 1$, then device 1 interrupt is recognized and so on. The processor uses a flag bit known as *interrupt enable (IE)* in its status register (SR) to process the interrupt. When this flag bit is '1', the CPU responds to the presence of interrupt by enabling INTA line; otherwise not. When the INTA is accepted by a device, device puts its own interrupt vector address (VAD) to the data bus using interrupt controller.

6. Compare RISC with CISC.

Answer

See answer of question number 6 of 2007 (CS-404).

Group-C
(Long-Answer Questions)

Answer any *three of* the following. $3 \times 15 = 45$

7. (a) Explain Booth's algorithm. Apply Booth's algorithm to multiply the two numbers (+ 14) and (–12). Assume the multiplier and multiplicand to be of 5 bits each.
   (b) Give the flowchart for division of two binary numbers and explain. 10 + 5

Answer

(a) For Booth's algorithm, see the answer of question no. 8(a) of 2007 (CS-303).
For multiplication of two numbers (+14) and (–12), see answer of question no. 11(a) of 2007 (CS-404).

(b) See answer of question no. 11(b) of 2007 (CS-404).

8. (a) Explain the memory hierarchy pyramid, showing both primary and secondary memories in the diagram and also explain the relationship of cost, speed and capacity.

(b) Given the following, determine size of the sub-fields (in bits) in the address for direct mapping, associative and set associative mapping cache schemes.
   - We have 256 MB main memory and 1 MB cache memory.

- The address space of this processor is 256 MB.
- The block size is 128 bytes.

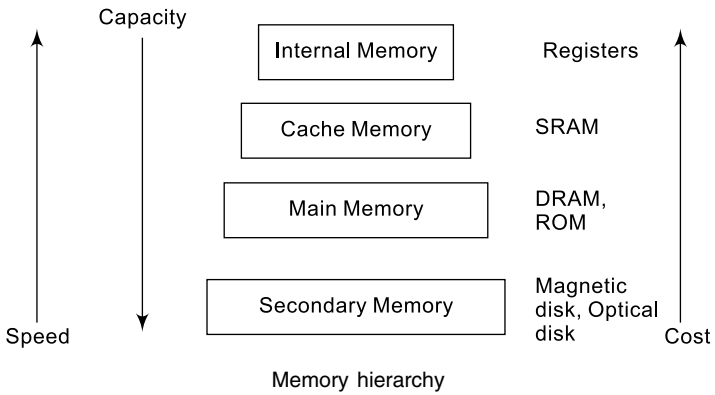
There are 8 blocks in a cache set. 5 + 10

Answer

(a) The total memory capacity of a computer can be considered as being a hierarchy of components. The memory hierarchy system consists of all storage devices used in a computer system and are broadly divided into following four groups, shown in the pyramid figure below.

- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory

Secondary memory: The slow-speed and low-cost devices that provide backup storage are called secondary memory. The most commonly used secondary memories are magnetic disks, such as hard disk, floppy disk and magnetic tapes. This type of memory is used for storing all programs and data, as this is used in bulk size. When a program not residing in main memory is needed to execute, it is transferred from secondary memory to main memory. Programs not currently needed in main memory (in other words, the programs that are not currently executed by the processor) are transferred into secondary memory to provide space for currently used programs and data.



Memory hierarchy

Main memory: This is the memory that communicates directly with CPU. Only programs and data currently needed by the CPU for execution reside in the main memory. Main memory occupies central position in hierarchy by being able to communicate directly with CPU and with secondary memory devices through an I/O processor.

Cache memory: This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than a main memory, thus the processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as a buffer between the CPU and the main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of computer.

Internal memory: This memory refers to the high-speed registers used inside the CPU. These registers hold temporary results when a computation is in progress. There is no speed disparity between these registers and the CPU because they are fabricated with the same technology. However, since registers are very expensive, only a few registers are used as internal memory.

(b) Given,

The capacity of main memory = 256 MB

The capacity of cache memory = 1MB

Block size = 128 bytes.

A set contains 8 blocks.

Since the address space of the processor is 256 MB, the processor generates address of 28-bit to access a byte (word). Because 256 MB = 2^{28}.

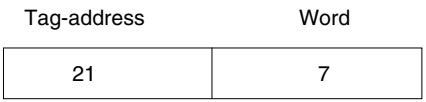
The number of blocks main memory contains = 256 MB / 128 bytes = 2^{21}.

Therefore, no. of bits required to specify one block in main memory = 21.

Since the block size is 128 bytes.

The no. of bits required to access each word (byte) = 7.

For associative cache, the address format is:

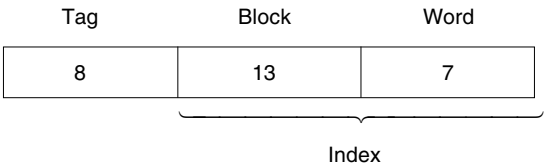


The number of blocks cache memory contains = 1 MB/128 bytes = 2^{13}.

Therefore, no. of bits required to specify one block in cache memory = 13.

The tag field of address = 28 – (13 + 7) = 8-bit.

For direct cache, the address format is:



In case of set-associative cache:

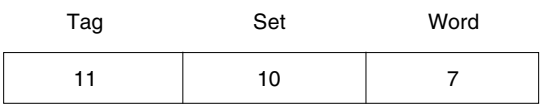
A set contains 8 blocks.

Therefore, the number of sets in cache = $2^{13}/8 = 2^{10}$.

Thus, the number of bits required to specify each set = 10.

The tag field of address = 28 – (10 + 7) = 11-bit.

For set-associative cache, the address format is:

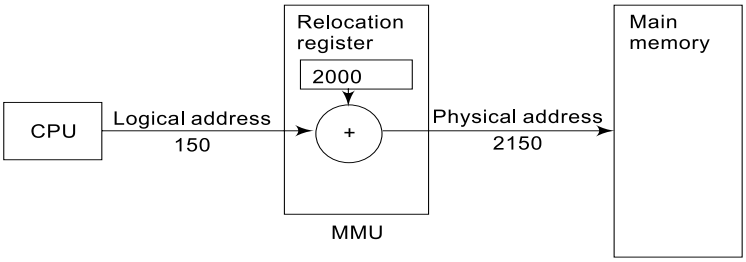


9. (a) Explain the mapping of virtual address to physical address.
   (b) Explain the reading and writing operation of a basic static RAM cell.
   (c) Why does a DRAM cell need refreshing? 5 + 5 + 5

Answer

(a) When a program needs to be executed, the CPU would generate addresses, called *logical* or *virtual* addresses. The corresponding addresses in the physical memory, as occupied by the executing program, are called *physical* addresses. The set of all logical addresses generated by the CPU or program is called *logical-address space* and the set of all physical addresses corresponding to these logical addresses is called *physical-address space*. The memory-management unit (MMU) maps each logical address to a physical address during program execution.

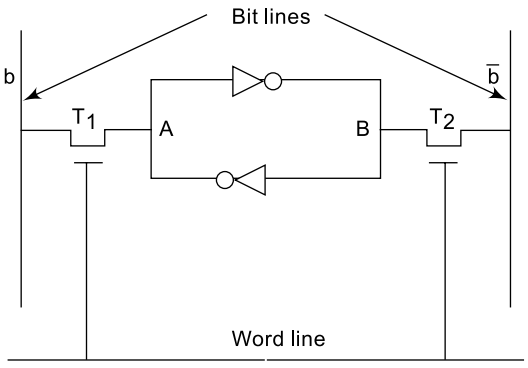
The figure below illustrates this mapping method, which uses a special register called base register or relocation register.



A memory-management scheme

The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

(b) Static memories (SRAMs) are memories that consist of circuits capable of retaining their state as long as power is applied. Thus, this type of memories are called volatile memories. The figure below shows a cell diagram of SRAM memory. A latch is formed by two inverters connected as shown in the figure. Two transistors T_1 and T_2 are used for connecting the latch with two bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch retains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point B is 0. This state is retained as long as the word line is not activated.



A SRAM cell

Read Operation: For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation: Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then, the bit value to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.

(c) Information is stored in a dynamic memory cell in the form of a charge on a capacitor. Due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitor drops below some threshold value. Thus this charge in capacitor needs to be periodically recharged or refreshed.

10. (a) What are the various modes of data transfer between computer and peripherals? Explain.
    (b) Differentiate between isolated I/O and memory mapped I/O.
    (c) Show how computer bus is organized using tri-state buffer. 5 + 5 + 5

Answer

(a) The modes of data transfer between computer and peripherals are:
1. Programmed I/O.
2. Interrupt-initiated I/O.
3. Direct memory access (DMA).

Programmed I/O: This is the software method where CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

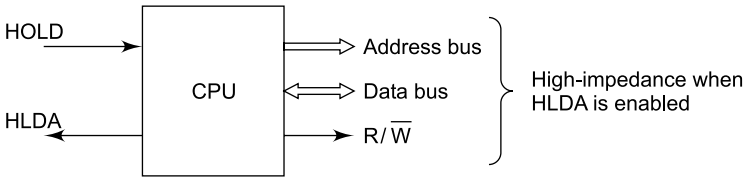
Interrupt-initiated I/O: In the programmed I/O method, the program constantly monitors the device status. Thus, the CPU stays in the program until the I/O device indicates that it is ready for data transfer. This is time-consuming process since it keeps the CPU busy needlessly. It can be avoided by letting the device controller continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to an *interrupt-service-routine (ISR) or I/O routine or interrupt handler* to process the I/O transfer, and then returns to the task it was originally performing. Thus, in the interrupt-initiated mode, the ISR software (i.e. CPU) performs data transfer but is not involved in checking whether the device is ready for data transfer or not. Therefore, the execution time of CPU can be optimized by employing it to execute normal program, when no data transfer is required.

Direct Memory Access (DMA): To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called direct memory access (DMA).

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA controller. The DMA controller performs the functions that would

normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure below shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.



CPU bus signals for DMA transfer.

(b) 1. In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

2. The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.

3. Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.

4. Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

(c) See answer of question number 10 (e) of 2007 (CS-303).

11. (a) What is meant by DMA? Why is it useful? Briefly explain, with suitable diagram, the DMA operation in association with CPU.

(b) Draw the schematic diagram for daisy chain polling arrangement in case of vectored interrupt for three devices. 2 + 2 + 4 + 7

Answer

(a) See answer of question number 8 (a) of 2007 (CS-404).

(b) See answer of question number 8 (b) of 2007 (CS-404).

2008
Computer Organization and Architecture (CS-404)

Semester: 4th

Time Allotted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following : $10 \times 1 = 10$

(i) Thrashing
- (a) reduces page I/O
- (b) decreases the degree of multiprogramming
- (c) implies excessive page I/O
- (d) none of these.

Answer
(c) implies excessive page I/O

(ii) When signed numbers are used in binary arithmetic, then which one of the following notations would have unique representation for zero?
- (a) Sign magnitude
- (b) Sign 1's complement
- (c) Sign 2's complement
- (d) None of these.

Answer
(c) Sign 2's complement

(iii) If the memory chip size is 256×1 bits, then the number of chips required to make up 1 kbytes of memory is
- (a) 32
- (b) 24
- (c) 12
- (d) 8

Answer
(a) 32

(iv) How many address bits are required for a 512×4 memory?
(a) 512 (b) 4 (c) 9 (d) $A_0 - A_6$

Answer
(c) 9

(v) What is the 2's complement representation of –20 in a 16-bit micro-computer?
(a) 0000 0000 0001 1000 (b) 1111 1111 1110 0111
(c) 1111 1111 1110 1000. (d) none of these.

Answer
(d) none of these

(vi) The technique of placing software in a ROM semiconductor chip is called
(a) PROM (b) EPROM (c) FIRMWARE (d) Micro-processor.

Answer
(c) FIRMWARE

(vii) Which one is the advantage of virtual memory?
(a) Faster access to memory on an average
(b) Process can be given protected address spaces
(c) Program larger than the physical memory size can be run
(d) None of these.

Answer
(c) Program larger than the physical memory size can be run

(viii) A page fault
(a) occurs when a program access a page memory
(b) is an error in a specific page
(c) is an access to a page not currently in memory
(d) none of these.

Answer
(c) is an access to a page not currently in memory

(ix) Convert(FAFAFA)$_{16}$ into Octal form
(a) 76767676 (b) 76575372 (c) 76737672 (d) 76727672.

Answer
(b) 76575372

(x) The logic circuitry in ALU is
(a) entirely combinational (b) entirely sequential
(c) combinational cum sequential (d) none of these.

Answer
(a) entirely combinational

Group-B
(Short-Answer questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. Explain the memory hierarchy pyramid, showing both primary and secondary memory in the diagram and also explain the relationship of cost, speed and capacity.

Answer

See answer of question number 8(a) of 2008(CS-404(EI)).

3. Discuss about the different hazards in pipelining.

Answer

Pipeline hazards: Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. The instruction is said to be *stalled*. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall.

Types of pipeline hazards are:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards

They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Solution of control hazards:

In order to cope with the adverse effects of branch instructions, an important technique called *prefetching* is used. *Prefetching technique* states that: Instruction words ahead of the one currently being decoded in the instruction-decoding (ID) stage are fetched from the memory system before the ID stage requests them.

Structural hazards

Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Example: Instruction ADD R4, X fetches operand X from memory in the OF stage at 3rd clock period. The memory doesn't accept another access during that period. For this, (i+2)th instruction cannot be initiated at 3rd clock period to fetch the instruction from memory. Thus, one clock cycle is stalled in the pipeline for all subsequent instructions. This is illustrated next.

Clock cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
ADD R4,X →	IF	ID	OF	EX	WB								
Instr. i+1		IF	ID	OF	EX	WB							
Instr. i+2			stall	IF	ID	OF	EX	WB					
Instr. i+3					IF	ID	OF	EX	WB				

Penalty: 1 cycle.

Structural hazard in instruction pipeline

Solution of structural hazards:

Certain resources are duplicated in order to avoid structural hazards. Functional units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time. A classical way to avoid hazards at memory access is by providing separate data and instruction caches.

Data hazards

Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard*.

Example: We have two instructions, I1 and I2. In a pipeline the execution of I2 can start before I1 has terminated. If in a certain stage of the pipeline, I2 needs the result produced by I1, but this result has not yet been generated, we have a data hazard.

According to various data update patterns in instruction pipeline, there are three classes of data hazards exist:

- Write After Read (WAR) hazards
- Read After Write (RAW) hazards
- Write After Write (WAW) hazards

Solution of data hazards:

The system must resolve the interlock situation when a hazard is detected. Consider the sequence of instructions {… I, I + 1, …, J, J + 1, …} in which a hazard has been detected between the current instruction J and a previous instruction I. This hazardous situation can be resolved in one of the two following ways:

- One simple solution is to stall the pipeline and to ignore the execution of instructions J, J + 1, …, down the pipeline until the instruction I has passed the point of resource conflict.
- A more advanced approach is to ignore only instruction J and continue the flow of instructions J + 1, J + 2, …, down the pipeline. However, the potential hazards due to the suspension of J must be continuously tested as instructions J + 1, J + 2, … execute prior to J. Thus, multilevel of hazard detection may be encountered, which requires much more complex control policies to resolve such multilevel of hazards.

4. Explain how a RAM of capacity 2 kbytes can be mapped into the address space $(1000)_H$ to $(17FF)_H$ of a CPU having a 16-bit address lines. Show how the address lines are decoded to generate the chip select condition for the RAM.

Answer

Since the capacity of RAM memory is 2 KB, the memory uses 11 (2 KB = 2^{11}) address lines, say namely $A_{10} - A_0$, to select one word. Thus, memory's internal address decoder uses 11 lines $A_{10} - A_0$ to select one word.

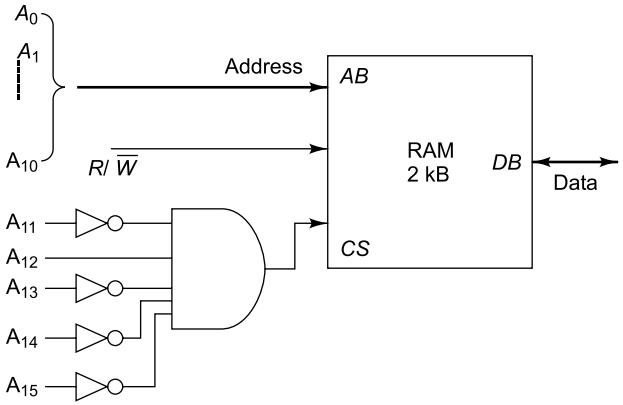
To select this memory module, remaining 5 (i.e. 16 – 11) address lines $A_{15} - A_{11}$ are used. Thus, an external decoding scheme is employed on these higher-order five address bits of processor's address.

The address space of the memory is 1000_H and $17FF_H$.

Therefore, the starting address (1000_H) in memory is as:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Based on the higher-order five bits (00010), external decoding scheme performs a logical AND operation on address values: $\overline{A_{15}}$, $\overline{A_{14}}$, $\overline{A_{13}}$, A_{12} and $\overline{A_{11}}$. The output of AND gate acts as chip select (CS) line. The address decoding scheme is shown in figure below.



5. Evaluate the following arithmetic statement using three addresses, two addresses and one address instructions:

$$X = (A+B) * (C+D)$$

Answer

See answer of question number 4 of 2008(CS-404(EI)).

6. Use 8-bit two's complement integers, perform the following computations : 2 + 2 + 1
   (i) –34 + (–12) (ii) 17 – 35 (iii) 18 – (–5).

Answer

(i) In 2's complement representation, –34 = 1101 1110
–12 = 1111 0100
Adding these two numbers, we get, 11101 0010, which is 9-bit result. By addition rule discard the 9th bit and get the result: 1101 0010, which shows that the number is negative. To get the result in its familiar form, take 2's complement of the result. The result is –46.

(ii) 17 – 35: This is subtraction and we know that 2's complement of (35) is to be added with 17. The representation of 17 is 0001 0001 and 2's complement of 35 is 1101 1101. After addition, we get, 1110 1110. This negative number and its value is –18.

(iii) 18 – (–5) = 18 + 5. The representation of 18 is 0001 0010 and representation of 5 is 0000 0101. We get after addition, 0001 0111. The result is equivalent to decimal 23.

Group – C
(Long-Answer Questions)

Answer any *three* questions. 3 × 15 = 45

7. (a) Multiply +14 and –13 using Booth's sequential method of multiplication. Draw the corresponding circuit block diagram.
   (b) Multiply +12 and –11 using modified Booth's sequential method of multiplication. Draw the corresponding circuit block diagram. (4 + 4) + (4 + 3)

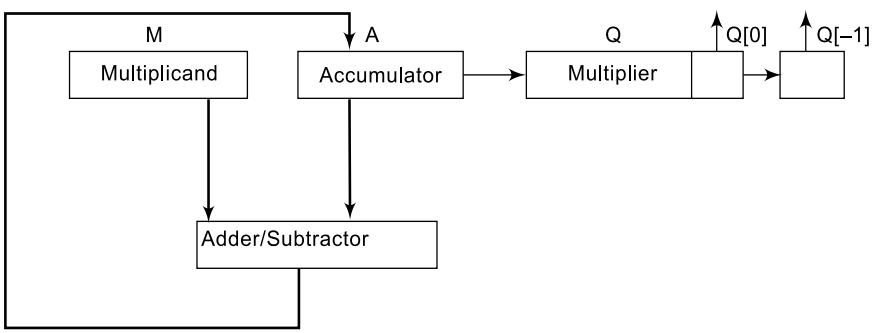
Answer

(a) Multiplication of numbers + 14 and –13:
Multiplicand, M = + 14 = 01110 and multiplier, Q = –13 = 10011.

	M	A	Q	Size
Initial Configuration	01110	00000	10011 0	5
Step-1 As Q[0]=1and Q[–1]=0				
A + A – M	01110	10010	10011 0	—
ARS(AQ)	01110	11001	01001 1	4
Step-2 As Q[0]=1 and Q[–1]=1				
ARS(AQ)	01110	11100	10100 1	3
Step-3 As Q[0] = 0 and Q[–1] = 1				
A = A + M	01110	01010	10100 1	–
ARS(AQ)	01110	00101	01010 0	2
Step-4 As Q[0] = 0 and Q[–1] = 0				
ARS(AQ)	01110	00010	10101 0	1
Step -5 As Q[0] = 1and Q[–1] = 0				
A = A – M	01110	10100	10101 0	–
ARS (AQ)	01110	11010	01010 1	0

Since, the size register becomes 0, the algorithm is terminated and the product is = AQ = 11010 01010, which shows that the product is a negative number. To get the result in familiar form, take the 2's complement of the magnitude of the number and the result is –182 in decimal.

The circuit block diagram of the Booth's sequential multiplication algorithm is shown below.



(b) A faster version of Booth's multiplication algorithm for signed numbers, known as the *modified Booth's algorithm*, examines three adjacent bits Q[i + 1] Q[i] Q[i – 1] of the multiplier Q at a time, instead of two. Apart from three basic actions performed by original Booth's algorithm, which can be expressed as: add 0, $1 \times M$ (multiplicand) and $\bar{1} \times M$ to A (the accumulated partial products), this modified algorithm performs two more actions: add $2 \times M$ and $\bar{2} \times M$ to A. These have the effect of increasing the radix from 2 to 4 and allow an $N \times N$ multiplication requiring only N/2 partial products.

The following table lists the multiplicand selection decisions for all possibilities.

Q[i + 1]	*Q[i]*	*Q[i – 1]*	*Multiplicand selected at position i*
0	0	0	$0 \times M$
0	0	1	$1 \times M$
0	1	0	$1 \times M$
0	1	1	$2 \times M$
1	0	0	$\bar{2} \times M$
1	0	1	$\bar{1} \times M$
1	1	0	$\bar{1} \times M$
1	1	1	$0 \times M$

For the multiplication of two numbers +12 and –11:

Operands	Values	i	Q[i + 1] Q[i] Q[i – 1]	Action
Multiplicand M = +12	01100			
Multiplier Q = –11	10101			

After including extended sign bit and implied 0 to right of lsb, multiplier = 1101010

P0	00000 01100	0	010	Add $1 \times M$ to A
P2	00001 100xx	2	010	Add $1 \times M$ to A
P4	11010 0xxxx	4	110	Add $\bar{1} \times M$ to A

Product = 11011 11100 = P0 + P2 + P4. This result shows that it is equivalent to –132.

8. (a) Show the steps for Restoring and Non-restoring method of division when 9 is divided by 2 using 4-bit representation. Draw the block diagram and explain.
   (b) Design the Control Unit for the Restoring Division approach. 3 + 2 + 3 + 7

Answer

(a) Division of 9 by 2 using *restoring* division method:

Dividend Q = 9 = 1001 and divisor M = 2 = 0010.

	M	A	Q	Size
Initial Configuration	00010	00000	1001	4
Step-1				
LS(AQ)	00010	00001	001–	—
A=A – M	00010	11111	001–	—
As Sign of A= –ve				
Set Q[0]=0				
& Restore A	00010	00001	0010	3

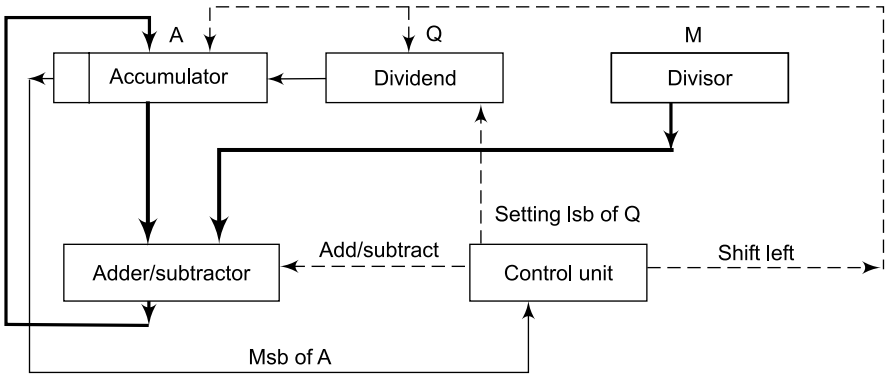
Step-2				
LS(AQ)	00010	00010	010–	—
A=A – M	00010	00000	010–	—
As Sign of A= +ve				
Set Q[0]=1	00010	00000	0101	2
Step-3				
LS(AQ)	00010	00000	101–	—
A=A – M	00010	11110	101–	—
As Sign of A= –ve				
Set Q[0]=0				
Restore A	00010	00000	1010	1
Step-4				
LS(AQ)	00010	00001	010–	—
A=A – M	00010	11111	010–	—
As Sign of A= –ve				
Set Q[0]=0				
Restore A	00010	00001	0100	0

From the above result, we see that the quotient = Q = 0100 = 4 and remainder = A = 00001 = 1.
Division of 9 by 2 using *non-restoring* division method:
Dividend Q = 9 = 1001 and divisor M = 2 = 0010.

	M	A	Q	Size
Initial Configuration	00010	00000	1001	4
Step-1				
As Sign of A= +ve				
LS(AQ)	00010	00001	001–	—
A=A – M	00010	11111	001–	—
As sign of A= –ve				
Set Q[0]=0	00010	11111	0010	3
Step-2				
As sign of A= –ve				
LS(AQ)	00010	11110	010–	—
A=A + M	00010	00000	010–	—
As sign of A= +ve				
Set Q[0]=1	00010	00000	0101	2
Step-3				
As sign of A = +ve				
LS(AQ)	00010	00000	101–	—
A=A – M	00010	11110	101–	—
As sign of A= –ve				
Set Q[0]=0	00010	11110	1010	1

Step-4				
As sign of A= –ve				
LS(AQ)	00010	11101	010–	—
A=A + M	00010	11111	010–	—
As sign of A= –ve				
Set Q[0]=0				
Restore A	00010	00001	0100	0
(i.e. A= A+M)				

From the above last step, we conclude that quotient = 0100 = 4 and remainder = 00001 = 1. The block diagram of the restoring division method is shown below.



(b) The microprogrammed control unit for restoring divison method:

The symbolic microprogram for n × n restoring divison is as follows:

Control Memory Address		*Control word*
0	START	A ← 0, M ← Inbus, L ← 4
1		Q ← Inbus
2	LOOP	LS (AQ)
3		A ← A – M
4		If A[n] = 0 then goto ONE
5		Q[0] ← 0, A ← A + M
6	ONE	Q[0] ← 1
7		L ← L –1
8		If Z = 0 then go to LOOP
9		Outbus ← A;
10		Outbus ← Q;
11	HALT	Go to HALT;

In this task, two conditions, sign of A, i.e. A[n] = 0 and Z = 0, are tested. Here, Z corresponds to the L register. When L ≠ 0, Z is reset to 0, otherwise Z is set to 1. These two conditions are applied as inputs to the condition select MUX. Additionally, to take care of no-branch and unconditional-branch situations, a logic 0 and logic 1 are applied as data inputs to this MUX, respectively. Therefore, The MUX is able to handle four data inputs and thus must be at least an 4:1. The size of the condition select field must be 2 bits in length.

With this design, the condition select field may be interpreted as below:

Condition select	*Action taken*
00	No branching
01	Branch if A[n] =0
10	Branch if Z =0
11	Unconditional branching

With these details, the size of the control word is calculated as follows:

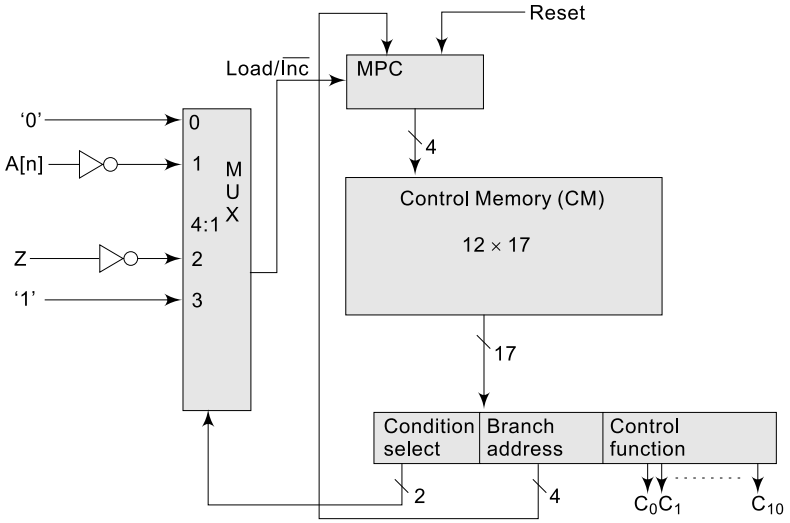
Size of a control word = size of the condition select field + size of the branch address field + number of control functions

= 2 + 4 + 11

= 17 bits.

Hence, the sizes of the CMDB and CM are 17 bits and 12 × 17, respectively. The complete hardware organization of the control unit and control signals is shown in the next figure.

C_0 : A ← 0
C_1 : M ← Inbus
C_2 : L ← 4
C_3 : Q ← Inbus
C_4 : LS (AQ)
C_5 : F ← 1 – r
C'_5 : F ← 1 + r
C_6 : A ← F
C_7 : Q[0] ← 1
C'_7 : Q[0] ← 0
C_8 : L ← L – 1
C_9 : Outbus ← A
C_{10} : Outbus ← Q



Microprogrammed n × n restoring divider control unit

Finally, the generation of binary microprogram stored in the CM is discussed. There exists a control word for each line of the symbolic program listing. For example, consider the first line (0^{th}) of the symbolic listing program mentioned previously. This instruction, being a simple load instruction, introduces no branching. Therefore, the condition-select field should be 00. Thus, the contents of the branch address field are irrelevant. However, without any loss of generality, the contents of this field can be reset to 0000. For this instruction, three micro-operations C_0, C_1 and C_2 are activated. Therefore, only the corresponding bit positions in the control function fields are set to 1. This results in the following binary microinstruction:

Condition select	*Branch address*	*Control function*
00	0000	11100000000

Continuing in this way, the complete binary microprogram for n × n restoring divider can be produced, as in the following table.

Control Memory address		*Condition select (2-bit)*	*Branch address (4-bit)*	*Control function (11-bit)*
In decimal	*In binary*			$C_0C_1.....C_{10}$
0	0000	00	0000	11100000000
1	0001	00	0000	00010000000
2	0010	00	0000	00001000000
3	0011	00	0000	00000110000
4	0100	01	0110	00000000000
5	0101	00	0000	00000010000
6	0110	00	0000	00000001000
7	0111	00	0000	00000000100
8	1000	10	0010	00000000000
9	1001	00	0000	00000000010
10	1010	00	0000	00000000001
11	1011	11	1011	00000000000

9. (a) What is Cache memory ? Why is it needed ?
Explain the Write-through and Write-back mechanism.
Why is set-associative mapping technique more advantageous than direct or associative mapping technique?
A computer has 512 KB cache memory and 2 MB main memory. If the block size is 64 bytes, then find out the subfields for
(i) direct mapped cache
(ii) associative
(iii) 8-way set associative cache.
(b) Why memory hierarchy is needed?
What are the different levels in memory hierarchy? 11 + 4

Answer

(a) This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as a buffer between the CPU and main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of computer.

There are two methods in writing into cache memory:

Write-Through Policy: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus, the main memory always contains the same data as the cache. But it is a slow process, since each time main memory needs to be accessed.

Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified or dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set. The philosophy of this method is based on the fact that during a write operation, the word residing in cache may be accessed several times (temporal locality of reference). This method reduces the number of references to main memory. However, this method may encounter the problem of inconsistency due to two different copies of the same data, one in cache and other in main memory.

The set-associative mapping is a combination of the direct- and associative-mapping techniques. Blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of a specific set. Hence, the conflict problem of the direct method is eased by having a few choices for block placement. At the same time, the hardware cost is reduced by decreasing the size of the associative search. That is why the set-associative mapping technique is more advantageous than direct- or associative- mapping technique.
Given,

The size of cache = 512 KB
The size of main memory = 2 MB
The block size = 64 bytes.
The main memory capacity = 2 MB = 2^{21} bytes.
So, the processor generates 21-bit address to access each word in cache memory.
The cache memory size = 512 KB = 2^{19} bytes and each block size = 64 bytes = 2^6 bytes.
Therefore, the number of blocks in cache memory = 512 KB/64 bytes = 2^{13}.
So, the block field size = 13-bit.
Since each block size = 64 bytes, the sector (or word) field size = 6-bit.
Thus, the tag field size = 21 – (13 + 6) = 2-bit.
Therefore the address format in *direct-mapped* cache is as follows:

2	13	6
Tag	Block	Sector (or Word)

The number of blocks main memory contains = 2 MB/64 bytes = 2^{15}.
Therefore, no. of bits required to specify one block in main memory = 15.
Since, the block size is 64 bytes.
The no. of bits required to access each word (byte) = 6.
For associative cache, the address format is:

Tag-address	Word
15	6

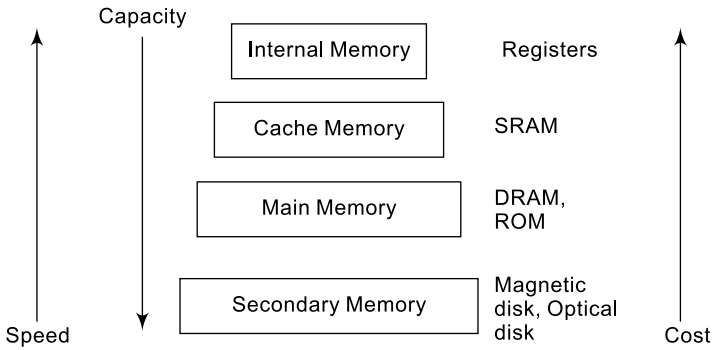
In case of 8-way set-associative cache:
Each set contains 8 blocks.
Therefore, the number of sets in cache = $2^{13}/8 = 2^{10}$.
Thus, the number of bits required to specify each set = 10.
The tag field of address = (21 – (10 + 6) = 5-bit.
For 8-way set-associative cache, the address format is:

Tag	Set	Word
5	10	6

(b) Ideally, we would like to have the memory which would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three requirements simultaneously. If we increase the speed and capacity, then cost will increase. We can achieve these goals at optimum level by using several types of memories, which collectively give a memory hierarchy.

A memory hierarchy system is broadly divided into following four groups, shown in figure below.

- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory



10. (a) If two *n* bit numbers are added then the result will be maximum how bit long?
    (b) If two *n* bit numbers are multiplied then the result will be maximum how bit long?
    (c) Design a 4-bit Arithmetic unit using multiplexers and full adders.
    (d) Two 4-bit unsigned numbers are to be multiplied using the principle of carry save adders. Assume the numbers to be $A_3\ A_2\ A_1\ A_0$ and $B_3\ B_2\ B_1\ B_0$. Show the arrangement and interconnection of the adders and the input signals so as to generate an eight bit product as $P_7\ P_6\ P_5\ P_4\ P_3\ P_2\ P_1\ P_0$. 1 + 1 + 6 + 7

Answer

(a) If two *n* bit numbers are added then the result will be maximum (n + 1)-bit long.

(b) If two *n* bit numbers are multiplied then the result will be maximum (n + n)-bit i.e. 2n-bit long.

(c) The diagram of a 4-bit arithmetic circuit is shown in figure below. The circuit has a 4-bit parallel adder and four multiplexers for 4-bit arithmetic unit. There are two 4-bit inputs A and B, and the 5-bit output is K. The size of each multiplexer is 4:1. The two common selection lines for all four multiplexers are S_0 and S_1. C_{in} is the carry input of the parallel adder and the carry out is C_{out}. The four inputs to each multiplexer are B- value, $\overline{B}$ -value, logic-0 and logic-1.

The output of the circuit is calculated from the following arithmetic sum:

$$K = A + Y + C_{in}$$

where A is a 4-bit number, Y is the 4-bit output of multiplexers and C_{in} is the carry input bit to the parallel adder. By this circuit it is possible to get 8 arithmetic micro-operations, as listed in the Table below.

Case 1: When $S_1\ S_0 = 00$.

In this case, the values of B are selected to the Y inputs of the adder. If $C_{in} = 0$, output K = A + B. If $C_{in} = 1$, output K = A + B + 1. In both cases the micro-operation addition is performed without carry or with the carry input.

Case 2: When $S_1\ S_0 = 01$.

The complements of B are selected to the Y inputs of the adder. If $C_{in} = 0$, output $K = A + \overline{B}$. This means the operation is subtraction with borrow. If $C_{in} = 1$, output $K = A + \overline{B} + 1$, which is equivalent to A + 2's complement of B. Thus this gives the subtraction A – B.

Case 3: When $S_1\ S_0 = 10$.

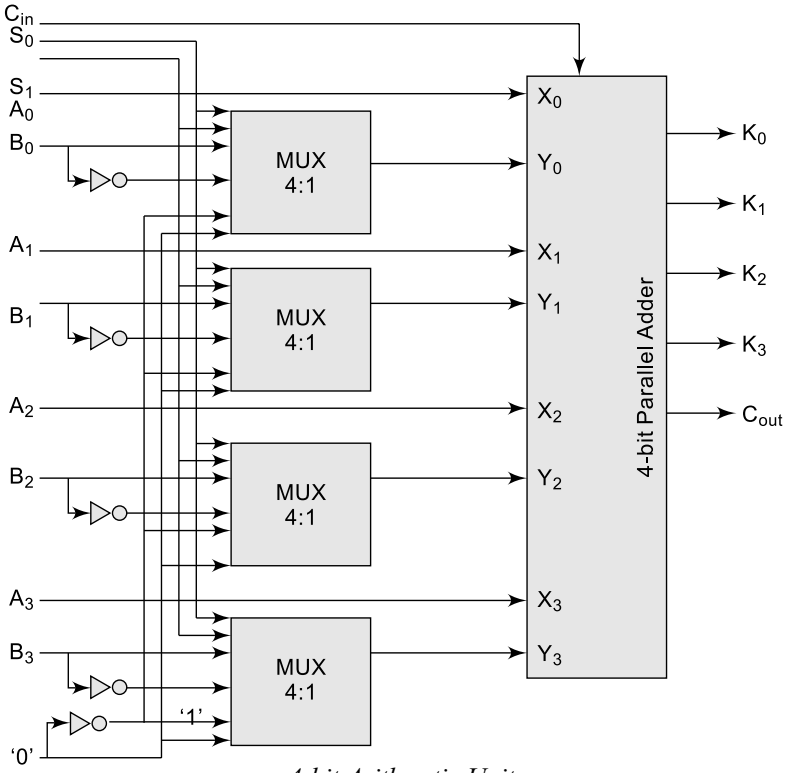
Here, all 1s are selected to the Y inputs of the adder. This means Y = (1111), which is equivalent to 2's complement of decimal 1, that means, Y = –1. If $C_{in} = 0$, the output K = A – 1,

which is a decrement operation. If $C_{in} = 1$, the output K = A – 1 + 1 = A. This causes the direct transfer of A to K.

Case 4: When $S_1\ S_0 = 11$.
In this case, all 0s are selected to the Y inputs of the adder. If $C_{in} = 0$, the output K = A, which is a transfer operation. If $C_{in} = 1$, output K = A + 1. This means the value of A is incremented by 1. Observe that only seven different arithmetic micro-operations are deduced, because the transfer operation is generated twice.

Table Arithmetic unit function table

S_1	S_0	C_{in}	Y	$K = A + Y + C_{in}$	Operation
0	0	0	B	K = A + B	Addition
0	0	1	B	K = A + B + 1	Addition with carry
0	1	0	$\overline{B}$	$K = A + \overline{B}$	Subtraction with borrow
0	1	1	$\overline{B}$	$K = A + \overline{B} + 1$	Subtraction
1	0	0	1	K = A – 1	Decrement
1	0	1	1	K = A	Transfer
1	1	0	0	K = A	Transfer
1	1	1	0	K = A + 1	Increment



4-bit Arithmetic Unit

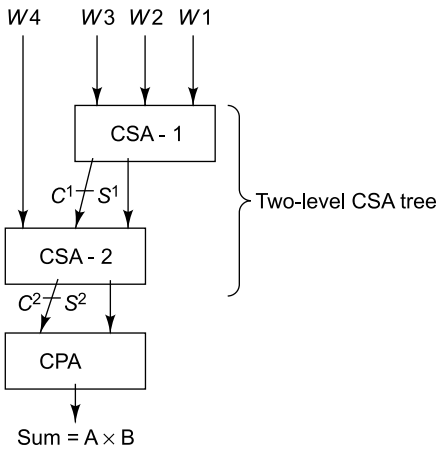
(d) The multiplication of two unsigned is done by repeated add-shift operations. Add-shift multiplication of two 4-bit numbers is illustrated in figure below.

					A3	A2	A1	A0 = A
					B3	B2	B1	B0 = B
				A3B0	A2B0	A1B0	A0B0	= W1
			A3B1	A2B1	A1B1	A0B1		= W2
		A3B2	A2B2	A1B2	A0B2			= W3
	A3B3	A2B3	A1B3	A0B3				= W4
P7	P6	P5	P4	P3	P2	P1	P0	= A × B = Product

Add-shift multiplication of two 4-bit numbers (A × B = Product)

The additions of partial products W1, W2, W3 and W4, which are generated using bit-wise AND logic operations, can be done using CSA-tree as shown in figure below to realize the multiplier for 4-bit numbers.

The first carry-save adder (CSA-1) adds W1, W2 and W3 and produces a sum vector (S^1) and a carry vector (C^1). The sum vector, the shifted carry vector and the fourth partial product W4 are applied as the inputs to the second CSA. The results produced by the second CSA are then added by a CPA to generate the final summation Sum.



11. (a) Explain the basic Direct Memory Access (DMA) operation for transfer of data bytes between memory and peripheral.
    (b) What is programmed I/O technique? Why is it not very useful?
    (c) What are the different types of interrupt? Give example.
    (d) Give the main reason why DMA based I/O is better in some circumstances than interrupt driven I/O. 6 + 3 + 3 + 3

Answer

(a) See answer of question no. 9 (a) of 2007 (CS-303).
(b) See answer of question no. 9 (c) of 2007 (CS-303).

(c) Types of interrupt:
There are basically three types of interrupts: external, internal or trap and software interrupts.

External interrupt: These are initiated through the processors' interrupt pins by external devices. Examples include interrupts by input-output devices and console switches. External interrupts can be divided into two types: maskable and non-maskable.

Maskable interrupt: The user program can enable or disable all or a few device interrupts by executing instructions EI or DI.

Non-maskable interrupt: The user program cannot disable it by any instruction. Some common examples are: hardware error and power fail interrupt. This type of interrupt has higher priority than maskable interrupts.

Internal interrupt: This type of interrupts is activated internally by exceptional conditions. The interrupts caused due to overflow, division by zero and execution of an illegal op-code are common examples of this category.

Software interrupt: A software interrupt is initiated by executing an instruction like *INT n* in a program, where n refers to the starting address of a procedure in program. This type of interrupts is used to call operating system. The software interrupt instructions allow use to switch from user to supervisor mode.

(d) See answer of question no. 9 (b) of 2007 (CS-303).

2008
Computer Organization
(CS-303)

Semester: 3rd

Time Alloted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following :
   (i) When signed numbers are used in binary arithmetic then which of the following notations would have unique representation for zero?
   (a) Sign magnitude (b) 1's complement
   (c) 2's complement (d) None of these.

   Answer
   (c) 2's complement

   (ii) The logic circuitry in ALU is
   (a) entirely combinational (b) entirely sequential
   (c) combinational cum sequential (d) none of these.

   Answer
   (a) entirely combinational

   (iii) In a micro-processor, the address of the next instruction to be executed is stored in
   (a) stack pointer (b) address latch
   (c) program counter (d) general purpose register.

   Answer
   (c) program counter

   (iv) The technique of placing software in a ROM semiconductor chip is called
   (a) PROM (b) EPROM (c) FIRMWARE (d) Micro- processor.

   Answer
   (c) FIRMWARE

(v) Cache memory
(a) increases performance (b) increases machine cycle
(c) reduces performance (d) none of these.

Answer
(a) increases performance

(vi) Associative memory is a
(a) very cheap memory (b) pointer addressable
(c) content addressable memory (d) slow memory

Answer
(c) content addressable memory

(vii) A single bus structure is primarily found in
(a) mainframe computers (b) super computers
(c) high-performance machines (d) mini and micro-computers

Answer
(d) mini and micro-computers

(viii) Memory mapped I/O scheme used for the allocation of address to memories and I/O devices is used for
(a) small systems (b) large systems
(c) large and small systems (d) very large systems

Answer
(c) large and small systems

(ix) The conversion of $(FAFAFA)_{16}$ into octal form is
(a) 76767676 (b) 76575372 (c) 76737672 (d) 76727672.

Answer
(b) 76575372

(x) Which of the following addressing modes is used in the instruction PUSH B?
(a) Immediate (b) Register (c) Direct (d) register indirect

Answer
(a) Immediate

Group-B
(Short-Answer Questions)

Answer any three of the following. $3 \times 5 = 15$

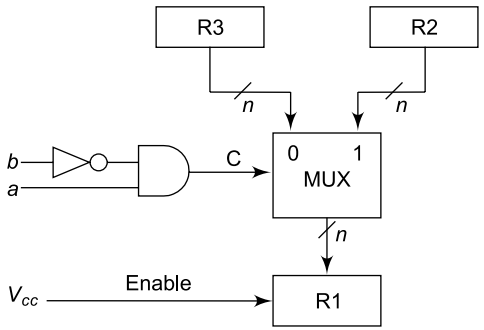
2. Show the circuit diagram for implementing the following register transfer operation. If ($a\bar{b} = 1$) then R1 ← R2 else R1 ← R3, where a and b are control variables.

Answer
Here, depending on the a and b values, n-bit content of R2 or R3 register is copied to R1 register. Such a selective register transfer micro-operation can be expressed as follows:

C: R1 ← R2
C′: R1 ← R3 [C′ indicates complement of C].

where $C = a \wedge b'$ and $C' = (a \wedge b')' = a' \vee b$
A hardware implementation for this transfer is shown in the figure below.

The R2 register is selected by the MUX if condition C = 1; otherwise register R3 is selected as source register.



Hardware implementation of "if ($a\bar{b}$ = 1) then R1 ← R2 else R1 ← R3".

3. What do you mean by instruction cycle, machine cycles and T states?

Answer

See answer of question no. 4 of 2007 (CS-404).

4. What is virtual memory ? Why is it called virtual? Write the advantage of virtual memory. 2 + 1 + 2

Answer

See answer of question no. 2 of 2007 (CS-404).

5. What are the advantages of microprogramming control over hardwired control? What is the role of an operating system? 3+2

Answer

The main advantage of microprogramming is that it provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions). With micro-programming, many additions and changes are made by simply changing the microprogram in the control memory (ROM), whereas, a small change in the hardwired approach may lead to redesigning the entire system.

The main roles of an operating system:

1. Managing the user's programs
2. Managing the memories of the computer
3. Managing the I/O operations
4. Controlling the security of a computer

6. What are the different types of interrupt ? Give examples. What is programmed I/O technique? 3 +2

Answer

Types of interrupt

There are basically three types of interrupts: external, internal or trap and software interrupts.

External interrupt: These are initiated through the processors' interrupt pins by external devices.

Examples include interrupts by input–output devices and console switches. External interrupts can be divided into two types: maskable and non-maskable.

Maskable interrupts: The user program can enable or disable all or a few device interrupts by executing instructions EI or DI.

Non-maskable interrupts: The user program cannot disable it by any instruction. Some common examples are hardware error and power fail interrupt. This type of interrupt has higher priority than maskable interrupts.

Internal interrupt: This type of interrupt is activated internally by exceptional conditions. The interrupts caused due to overflow, division by zero and execution of an illegal op-code are common examples of this category.

Software interrupts: A software interrupt is initiated by executing an instruction like INT *n* in a program, where *n* refers to the starting address of a procedure in program. This type of interrupt is used to call an operating system. The software interrupt instructions allow us to switch from user to supervisor mode.

Programmed I/O: This is the software method where the CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. In other words, the CPU polls the devices for next data transfer. This is why the programmed I/O is sometimes called *polled I/O*. Through the mid-1990s, programmed I/O was the only way that most systems ever accessed IDE/ATA hard disks.

Group-C
(Long-Answer Questions)

Answer any *three of* the following. $3 \times 15 = 45$

7. (a) What is the Von Neumann concept and its bottleneck?
   (b) Represent the decimal value - 7.5 in IEEE - 754 single precision floating- point format.
   (c) Compare parallel adder with serial adder.
   (d) What is the necessity of guard bits?
   (e) Explain and draw 4-bit binary decrementer circuit. $4 + 3 + 4 + 1 + 3$

Answer

(a) See answer of question no. 8(c) of 2007 (CS-303).

(b) The decimal number – 7.5 = - 111.1 in binary = - 1.111 x 2^2
The 23-bit mantissa M = 0.111000 000000 000000 00000
The biased exponent E′ = E + 127 = 2 + 127 = 129 = 1000 0001
Since the number is negative, the sign bit S = 1
Therefore, the IEEE single-precision (32-bit) representation is

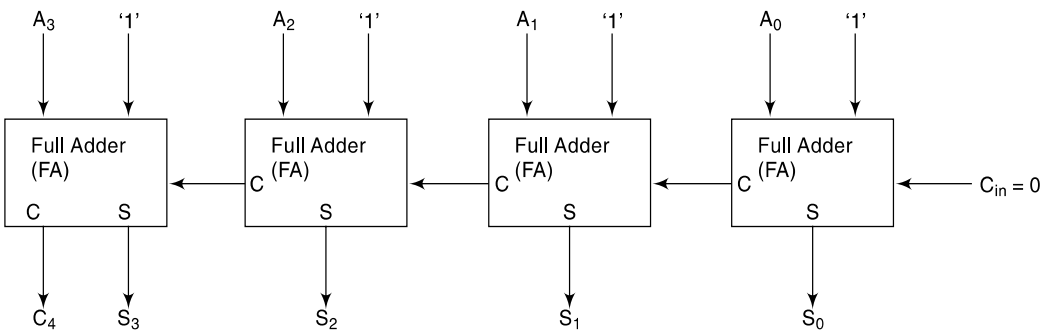
1	1000 0001	111000 000000 000000 00000

(c)

Parallel Adder	*Serial Adder*
1. This adder is a combinational circuit, which adds all bits of two numbers in one clock cycle.	1. This adder is a sequential circuit, which performs the addition of two binary numbers serially bit by bit starting with lsb.
2. This adder, being a combinational circuit, is faster than a serial adder. In one clock period all bits of two numbers are added.	2. The serial adder is very slow since it takes *n* clock cycles for addition of *n*-bit numbers.
3. The hardware cost is more than that of serial adder because, number of adder blocks needed is equal to the number of bits in operands.	3. The serial adder circuit is small and hence, it is very inexpensive irrespective of the number of bits to be added.

(d) When the mantissa is shifted right, some bits at the rightmost position (least significant position) are lost. In order to obtain maximum accuracy of the final result; one or more extra bit known as *guard bits*, are included in the intermediate steps. These bits temporarily contain the recently shifted out bits from the rightmost side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are not stored. However, based on the guard bits, the value of the mantissa can be made more precise by the rounding technique.

(e) The binary decrementer unit performs the decrement micro-operation. The decrement micro-operation subtracts value one from the number stored in a register. For example, if a 4-bit register has a binary value 1001, it will be 1000 after the decrement operation. The subtraction can easily be implemented using combinational circuit half-subtractors or sequential circuit binary down counter. The decrement micro-operation can be realized with combinational circuit full adders. The subtraction of two binary numbers can be performed by taking the 2's complement of the subtrahend and then adding it to the minuend. The diagram of a 4-bit combinational decrementer circuit has been implemented using full adders, shown in the figure below.



4-bit Decrementer Circuit

Here, we are adding a bit 1 as one of the inputs to the full adder. This means that binary number (1111) is added with the operand number A. The binary number (1111) means –1 in decimal, since the negative number is represented in computers using signed 2's complement method. That means, we are adding –1 with the operand value stored in register A.

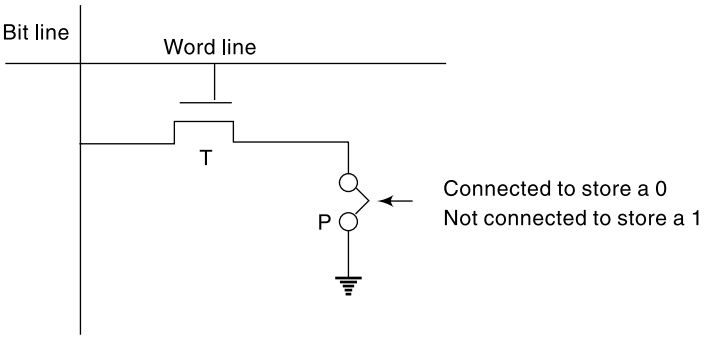
8. (a) Draw the internal cell diagram of PROM and explain its functionality.
   (b) What is cache memory ? How does it increase the performance of a computer ? What is hit ratio?

(c) A three-level memory system having cache access time of 5 nsec and disk access time of 40 nsec, has a cache hit ratio of 0·96 and main memory hit ratio of 0.9. What should be the main memory access time to achieve an overall access time of 16 nsec?

(d) Define : (i) rotational latency (ii) seek time. 4+4+5+2

Answer

(a) A PROM cell structure is shown in the figure below. A logic value 1 is stored in the cell if the transistor is not connected to the ground at point P; otherwise, a binary 0 is stored. The bit line is connected through a resistor to the power supply. Inserting a fuse at point P in the figure achieves programmability. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses of cells at these locations using high-voltage currents. The PROM's cells are once programmable, i.e., the user can store the desired bits in the cells only once and these bits cannot be altered.



A PROM memory cell

In order to read the state of the cell, the word line is activated to close the transistor, which acts as a switch.

(b) This is a special high-speed main memory, used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than the main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as a buffer between the CPU and main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of a computer.

The performance of the cache memory is measured in terms of a quantity called *hit ratio*. When the CPU refers to memory and finds the word in cache, it is said that a hit occurred. If the word is not found in the cache, then the CPU refers to the main memory for the desired word and it is refereed to as a miss to cache. The hit ratio (h) is defined below:

$$\text{Hit ratio } (h) = \frac{\text{number of hits}}{\text{total CPU references to memory}}$$

$$= \frac{\text{number of hits}}{\text{number of hits} + \text{number of misses}}$$

Thus, the hit ratio is nothing but a probability of getting hits out of some number of memory references made by CPU. So its range is $0 \le h \le 1$.

(c) Given,

Cache access time, t_c = 5 ns

Disk access time, t_s = 40 ns

Cache hit ratio, h_c = 0.96

Main memory hit ratio h_m = 0.9

Effective access time, t_{av} = 16 ns

We have to find out the main memory access time, t_m

We know,

$$t_{av} = h_c t_c + (1 - h_c)\, h_m\, (t_c + t_m) + (1 - h_c)\, (1 - h_m)\, (t_c + t_m + t_s)$$

i.e., $16 = 0.96 * 5 + 0.04 * 0.9 * (5 + t_m) + 0.04 * 0.1 * (5 + t_m + 40)$

i.e., $t_m = 271$

Therefore, required main memory access time is 271 ns.

(d) To access data in disk, the read-write head must be placed on the proper track based on the given cylinder address. The time required to position the read-write head over the desired track is known as the *seek time*, t_s. This depends on the initial position of the head relative to the specified track or cylinder address. Seeking the required track is the most time-consuming operation because it involves moving the read-write head arm. After positioning the read-write head on the desired track, the disk controller has to wait until the desired sector is under the read-write head. This waiting time is known as *rotational latency*, t_l. Rotational latency depends on the rotation speed of the disk. The *access time* of the disk is the sum of t_s and t_l.

9. (a) What is instruction cycle ? Draw the time diagram for memory write operation.
   (b) Explain the basic DMA operations for transfer of data between memory and peripherals.
   (c) Evaluate the arithmetic statement $X = (A * B)/(C + D)$ in one, two and three address machines. 1 + 4 + 5 + 5

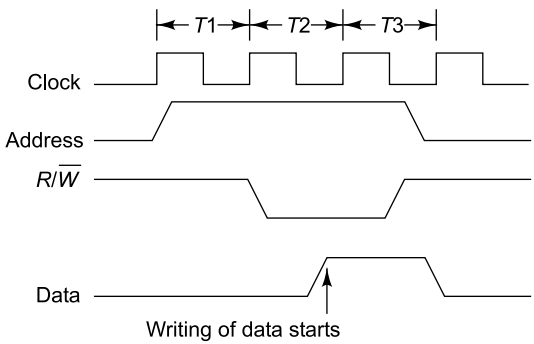
Answer

(a) For instruction cycle, see answer of question no. 4 of 2007 (CS-404).

For time diagram of memory write,

The timing diagram for memory write is shown in the next figure.

Similar to the memory read, to write a memory word, the address is specified on the address bus to select the word among many available words at first clock period T1. At second clock period T2, memory write signal is activated and after seeing the write signal activated, memory stores data in it from the data bus. The total operation needs three clock periods.



Timing diagram for memory write

(b) See answer of question no. 9(a) of 2007 (CS-303).

(c) We have to evaluate the arithmetic statement

$$X = (A * B)/(C + D)$$

using zero, one, two or three address instructions. For this, LOAD symbolic op-code is used for transferring data to the register from memory. STORE symbolic op-code is used for transferring data to the memory from register. The symbolic op-codes ADD, MULT and DIV are used for the arithmetic operations addition, multiplication and division respectively. Assume that the respective operands are in memory addresses A, B, C and D and the result must be stored in the memory at address *X*.

Using three-address instructions, the program code in assembly language is as follows:

```
MULT R1, A, B       ; R1 ← M[A] * M[B]
ADD R2, C, D        ; R2 ← M[C] + M[D]
DIV X, R1, R2       ; X ← R1/R2
```

Using two-address instructions, the program code in assembly language is as follows:

```
LOAD R1, A          ; R1 ← M[A]
MULT R1, B          ; R1 ← R1 * M[B]
LOAD R2, C          ; R2 ← M[C]
ADD R2, D           ; R2 ← R2 + M[D]
DIV R1, R2          ; R1 ← R1/R2
STORE X, R1         ; X ← R1
```

Using one-address instructions, the program code in assembly language is as follows:

```
LOAD C              ; AC ← M[C]
ADD D               ; AC ← AC + M[D]
STORE T             ; T ← AC
LOAD A              ; AC ← M[A]
MULT B              ; AC ← AC * M[B]
DIV T               ; AC ← AC / M[T]
STORE X             ; X ← AC
```

10. (a) Given the following, determine the size of the sub-fields in the address for direct mapping, associative mapping and set-associative mapping cache schemes:

Main memory size	512 MB
Cache memory size	1 MB
Address space of processor	512 MB
Block size	128 B
8 blocks in cache set	

(b) Differentiate between memory mapped I/O and I/O mapped I/O. 10 + 5

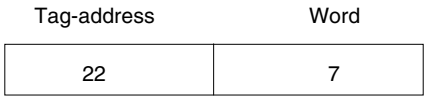
Answer

(a) Given,

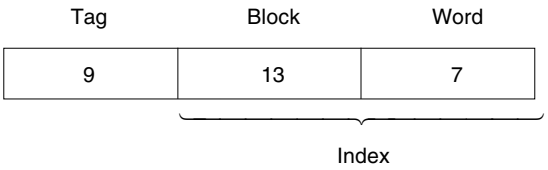
The capacity of main memory = 512 MB

The capacity of cache memory = 1MB

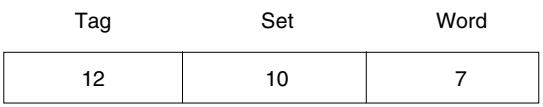
Block size = 128 bytes
A set contains 8 blocks.
Since the address space of the processor is 512 MB.
The processor generates address of 29-bit to access a byte (word).
The number of blocks main memory contains = 512 MB / 128 bytes = 2^{22}.
Therefore, no. of bits required to specify one block in main memory = 22.
Since the block size is 128 bytes.
The no. of bits required to access each word (byte) = 7.
For associative cache, the address format is



The number of blocks cache memory contains = 1 MB / 128 bytes = 2^{13}.
Therefore, no. of bits required to specify one block in cache memory = 13.
The tag field of address = 29 – (13 + 7) = 9-bit.
For direct cache, the address format is



In case of a set-associative cache:
A set contains 8 blocks.
Therefore, the number of sets in cache = 2^{13} / 8 = 2^{10}.
Thus, the number of bits required to specify each set = 10.
The tag field of address = 29 – (10 + 7) = 12-bit.
For set-associative cache, the address format is



(b) 1. In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

2. The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.

3. Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.

4. Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

11. Write short notes on any *three* of the following: 3×5

(a) Magnetic recording
(b) Cache replacement policies
(c) Non-restoring division method
(d) Addressing modes
(e) Booth's algorithm

Answer

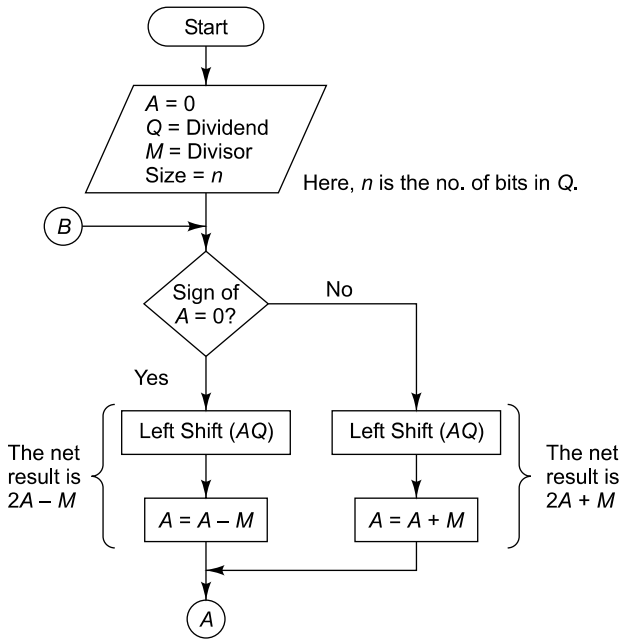
(a) See answer of question no. 10 (a) of 2007 (CS-303).

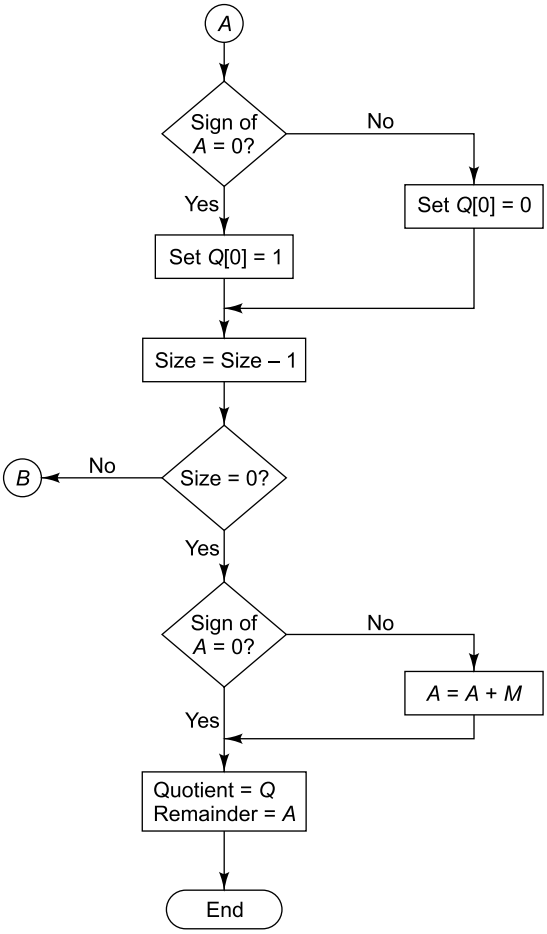
(b) In case a miss occurs in cache memory, then a new data from main memory needs to be placed over old data in the selected location of cache memory. In case of direct mapping cache, we have no choice and thus no replacement algorithm is required. The new data has to be stored only in a specified cache location as per the mapping rule for the direct mapping cache. For associative mapping and set-associative mapping, we need a replacement algorithm since we have multiple choices for locations. We outline below some most used replacement algorithms.

First-In First-Out (FIFO) Algorithm: This algorithm chooses the word that has been in the cache for a long time. In other words, the word which entered the cache first, gets pushed out first.

Least Recently Used (LRU) Algorithm: This algorithm chooses the item for replacement that has been used by the CPU minimum number of times in the recent past.

(c) Non restoring division method: It is described in the following flowchart.





Non-restoring division method

This algorithm removes the restoration steps of restoring algorithm, though it may require a restoration step at the end of algorithm for remainder A, if A is negative.

(d) To obtain the addresses of operands, the address mode is needed. A processor can support various addressing modes in order to give flexibility to the users. The addressing mode gives the way addresses of operands are determined.
The various addressing modes are discussed next.

1. Implied (or inherent) mode: In this mode the operands are indicated implicitly by the instruction. The accumulator register is generally used to hold the operand and after the instruction execution the result is stored in the same register. For example,
RAL; Rotates the content of the accumulator left through carry.
CMA; Takes complement of the content of the accumulator.

2. Immediate mode: In this mode the operand is mentioned explicitly in the instruction. In other words, an immediate-mode instruction contains an operand value rather than an address of it in the address field. To initialize registers to a constant value, this mode of instructions is useful. For example:

MVI A, 06; Loads equivalent binary value of 06 to the accumulator.
ADI 05; Adds the equivalent binary value of 05 to the content of AC.

3. Stack addressing mode: Stack-organized computers use stack addressed instructions. In this addressing mode, all the operands for an instruction are taken from the top of the stack. The instruction does not have any operand field. For example, the instruction

SUB

uses only one op-code (SUB) field, no address field. Both the operands are in the topmost two positions in the stack, in consecutive locations. When the SUB instruction is executed, two operands are popped out automatically from the stack one-by-one. After subtraction, the result is pushed onto the stack. Since no address field is used, the instruction is short.

4. Register (direct) mode: In this mode the processor registers hold the operands. In other words, the address field is now register field, which contains the operands required for the instruction. For example:

ADD R1, R2; Adds contents of registers R1 and R2 and stores the result in R1.

5. Register indirect mode: In this mode the instruction specifies an address of CPU register that holds the address of the operand in memory. In other words, address field is a register which contains the memory address of operand.

6. Auto-increment or auto-decrement mode: This is similar to the register indirect mode except that after or before register's content is used to access memory it is incremented or decremented. It is necessary to increment or decrement the register after every access to an array of data in memory, if the address stored in the register refers to the array. This can be easily achieved by this mode.

7. Direct (or absolute) address mode: In this mode the instruction contains the memory address of the operand explicitly. Thus, the address part of the instruction is the effective address. Examples of direct addressing are:

STA 2500H ; Stores the content of the accumulator in the memory location 2500H.

LDA 2500H ; Loads the accumulator with the content of the memory location 2500H.

8. Indirect address mode: In this mode the instruction gives a memory address in its address field which holds the address of the operand. Thus, the address field of the instruction gives the address where the effective address is stored in memory. The following example illustrates the indirect addressing mode:

MOV (X), R1 ; Content of the location whose address is given in X is loaded into register R1.

9. Relative address mode or PC-relative address mode: In this mode the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. The instruction specifies the memory address of operand as the relative position of the current instruction address. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address.

10. Indexed address mode: In this mode the effective address is determined by adding the content of index register (XR) with the address part of the instruction. This mode is useful in accessing operand array. The address part of the instruction gives the starting address of an operand array in memory. The index register is a special CPU register that contains an index value for the operand. The index value for operand is the distance between the starting address and the address of the operand. For example, an operand array starts at memory address 1000 and assume that the index register XR contains the value 0002. Now consider load instruction

LDA 1000

The effective address of the operand is calculated as:

Effective address = 1000 + content of XR
= 1002.

11. Base register address mode: This mode is used for relocation of the programs in the memory. *Relocation* is a technique of moving program or data segments from one part of memory to another part of memory. Relocation is an important feature of multiprogramming systems. In this mode the content of the base register (BR) is added to the address part of the instruction to obtain the effective address.

(e) See answer of question no. 8(a) of 2007 (CS-303).

2009
Computer Organization and Architecture (CS-404)

Semester: 4th

Time Alloted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) With 2's complement representation, the range of values that can be represented on the data bus of an 8-bit microprocessor is given by

(a) –128 to +127 (b) –128 to +128
(c) –127 to +128 (d) –256 to +256

Answer

(a) –128 to +127

(ii) When signed numbers are used in binary arithmetic then which one of the following notations would have unique representation for zero ?

(a) Sign magnitude (b) Sign 1's complement
(c) Sign 2's complement (d) None of these

Answer

(c) Sign 2's complement

(iii) If the memory chip size is 256×1 bits then the number of chips required to make up 1 kilobyte of memory is

(a) 32 (b) 24 (c) 12 (d) 8

Answer

(a) 32

(iv) How many address bits are required for a 512×4 memory?

(a) 512 (b) 4 (c) 9 (d) A_O-A_6.

Answer

(c) 9

(v) What is the 2's complement representation of –24 in a 16-bit microcomputer ?
(a) 0000 0000 0001 1000 (b) 1111 1111 1110 0111
(c) 1111 1111 1110 1000 (d) 0001 0001 1111 0011.

Answer
(c) 1111 1111 1110 1000

(vi) The technique of placing software in a ROM semiconductor chip is called
(a) PROM (b) EPROM
(c) FIRMWARE (d) microprocessor

Answer
(c) FIRMWARE

(vii) The logic circuit in the ALU is
(a) entirely combinational (b) very cheap memory
(c) content addressable memory (d) slow memory

Answer
(a) entirely combinational

(viii) The principle of locality justifies the use of
(a) interrupts (b) polling (c) DMA (d) cache memory

Answer
(d) cache memory

(ix) Conversion of $(FAFAFA)_{16}$ into octal form is
(a) 76767676 (b) 76575372 (c) 76737672 (d) 76727672

Answer
(b) 76575372

(x) Associative memory is a
(a) pointer addressable memory (b) very cheap memory
(c) content addressable memory (d) slow memory

Answer
(c) content addressable memory

Group-B
(Short-Answer Questions)

Answer any *three of* the following questions. $3 \times 5 = 15$

2. Describe stack base CPU.

Answer
See answer of question number 10 (d) of 2007 (CS-303).

3. Write three points to differentiate I/O mapped IO and memory mapped IO.

Answer

1. In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

2. The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.
3. Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.
4. Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

4. Write a short note on bus organization using tri-state buffer.

Answer

See answer of question number 10 (e) of 2007 (CS-303).

5. Write + 7_{10} in IEEE 64 bit format.

Answer

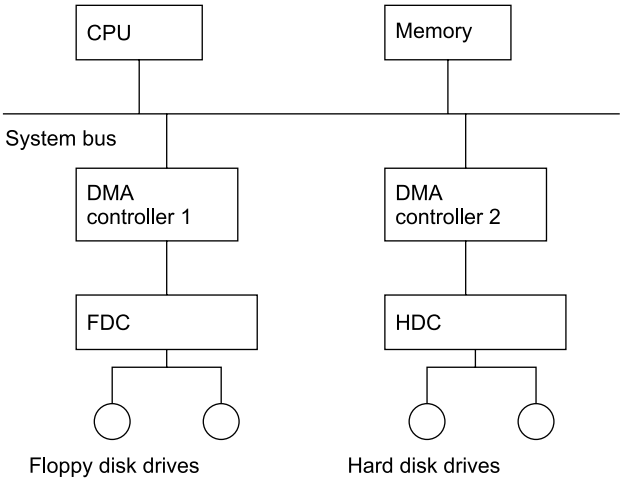
See answer of question number 3 (a) of 2008 (CS-404(EI)).

6. (a) Where does DMA mode of data transfer find its use?
   (b) What are the different types of DMA controllers and how do they differ in their functioning? 2 + 3

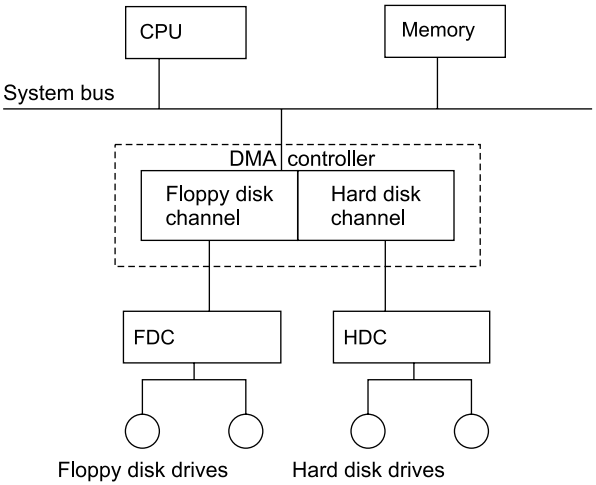
Answer

(a) To transfer large blocks of data at high speed, the DMA method is used. A special controlling unit is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU.

(b) DMA controllers are of two types:
— Independent DMA controller
— DMA controller having multiple DMA-channels
Independent DMA controller:
For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in the figure below for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.



DMA controller having multiple DMA-channels:
In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in the figure below for floppy disk controller (FDC) and hard disk controller (HDC).



Group-C
(Long-Answer Questions)

Answer any *three of* the following questions. $3 \times 15 = 45$

7. (a) Describe the function of major components of a digital computer with a neat sketch.
   (b) Explain the role of an operating system in a computer system.
   (c) Explain the relative advantages and disadvantages of a parallel adder over a serial adder.
   (d) What is the difference between a carry-look ahead adder and a ripple carry adder?
   $7 + 4 + 2 + 2$

Answer

(a) See answer of question number 7(a) of 2007 (CS-303).

(b) An *operating system* (OS) is a set of programs and utilities, which acts as the interface between user programs and computer hardware. It creates a user-friendly environment. The following are the main roles of an operating system:
   (i) Program creation
   (ii) Program execution
   (iii) Accounting
   (iv) Controlling access to files
   (v) System access

(vi) Error detection and response
(vii) Managing the memories of the computer
(viii) Controlling the I/O operations
(ix) Controlling the security of the computer

(c) See answer of question number 9(b) of 2007 (CS-404).

(d) (i) The carry-look ahead adder (CLA) is much faster than a ripple carry adder (RCA). Also, addition delay in a CLA is independent of the number of bits to be added in two operand numbers, whereas, that in a RCA is dependent of the size of the two numbers to be added.
(ii) The circuitry of a CLA is more complex than that of an RCA.

8. (a) Give Booth's algorithm for multiplication of signed 2's complement numbers.
(b) Multiply (+15) and (–11) using Booth's algorithm.
(c) Give the flowchart for division of two binary numbers using restoring division algorithm and explain. 5 + 5 + 5

Answer

(a) See answer of question number 8(a) of 2007 (CS-303).

(b) Multiplication of numbers $(+ 15)_{10}$ and $(-11)_{10}$:
Multiplicand, M = + 15 = 01111 and multiplier, Q = –11 = 10101.

	M	A	Q	Size
Initial Configuration	01111	00000	101010	5
Step-1				
As Q[0] = 1 and Q[–1] = 0				
A = A – M	01111	10001	101010	–
ARS(AQ)	01111	11000	110101	4
Step-2				
As Q[0] = 0 and Q[–1] = 1				
A = A + M	01111	00111	110101	–
ARS(AQ)	01111	00011	111010	3
Step-3				
As Q[0] = 1 and Q[–1] = 0				
A = A – M	01111	10100	111010	–
ARS(AQ)	01111	11010	011101	2
Step-4				
As Q[0]=0 and Q[–1] = 1				
A = A + M	01111	01001	011101	–
ARS(AQ)	01111	00100	101110	1

Step-5

As Q[0] = 1 and
Q[–1] = 0

A = A – M	01111	10101	101110	–
ARS (AQ)	01111	11010	110111	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 1101011011, which shows that the product is a negative number. To get the result in familiar form, take the 2's complement of the magnitude of the number and the result is – 165 in decimal.

(c) See answer of question number 11(b) of 2007 (CS-404).

9. (a) Give the merits and demerits of the floating-point and fixed-point presentations storing real numbers.
   (b) What are biased exponents?
   (c) What are guard bits?
   (d) Convert – 32.75 to IEEE 754 single-precision floating point.
   (e) Use IEEE single-precision floating point numbers to compute 13.25 + 4.5.

4 + 2 + 2 + 3 +4

Answer

(a) *Merits of fixed-point representation*:
   (i) This method of representation is suitable for representing integers in registers.
   (ii) Very easy to represent, because it uses only one field: magnitude field.

Demerits of fixed-point representation:
   (i) Range of representable numbers is restricted.
   (ii) It is very difficult to represent complex fractional numbers.
   (iii) Since there is no standard representation method for it, it is some time confusing to represent a number in this method.

Merits of floating-point representation:
   (i) By this method, any type and any size of numbers can be represented easily.
   (ii) There are several standardized representation methods for this.

Demerits of floating-point representation:
   (i) Relatively complex representation, because it uses basically two fields: mantissa and exponent fields.
   (ii) Length of register for storing floating-point numbers is large.

(b) In order to eliminate the sign bit of the exponent E, one positive constant (C) is added with the original exponent E to make it as an unsigned integer number E'. This E' is called biased exponent, where $E' = E + C$. Thus, in this process, we can use n bits to represent an exponent instead of $(n - 1)$ bits.

(c) When the mantissa is shifted right, some bits at the right most position (least significant position) are lost. In order to obtain maximum accuracy of the final result; one or more extra bits known as *guard bits*, are included in the intermediate steps. These bits temporarily contain the recently shifted out bits from the right most side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are not stored. However, based on

the guard bits, the value of the mantissa can be made more precise by one of three rounding techniques: chopping, von Neumann rounding and rounding.

(d) The decimal number – 32.75 = –100000.11 in binary = $-1.0000011_2 \times 2^5$
The 23-bit mantissa M = 0.00000 110000 000000 000000
The biased exponent E' = E + 127 = 5 + 127 = 132 = 1000 0100
Since the number is negative, the sign bit S = 1
Therefore, the IEEE 754 single-precision (32-bit) representation is:

1	1000 0100	00000 110000 000000 000000

(e) The decimal number 13.25 = $1.10101_2 \times 2^3$
Therefore, the IEEE single-precision (32-bit) representation of 13.25 is

0	1000 0010	10101 000000 000000 000000

The decimal number 4.5 = $1.001_2 \times 2^2$
Therefore, the IEEE single-precision (32-bit) representation of 4.5 is

0	1000 0001	00100 000000 000000 000000

Shifting 4.5 to the right one place to make the exponents the same (i.e., 3) gives 4.5 = $0.10010_2 \times 2^3$. Adding mantissas ($1.10101_2 + 0.10010_2$) gives a result of 10.00111_2.

So, we have to shift the resultant mantissa to one position right to get the mantissa as 1.000111_2 and thus exponent as 4.
Therefore, the IEEE single-precision representation of this is

0	1000 0011	00011 100000 000000 000000

10. (a) Compare RISC with CISC.
(b) What do you mean by pipeline processing?
(c) What are instruction pipeline and arithmetic pipeline?
(d) Differentiate between polled I/O and interrupt driven I/O.
(e) Distinguish between vectored and non-vectored interrupts. 4 + 2 + 2 + 3 + 4

Answer

(a) See answer of question number 6 of 2007 (CS-404).
(b) See answer of question number 7(a) of 2007 (CS-404).
(c) *Instruction pipeline*: When the execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions, it is called instruction pipeline. All high-performance computers are now equipped with instruction pipeline.
Arithmetic pipeline: An arithmetic pipeline divides an arithmetic operation, such as multiply, into multiple arithmetic steps each of which is executed one-by-one in different arithmetic stages in the ALU. The number of arithmetic pipelines varies from processors to processors.

(d) (i) In the polled I/O or programmed I/O method, the CPU stays in the program until the I/O device indicates that it is ready for data transfer, that is, CPU is kept busy needlessly. But, in interrupt driven I/O method, CPU can perform its own task of instruction executions and is informed by raising an interrupt signal when data transfer is needed.

(ii) Polled I/O is low cost and simple technique; while, interrupt I/O technique is relatively high cost and complex technique. Because in second method, a device controller is used to continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer.

(iii) The polled I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. However, interrupt I/O method is very useful in modern high speed computers.

(e) Interrupt is a special signal to the CPU generated by an external device that causes the CPU to suspend the execution of one program and start the execution of another.

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially, the set-up time is quite large.

11. (a) What do you mean by logical address space and physical address space?

(b) Explain with an example how logical address is converted into physical address and also explain how page replacements take place.

(c) Write the advantages of virtual memory system.

(d) (i) How many address lines are present in a 256 k × 8 RAM?

(ii) How many such RAMs will be required to construct 1 M × 32 memory bank?

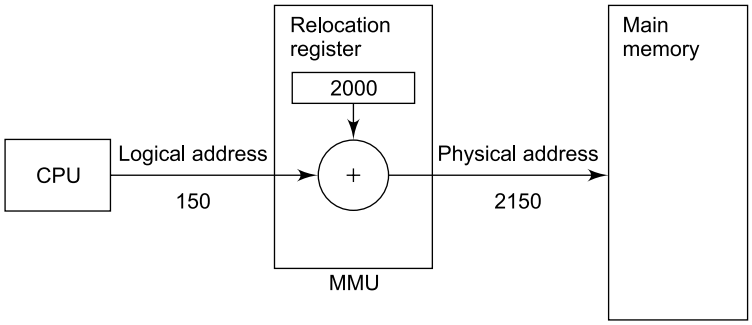
(iii) How many such RAMs will be required to construct 512 k × 32 memory bank?

$2 + 4 + 3 + (3 \times 2)$

Answer

(a) When a program needs to be executed, the CPU would generate addresses, called *logical* addresses. The corresponding addresses in the physical memory, as occupied by the executing program, are called *physical* addresses. The set of all logical addresses generated by the CPU or program is called *logical-address space* and the set of all physical addresses corresponding to these logical addresses is called *physical-address space*. The memory-management unit (MMU) maps each logical address to a physical address during program execution.

(b) The memory-management unit (MMU) maps each logical address to a physical address during program execution. The figure below illustrates this mapping method, which uses a special register called base register or relocation register.



A memory-management scheme

The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

Page Replacement

When a program starts execution, one or more pages are brought to the main memory and the page table is responsible to indicate their positions. When the CPU needs a particular page for execution and that page is not in main (physical) memory (still in the secondary memory), this situation is called *page fault*. When the page fault occurs, the execution of the present program is suspended until the required page is brought into main memory from secondary memory. The required page replaces an existing page in the main memory, when it is brought into main memory. Thus, when a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several replacement algorithms such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement* algorithm available.

The *FIFO algorithm* is simplest and its criterion is "select a page for replacement that has been in main memory for longest period of time".

The *LRU algorithm* states that "select a page for replacement, if the page has not been used often in the past". The LRU algorithm is difficult to implement, because it requires a counter for each page to keep the information about the usage of page.

The *optimal algorithm* generally gives the lowest page faults of all algorithms and its criterion is "replace a page that will not be used for the longest period of time". This algorithm is also difficult to implement, because it requires future knowledge about page references.

(c) See answer of question number 2 of 2007 (CS-404).

(d) (i) Since $256k = 2^{18}$, the RAM of size $256k \times 8$ requires 18 address lines.

(ii) Size of memory bank to be constructed is $1M \times 32$. Thus, it requires 20 (because $1M = 2^{20}$) address lines and 32 data lines.

Thus, total number of RAMs each of size $256k \times 8$ required = $(2^{20}/2^{18}) * (32/8) = 4 * 4 = 16$.

(iii) Size of memory bank to be constructed is $512k \times 32$. Thus, it requires 19 (because $512k = 2^{19}$) address lines and 32 data lines.

Thus, total number of RAMs each of size $256k \times 8$ required = $(2^{19}/2^{18}) * (32/8) = 2 * 4 = 8$.

2009
Computer Organization and Architecture (CS-404 (EI))

Semester: 4th

Time Alloted: 3 hours *Full Marks: 70*

Group-A
(Multiple-Choice Questions)

1. Choose the correct alternatives for the following : $10 \times 1 = 10$

(i) The logic circuit in ALU is
(a) entirely combinational (b) entirely sequential
(c) combinational cum sequential (d) none of these

Answer
(a) entirely combinational

(ii) In a microprocessor the address of the next instruction to be executed is stored in
(a) stack pointer (b) address latch
(c) program counter (d) general purpose register.

Answer
(c) program counter

(iii) The basic principle of a von Neumann computer is
(a) storing program and data in separate memory
(b) using pipeline concept
(c) storing both program and data in the same memory
(d) using a large number of registers.

Answer
(c) storing both program and data in the same memory

(iv) Physical memory broken down into groups of equal size is called
(a) page (b) tag (c) block (d) index

Answer
(c) Block

(v) The minimum number of operands with any instruction is
(a) 1 (b) 0 (c) 2 (d) 3.

Answer
(b) 0

(vi) The principle of locality justifies the use of
(a) interrupts (b) DMA (c) polling (d) cache memory

Answer
(d) cache memory

(vii) Instruction cycle is
(a) fetch-decode-execution (b) fetch-execution-decode
(c) decode-fetch-execution (d) none of these

Answer
(a) fetch-decode-execution

(viii) How many address bits are required for a 1024×8 memory ?
(a) 1024 (b) 5 (c) 10 (d) none of these.

Answer
(c) 10

(ix) The technique of placing software in a ROM semiconductor chip is called
(a) PROM (b) EPROM (c) EEPROM (d) firmware

Answer
(d) firmware

(x) Cache memory is used to increase the speed of
(a) hard disk (b) CPU (c) floppy disk (d) none of these

Answer
(b) CPU

Group-B
(Short-Answer questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. What do you mean by instruction cycle, machine cycle and T states ? 5

Answer
See answer of question no. 4 of 2007 (CS-404).

3. Briefly discuss register stack organization of CPU. 5

Answer
See answer of question no. 10(d) of 2007 (CS-303).

4. What is locality of reference ? What is memory mapping ? Why is it needed ? 2 + 1 + 2

Answer

Locality of reference: Analysis of a large number of typical programs shows that the CPU references to main memory during some time period tend to be confined within a few localized areas in memory. In other words, few instructions in the localized areas in memory are executed repeatedly for some

time duration and other instructions are accessed infrequently. This phenomenon is known as the property of locality of reference. This property may be understood considering that when a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitutes the loop. Thus loop tends to localize the references to memory for fetching the instructions.

Memory Mapping and Why: First the cache memory is accessed, when the CPU needs to access memory for a word. If the word is found in the cache, the CPU reads it from the fast cache memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed next to find the word. Due to the property of locality of reference, a block of words containing the one just accessed is then brought into the cache memory from main memory. The transfer of data as a block from main memory to cache memory is referred to as a mapping process.

5. Discuss IEEE representation of floating point number. 5

Answer

See answer of question no. 4(a) of 2007 (CS-303)

Group–C
(Long-Answer Questions)

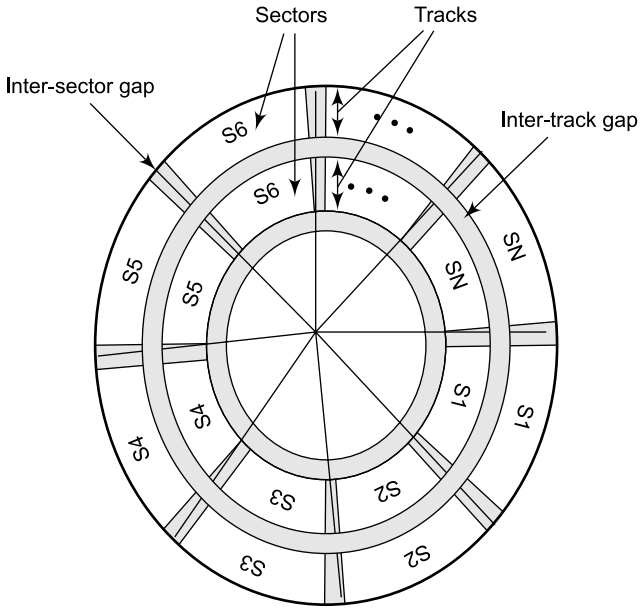
Answer any *three* questions. $3 \times 15 = 45$

6. (a) Write notes on magnetic disk and magnetic tape. 5
   (b) A disk pack has 19 surfaces. Storage area on each surface has an inner diameter of 22 cm and an outer diameter of 33 cm. Maximum storage density on each track is 2000 bits/cm and minimum spacing between tracks is 0.25 mm.
      (i) What is the storage capacity of the pack ?
      (ii) What is the data transfer rate in bytes per second at a rotational speed of 3600 rpm? 2 +2
   (c) What is seek time and rotational latency? 1 + 1
   (d) Suppose a DRAM memory has 4*k* rows in its array of bits cells. Its refreshing period is 64 ms. 4 clock cycles are required to access each row.
      (i) What is the time needed to refresh the memory if the clock rate is 133 MHz ?
      (ii) What fraction of the memory's time is spent for performing refreshes ? 2 + 2

Answer

(a) *Magnetic Disk:* Disks that are permanently attached to the unit assembly and cannot be removed by the general user are called hard disks. A disk drive with removable disks is called a floppy disk drive. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.

The magnetic disk is made of either aluminium or plastic coated with a magnetic material so that information can be stored on it. The recording surface is divided into a number of concentric circles called *tracks*. The tracks are commonly divided into sections called *sectors*. To distinguish between two consecutive sectors, there is a small *inter-sector gap*. In most systems, the minimum quantity of information transfer is a sector. Generally, the innermost track has maximum storage density (i.e., bits per linear inch) and outermost track has minimum density. The subdivision of one disk surface into tracks and sectors is shown in the figure.

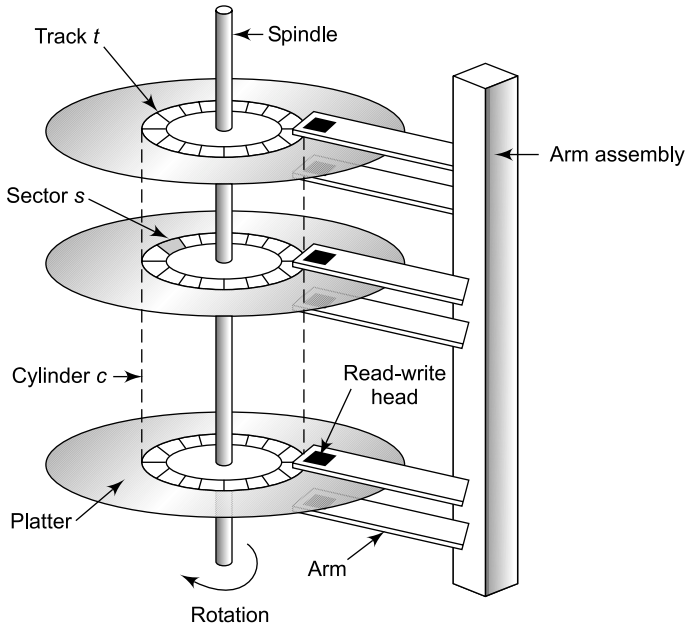


A single disk view

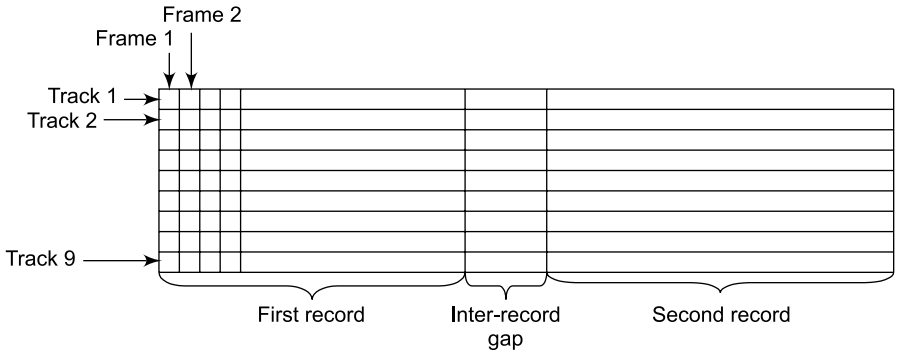
The information is accessed onto the tracks using movable read-write heads that move from the innermost to the outmost tracks and vice-versa. Generally, several identical disks are stacked over one another with some separation between them to form a *disk pack*. A typical disk pack is shown in the figure. There is one read-write head per surface. Therefore, if there are n disks, there are $2n$ surfaces. During normal operation, disks are rotated continuously at a constant angular velocity. Same radius tracks on different surfaces of disks form a logical *cylinder*. A disk pack with n disks has $2n$ tracks per cylinder. Another part of the disk is the electronic circuitry that controls the operation of the disk, which is called *disk controller*. To access data, the read-write head must be placed on the proper track based on the given cylinder address.

Magnetic Tape: Magnetic tapes were the first kind of secondary memory used in computer systems. Tape is flexible polyester coated with special magnetic material. A magnetic tape is similar to a home tape recorder. However, a magnetic tape holds digital information, whereas a tape recorder holds analog information. A magnetic tape is divided vertically into frames and horizontally into nine parallel tracks, as in the figure.

Each frame is capable of storing 9 bits of data. The first 8 bits form a data byte and the 9^{th} bit holds the parity. The parity bit is used for error correction and detection. Information is stored along tracks using read-write heads. Read-write heads are designed in such a way that they can access all nine tracks contained in a frame simultaneously. Data is written on the tape by varying the current through the read-write heads. Data is read or written in contiguous records. The records are separated by gaps referred to as inter-record gaps. The length of a magnetic tape is typically 2400 feet and it is stored on a reel. The major difficulty with this device is the particle contamination caused by improper manual handling.



A disk pack



A part of a magnetic tape.

(b) Given, no. of surfaces = 19
Inner track diameter = 22 cm
Outer track diameter = 33 cm
So, Total track width = (33-22)/2 cm = 5.5 cm
Track separation = 0.25 mm
Thus, no. of tracks/surface = (5.5 ∗ 10)/0.25 = 220
Minimum track circumference = 22 ∗ Π cm
Maximum track storage density = 2000 bits/ cm, which will be on innermost track.
So, data storage capacity/track = 22 ∗ Π ∗ 2000 bits = 138.23 Kbits
Disk speed = 3600 rpm

So, rotation time = 1/3600 minute = 16.67 ms (1 ms.=10^3 s)

i. Storage capacity = 19 ∗ 220 ∗ 138.23 kbits = 577.8 Mbits = 72.225 Mbytes.

ii. Data transfer rate = 138.23 kbits/16.67 ms = 8.2938 Mbits/s

This is the maximum data transfer rate excluding seek time and rotational latency.

(c) To access data from a magnetic disk, the read-write head must be placed on the proper track based on the given cylinder address. The time required to position the read-write head over the desired track is known as the *seek time*, t_s. This depends on the initial position of the head relative to the specified track or cylinder address. Seeking the required track is the most time-consuming operation because it involves moving the read-write head arm. After positioning the read-write head on the desired track, the disk controller has to wait until the desired sector is under the read-write head. This waiting time is known as *rotational latency*, t_l. Rotational latency depends on the rotation speed of the disk. The *access time* of the disk is the sum of t_s and t_l.

(d) Given,

Memory has 4k (= 4096) rows

It has refreshing period is 64 ms. In other words, period of refreshing all rows of memory is 64 ms.

To access each row, 4 clock cycles are required.

Therefore, it takes $4096 \times 4 = 16384$ cycles to refresh all rows.

(i) At a clock rate of 133 MHz, the time needed to refresh all rows (i.e. memory) is $16384 / (133 \times 10^6) = 123 \times 10^{-6}$ seconds (approx.).

(ii) Therefore, the refreshing process occupies 0.123 ms in each 64 ms time interval. Thus, the refreshing overhead is 0.123/64 = 0.0019, which is less than 0.2 per cent of the total time available for accessing the memory.

7. (a) Classify different types of ROM and briefly describe them. What is flash memory? Briefly describe the organization of a basic RAM cell. 3 + 1 + 3

(b) What is von Neumann architecture ? What is von Neumann bottleneck ? How can this be reduced? 2 + 1 + 2

(c) What is virtual memory ? Why is it called virtual ? 1 + 1

(d) What is tertiary memory ? 1

Answer

(a) Types of ROM

PROM Memory: Some ROM designs allow the data to be loaded into the cell by user, and then this ROM is called PROM (Programmable ROM). A fuse is used in each cell to achieve programmability. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses of cells at these locations using high-voltage currents. The PROM's cells are once programmable, i.e., the user can store the desired bits in the cells only once and these bits cannot be altered.

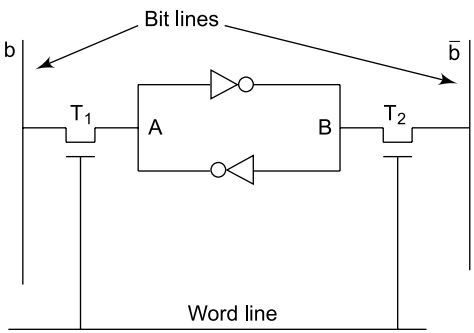
EPROM: An erasable PROM (EPROM) uses a transistor in each cell that acts as a programmable switch. The contents of an EPROM can be erased (set to all 1s) by burning out the device to ultraviolet light for a few (20 to 30) minutes. Since ROMs and PROMs are simpler and thus cheaper than EPROMs. The EPROMs are used during system development and debugging.

EEPROM: In many applications, permanent data have to be generated in a program application and need to be stored. For example, in a mobile phone the telephone numbers are to be kept permanently till the user wants to erase those data. Similarly, the user may wish to erase previously entered information. EEPROMs have an advantage in that the information in them can be selectively erased by writing 1s and each bit in the information can be stored again by writing the desired bit. An EEPROM needs two write operations at an address, one for erase and one for writing. RAM writes the information directly without first erasing that information at that address. But in the EEPROM, the stored information is non-volatile.

Flash Memory: A currently popular type of EEPROM, in which erasing is performed in large blocks rather than bit by bit, is known as flash EPROM or flash memory. Erasing in large blocks reduces the overhead circuitry, thus leading to greater density and lower cost. The current trend is "memory stick" made of flash memory that is used to Universal Serial Bus (USB) of the personal computer for data exchange between computers.

RAM Memory Cell Organization

Figure below shows a cell diagram of basic RAM memory. A latch is formed by two inverters connected as shown in the figure. Two transistors T_1 and T_2 are used for connecting the latch with two bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch retains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point B is 0. This state is retained as long as the word line is not activated.



A basic RAM cell

Read Operation: For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation: Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.

(b) See answer of question no. 8(c) of 2007 (CS-303)
Reduction of von Neumann bottleneck: This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required data-word to come. Another way to reduce the problem is by using special type computers known as *Reduced Instruction Set Computers (RISC)*. This class of computers generally uses a large number of registers, through which the most of the instructions are executed. This computer usually limits access to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

(c) See answer of question no. 2 of 2007 (CS-404)

(d) *Tertiary Memory*: This memory is a third level of storage, apart from main memory and secondary memory. It is primarily used for extraordinarily large data storage. This memory is much slower than the secondary memory and mainly used for storing rarely needed data. When this data is needed by the system, it is copied to secondary memory before use. Typical examples of this type memory include tape library and optical jukeboxes. In robotic mechanism, this memory is mainly used.

8. (a) Draw the flowchart of Booth's Multiplication Algorithm and explain it. Perform the Booth's Multiplication on (–9) * (–13) up to five digits. Show every step. 5 + 5
(b) Draw the flowchart of storing and re-storing Division Algorithm. 5

Answer

(a) For Booth's Algorithm: See answer of question no. 8(a) of 2007 (CS-303)
Multiplication of numbers $-9)_{10}$ and $(-13)_{10}$:
Multiplicand, M = –9 = 10111 and multiplier, Q = – 13 = 10011.

	M	A	Q		Size
Initial Configuration	10111	00000	10011	0	5
Step-1 As Q[0] = 1 and Q[–1] = 0					
A = A – M	10111	01001	10011	0	—
ARS(AQ)	10111	00100	11001	1	4
Step-2 As Q[0] = 1 and Q[–1] = 1					
ARS(AQ)	10111	00010	01100	1	3
Step-3 As Q[0] = 0 and Q[–1] = 1					
A = A + M	10111	11001	01100	1	—
ARS(AQ)	10111	11100	10110	0	2

Step-4				
As Q[0] = 0 and				
Q[–1] = 0				
ARS(AQ)	10111	11110	01011 0	1
Step-5				
As Q[0] = 1 and				
Q[–1] = 0				
A = A – M	10111	00111	01011 0	—
ARS (AQ)	10111	00011	10101 1	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 00011 10101, which shows that the product is a positive number and the result is + 117 in decimal.

(b) See answer of question no. 11(b) of 2007 (CS-404)

9. (a) What will be the maximum capacity of a memory which uses an address bus of size 12 bit? 1

(b) What is an instruction format ? What is instruction cycle ? Draw the state transition diagram of an instruction cycle. 2 + 1 + 3

(c) What is interrupt ? What is the difference between vectored & non-vectored interrupts? 1 + 2

(d) Why is DMA mode of data transfer used ? What are the different types of DMA controllers and how do they differ in their functioning ? 2 + 3

Answer

(a) The maximum capacity of memory will be 2^{12} words, i.e., 4096 words.

(b) *Instruction Format*: A computer usually has a variety of instruction formats. It is the task of the control unit within CPU to interpret each instruction code and to provide the necessary control functions needed to process the instruction. The most common format followed by instructions is depicted in the figure.

Different fields of instructions

The bits of the instruction are divided into groups called fields. The commonly used fields found in instruction formats are the following:

1. *Operation Code* (or, simply *Op-code*): This field states the operation to be performed. This field defines various processor operations, such as add, subtract, complement, etc.
2. *Address:* An address field designates a memory address or a processor register or an operand value.
3. *Mode*: This field specifies the method to get the operand or effective address of operand. In some computers' instruction set, the op-code itself explicitly specifies the addressing mode used in the instruction. A computer has various addressing modes.

For example, in the instruction ADD R1, R0; ADD is the op-code to indicate the addition operation and R1, R0 are the address fields for operands.

Instruction Cycle: The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle (see figure) that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

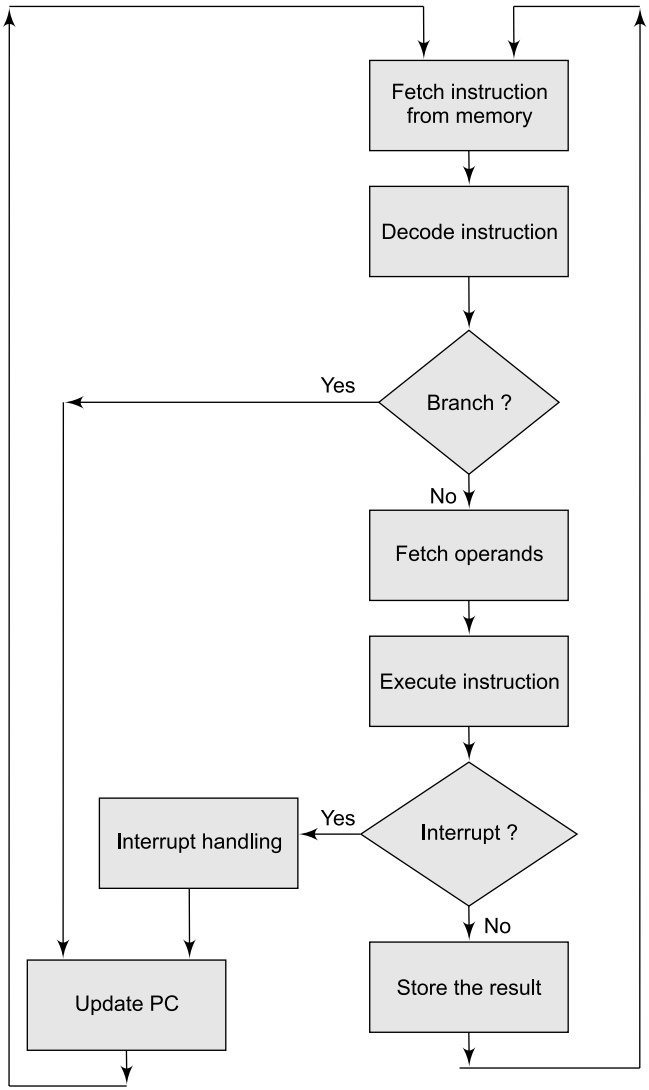


Figure 6 *Instruction cycle.*

The step 1 is basically performed using a special register in the CPU called *program counter* (PC) that holds the address of the next instruction to be executed. If the current instruction is simple arithmetic/logic or load/store type the PC is automatically incremented. Otherwise, PC is loaded with the address dictated by the currently executing instruction. The decoding done in step 2 determines the operation to be performed and the addressing mode of the instruction for calculation of address of operands. After getting the information about the addresses of operands, the CPU fetches the operands in step 3 from memory or registers and stores them its registers. In step 4, the ALU of processor executes the instruction on the stored operands in registers. After the execution of instruction, in phase 5 the result is stored back in memory or register and returns to step 1 to fetch the next instruction in sequence. All these sub-operations are controlled and synchronized by the control unit.

(c) See answer of question no. 10(e) of 2009 (CS-303)

(d) A special controlling unit called DMA controller is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA).*

DMA is useful, because it has following advantages:

(i) High-speed data transfer is possible, since CPU is not involved during actual transfer, which occurs between I/O device and the main memory.

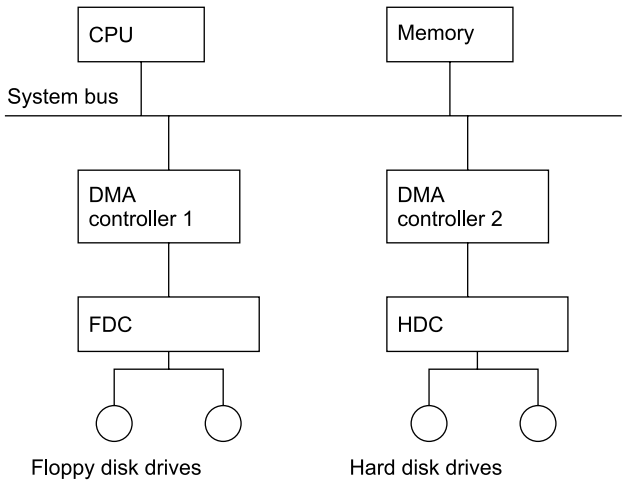
(ii) Parallel processing can be achieved between CPU processing and DMA controller's I/O operation.

DMA controllers are of two types:

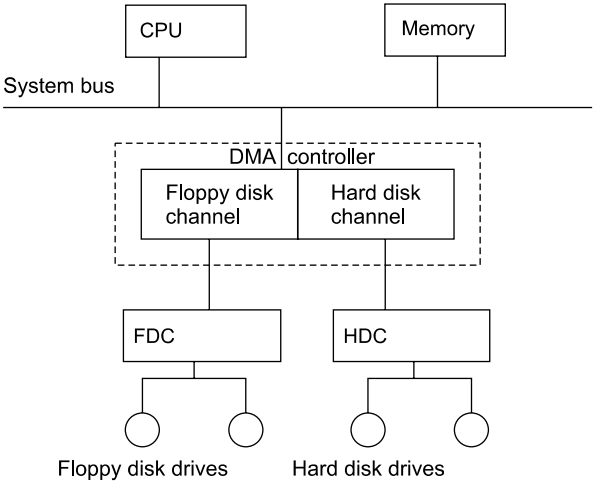
– Independent DMA controller
– DMA controller having multiple DMA-channels

Independent DMA controller:

For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.



DMA controller having multiple DMA-channels:
In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC).



10. (a) What is hit ratio? What is the difference between associative and set-associative mappings? 1 + 2
    (b) A hierarchical cache-main memory sub-system has the following specifications: Cache access time : 50 ns, Main memory access time : 500 ns, 80% of memory request for read, hit ratio : 0.9 for read access and write-through scheme is used.
        (i) Calculate the average access time of the memory system considering only memory read cycle.
        (ii) Calculate the average access time of the memory system both for read and write. 2 + 2
    (c) Briefly describe Flynn's classification of parallel computers. 4
    (d) Discuss about the different hazards in pipelining. 3
    (e) What is array processor? 1

Answer

(a) *Hit ratio:* When the CPU refers to memory and finds the word in cache, it is said that a *hit* occurred. If the word is not found in cache, then the CPU refers to the main memory for the desired word and it is referred to as a *miss* to cache. The hit ratio (h) is defined below:

$$\text{Hit ratio } (h) = \frac{\text{number of hits}}{\text{total CPU references to memory}}$$

$$= \frac{\text{number of hits}}{\text{number of hits} + \text{number of misses}}$$

Thus, the hit ratio is nothing but a probability of getting hits out of some number of memory references made by CPU. So its range is $0 \leq h \leq 1$.

Difference between associative and set-associative mappings: The associative cache memory uses the fastest and most flexible mapping method, in which both address and data of the memory word are stored. Whereas, in set-associative cache, two or more words can be stored under the same index address, which is not stored in the memory. Each data word is stored together with its tag. The number of tag-data words under an index is said to form a *set*. The set-associative cache has higher hit ratio compared to associative cache. But, the set-associative cache is the most expensive memory. The cost increases as set size increases.

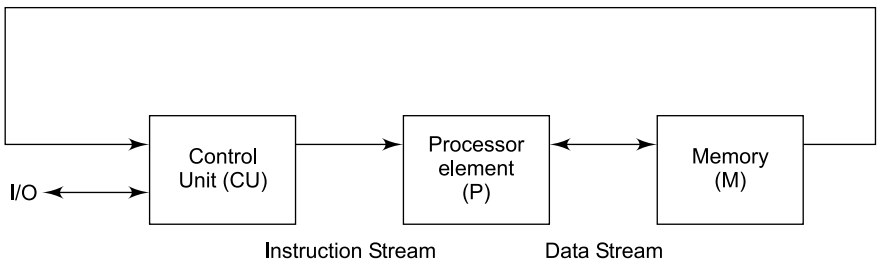
(b) Given,
Cache access time $t_c = 50$ ns
Main memory access time $t_m = 500$ ns
Probability of read $p_r = 0.8$
Hit ratio for read access $h_r = 0.9$
Writing scheme: write-through.

(i) Considering only memory read cycle,
the average access time $t_{av\text{-}r} = h_r * t_c + (1 - h_r) * (t_c + t_m)$
$= 0.9 * 50 + (1 - 0.9) * 550$
$= 100$ ns

(ii) For both read and write cycles,
the average access time $= p_r * t_{av\text{-}r} + (1 - p_r) * t_{m.}$ Since in write-through method, access time for write cycle will be the main memory access time.
$= 0.8 * 100 + (1 - 0.8) * 500$
$= 180$ ns

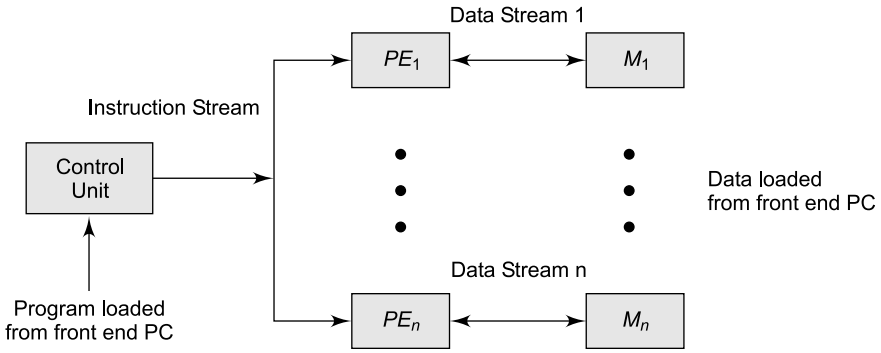
(c) *Flynn's classification:* Based on the number of simultaneous instruction and data streams used by a CPU during program execution, digital computers can be classified into four categories as:
- Single instruction stream-single data stream (SISD) machine.
- Single instruction stream-multiple data stream (SIMD) machine.
- Multiple instruction stream-single data stream (MISD) machine.
- Multiple instruction stream-multiple data stream (MIMD) machine.

SISD Computer
Most serial computers available today fall in this organization as shown in figure next. Instructions are executed sequentially but may be overlapped in their execution stages (In other words the technique of pipelining can be used in the CPU). Modern day SISD uniprocessor systems are mostly pipelined. Examples of SISD computers are IBM 360/91, CDC Star-100 and TI-ASC.



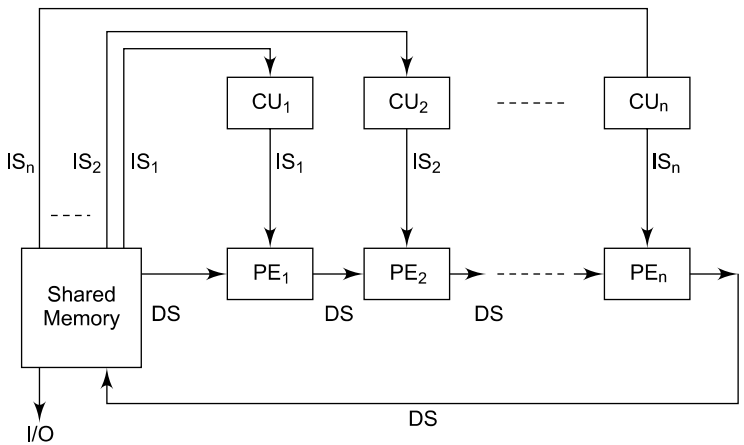
SIMD Computer

Array processors fall into this class. As illustrated in figure next, there are multiple processing elements supervised by the common control unit. All PEs (processing elements, which are essentially ALUs) receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams. The shared memory subsystem containing multiple modules is very essential. This machine generally used to process vector type data. Examples of SIMD computers includes Illiac-IV and BSP.



MISD Computer

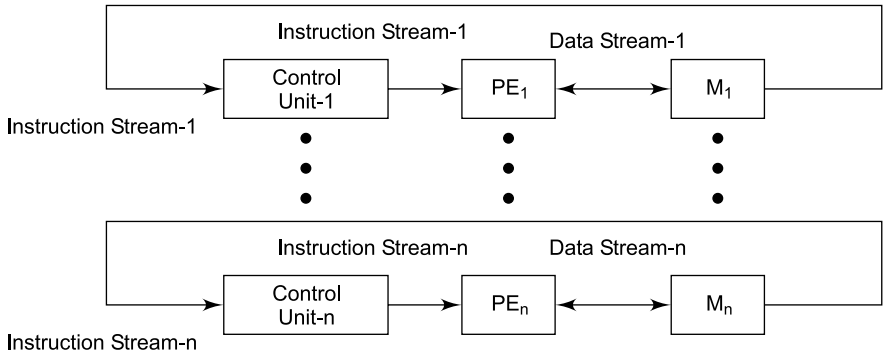
Very few or no parallel computers fit in this organization, which is conceptually illustrated in figure next. There are n processor elements, each receiving distinct instructions to execute on the same data stream and its derivatives. The results (outputs) of one processor element become the inputs (operands) of the next processor element in the series. This architecture is also known as *systolic arrays*.



Captions: CU: control unit PE: processing element IS: instruction stream DS: data stream.
MISD computer (Systolic array)

MIMD Computer

This category covers multiprocessor systems and multiple computer systems. The structure of MIMD computer is shown in figure next. An MIMD computer is called *tightly coupled (or Uniform Memory Access (UMA))* if the degree of interactions among the processors is high. Otherwise, we consider them *loosely coupled (or Non-Uniform Memory Access (NUMA))*. Most commercial MIMD computers are loosely coupled. Examples of MIMD multiprocessors are C.m*, C.mmp, Cray-3 and S-1.



MIMD computer

(d) *Pipeline hazards*: Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. The instruction is said to be *stalled*. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall.

Types of pipeline hazards are:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards

They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Solution of control hazards:

In order to cope with the adverse effects of branch instructions, an important technique called *prefetching* is used. *Prefetching technique* states that: Instruction words ahead of the one currently being decoded in the instruction-decoding (ID) stage are fetched from the memory system before the ID stage requests them.

Structural Hazards

Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Example: Instruction ADD R4, X fetches operand X from memory in the OF stage at 3rd clock period. The memory doesn't accept another access during that period. For this, (i+2)th instruc-

tion cannot be initiated at 3rd clock period to fetch the instruction from memory. Thus, one clock cycle is stalled in the pipeline for all subsequent instructions. This is illustrated next.

Clock cycles ⟶	1	2	3	4	5	6	7	8	9	10	11	12	13
ADDR4, X	IF	ID	OF	EX	WB								
Instr. i+1		IF	ID	OF	EX	WB							
Instr. i+2			stall	IF	ID	OF	EX	WB					
Instr. i+3					IF	ID	OF	EX	WB				

Penalty : 1 cycle

Structural hazard in instruction pipeline

Solution of structural hazards:
Certain resources are duplicated in order to avoid structural hazards. Functional units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time. A classical way to avoid hazards at memory access is by providing separate data and instruction caches.

Data Hazards
Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard.*

Example: We have two instructions, I1 and I2. In a pipeline the execution of I2 can start before I1 has terminated. If in a certain stage of the pipeline, I2 needs the result produced by I1, but this result has not yet been generated, we have a data hazard.

According to various data update patterns in instruction pipeline, there are three classes of data hazards exist:

- Write After Read (WAR) hazards
- Read After Write (RAW) hazards
- Write After Write (WAW) hazards

Solution of data hazards:
The system must resolve the interlock situation when a hazard is detected. Consider the sequence of instructions {…. I, I+1, …., J, J+1,….}in which a hazard has been detected between the current instruction J and a previous instruction I. This hazardous situation can be resolved in one of the two following ways:

- One simple solution is to stall the pipeline and to ignore the execution of instructions J, J+1,…., down the pipeline until the instruction I has passed the point of resource conflict.
- A more advanced approach is to ignore only instruction J and continue the flow of instructions J+1, J+2,…, down the pipeline. However, the potential hazards due to the suspension of J must be continuously tested as instructions J+1, J+2,… execute prior to J. Thus, multilevel of hazard detection may be encountered, which requires much more complex control policies to resolve such multilevel of hazards.

(e) An array processor is a synchronous array of parallel processors and consists of multiple processing elements (PEs) (PEs are essentially ALUs) under the supervision of one control unit (CU). An array processor can handle single instruction and multiple data (SIMD) streams. Hence, array processors are also known as SIMD computers. SIMD machines are especially designed to perform vector computations over large matrices or arrays of data.

2009
Computer Architecture and Organization (EC-503)

Time Alloted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following : $10 \times 1 = 10$

(i) A digital computer has a common bus system for 16 registers of 32-bits each. How many MUX are needed and what will be the size of each MUX?
(a) 32, 16 (b) 16, 32 (c) 8, 16 (d) 16, 8

Answer
(a) 32, 16×1

(ii) The basic principle of a Harvard computer is
(a) storing program and data in separate memory
(b) storing program and data in same memory
(c) using pipeline concept
(d) using a large number of registers

Answer
(a) storing program and data in separate memory

(iii) A digital computer has a memory unit of 32 k $\times$ 12 and cache memory of 512×12 words. The cache uses direct mapping. How many bits are there in tag, index field?
(a) 6, 10 (b) 10, 6 (c) 9, 6 (d) 6, 9.

Answer
(d) 6, 9

(iv) A 'hit' occurs
(a) when a word is found in virtual memory
(b) when a word is found in cache memory

(c) when a word is not found in virtual memory
(d) when a word is not found in cache memory

Answer

(b) when word is found in cache memory

(v) Delayed branching is related to
(a) pipeline hazard (b) pipeline remedy (c) both (a) and (b) (d) none of these

Answer

(b) pipeline remedy

(vi) Normalized representation of 0.00101×2^2 is
(a) 0.00101×2^2 (b) 1.01×2^2 (c) $1{\cdot}01 \times 2^{-1}$ (d) none of these

Answer

(c) $1{\cdot}01 \times 2^{-1}$

(vii) Principle of Locality justifies the use of
(a) DMA (b) cache memory (c) main memory (d) none of these

Answer

(b) Cache Memory

(viii) The first computer used to store a program is
(a) EDSAC (b) ENIAC (c) EDVAC (d) none of these

Answer

(b) ENIAC

(ix) Number of transistors in a CMOS static RAM cell is
(a) 1 (b) 4 (c) 6 (d) none of these

Answer

(c) 6

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. Draw the control circuit for the following RTL:

$T_1 : A \leftarrow B$
$T_2 : A \leftarrow C$ 5

Answer

Here, we have two control functions, T_1 and T_2. The RTL:

$T_1 : A \leftarrow B$
$T_2 : A \leftarrow C$

means that A will be loaded with the content of register B if control function T_1 is enabled, otherwise A will be loaded with the content of register B if control function T_2 is enabled. In other words, we can say that T_1 and T_2 are complementary to each other. Thus, the given RTL can be stated as "if ($T_1 = 1$) then $A \leftarrow B$ else $A \leftarrow C$". Therefore, the circuit diagram for the register transfer operations is as shown below.

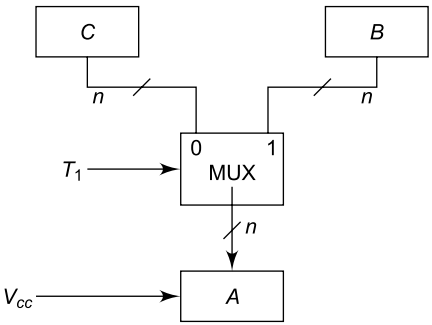


Figure 1 *Hardware implementation of RTL " $T_1 : A \leftarrow B$ $T_2 : A \leftarrow C$"*

3. With a diagram, distinguish between DRAM and SRAM. 5

Answer

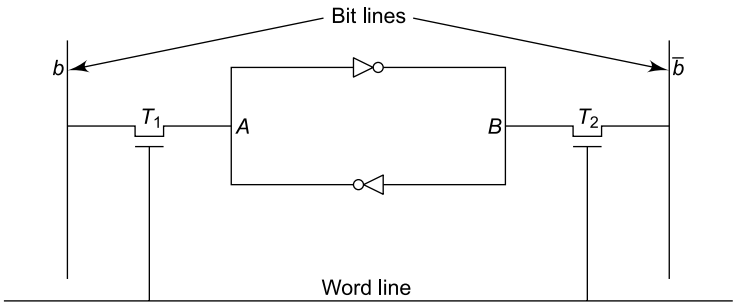


Figure 2 *A SRAM cell*

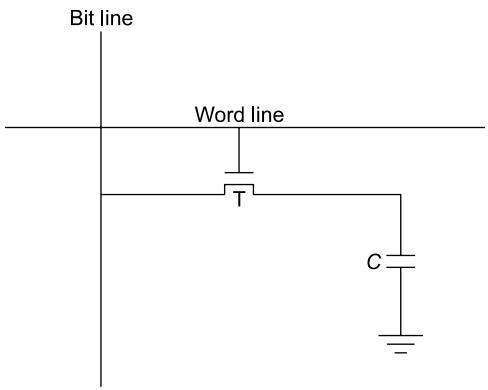


Figure 3 *A DRAM cell*

Differences between SRAM and DRAM:

1. The SRAM has lower access time, which means it is faster compared to the DRAM.
2. The SRAM requires constant power supply, which means this type of memory consumes more power; whereas, the DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
3. Due to the relatively small internal circuitry in the one-bit memory cell of DRAMs, the large storage capacity in a single DRAM memory chip is available compared to the same physical size SRAM memory chip. In other words, DRAM has high packaging density compared to the SRAM.
4. SRAM is costlier than DRAM.

4. (a) Write key features of von Neumann architecture of a computer and mention the bottlenecks.
   (b) How does Harvard architecture differ from von Neumann architecture? 2 + 1 + 2

Answer

(a) See answer of question no. 8(c) of 2007 (CS-303).
(b) See answer of question no. 2(b) of 2008 (CS-404(EI).

5. (a) What is cache mapping? Explain the difference between full associative and direct cache mapping.
   (b) What are 'write through' and 'write back' policies in cache? 1 + 2 + 2

Answer

(a) The transfer of data as a block from main memory to cache memory is referred to as a *cache mapping* process. Three types of cache mapping have been used:
   1. Associative mapping.
   2. Direct mapping.
   3. Set-associative mapping.

For second part: See answer of question number 3(a) of 2007 (CS-303).

(b) See answer of question number 3(b) of 2007 (CS-303).

6. What are the different types of interrupts? Give an example. What is programmed I/O technique? 5

Answer

There are basically three types of interrupts: external, internal or trap and software interrupts.

External interrupt These are initiated through the processors' interrupt pins by external devices. Examples include interrupts by input-output devices and console switches. External interrupts can be divided into two types—maskable and non-maskable.

Maskable interrupt The user program can enable or disable all or a few device interrupts by executing instructions EI or DI.

Non-maskable interrupt The user program cannot disable it by any instruction. Some common examples are: hardware error and power fail interrupt. This type of interrupt has higher priority than maskable interrupts.

Internal interrupt This type of interrupts are activated internally by exceptional conditions. The interrupts caused due to overflow, division by zero and execution of an illegal op-code are common examples of this category.

Software interrupt A software interrupt is initiated by executing an instruction like INT n in a program, where *n* refers to the starting address of a procedure in program. This type of interrupts is used to call the operating system. The software interrupt instructions allow to switching from user to supervisor mode.

Programmed I/O

This is the software method where the CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. In other words, the CPU polls the devices for next data transfer. This is why the programmed I/O is sometimes called *polled I/O*. Through the mid-1990s, programmed I/O was the only way that most systems ever accessed IDE/ATA hard disks.

Group-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. (a) Using Booth's algorithm multiply (–9) and (–3), when numbers are represented in 2's complement form. 9
   (b) Show how the non-restoring method is deduced from restoring division method. 4
   (c) Write down the steps of the algorithm of addition or subtraction of two floating point numbers. 2

Answer

(a) Multiplicand, $M = -9 = 10111$ and multiplier, $Q = -3 = 11101$.

	M	A	Q	Size
Initial Configuration	10111	00000	111010	5
Step-1 As $Q[0]=1$ and $Q[-1]=0$				
$A = A - M$	10111	01001	111010	–
ARS(AQ)	10111	00100	111101	4
Step-2 As $Q[0]=0$ and $Q[-1]=1$				
$A = A + M$	10111	11011	111101	–
ARS(AQ)	10111	11101	111110	3
Step-3 As $Q[0]=1$ and $Q[-1]=0$				

A = A – M	10111	00110	111110	–
ARS(AQ)	10111	00011	011111	2
Step-4 As $Q[0]=1$ and $Q[-1]=1$ ARS(AQ)	10111	00001	101111	1
Step-5 As $Q[0] = 1$ and $Q[-1] = 1$ ARS (AQ)	10111	00000	110111	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 00000 11011, which shows that the product is a positive number and the result is + 27 in decimal.

(b) In the restoring method, some extra additions are required to restore the number, when A is negative. Proper restructuring of the restoring division algorithm can eliminate that restoration step. This is known as the non-restoring division algorithm.

The three main steps in restoring division method are:

1. Shift AQ register pair to the left one position.
2. A = A – M.
3. If the sign of A is positive after the step 2, set Q[0] = 1; otherwise, set Q[0] = 0 and restore A.

Now, assume that the step 3 is performed first and then step 1 followed by step 2. Under this condition, the following two cases may arise.

Case 1: When A is positive:
Note that shifting A register to the left one position is equivalent to the computation of 2A and then subtraction. This gives the net effect on A as 2A - M.

Case 2: When A is negative:
First restore A by adding the content of M register and then shift A to the left one position. After that A will be subtracted from M register. So, all together they give rise the value of A as 2(A + M) – M = 2A + M.

Basis on these two observations, we design the non-restoring division method, where the restoration step is eliminated.

(c) The rule for the addition/subtraction operation is summarized below:

Steps:

i. Choose the number with the smaller exponent and shift its mantissa right a number of positions equal to the difference in exponents.
ii. Set the exponent of the result equal to the larger exponent.
iii. Perform addition/subtraction on the mantissas and determine the sign of the result.
iv. Normalize the result, if necessary.

8. (a) Define MIMD type parallel processing. Define speed-up of a parallel processing system. 2 + 2

(b) Show that when K jobs are processed over an N stage pipeline, the speed-up obtained is $S_p = (NK)/(N + K - 1)$ 6

(c) With the help of a neat diagram show the structure of a typical arithmetic pipeline performing $A * B + C$. 5

Answer

(a) *MIMD type parallel processing* This type processing based on the MIMD (Multiple Instruction Stream Multiple Data Stream) computers as defined by M. J. Flynn.

This category computers cover multiprocessor systems and multiple computer systems (see figure below). An MIMD computer is called *tightly coupled (or Uniform Memory Access (UMA)* if the degree of interactions among the processors is high. Otherwise, we consider them *loosely coupled (or Non-Uniform Memory Access (NUMA)*. Most commercial MIMD computers are loosely coupled.

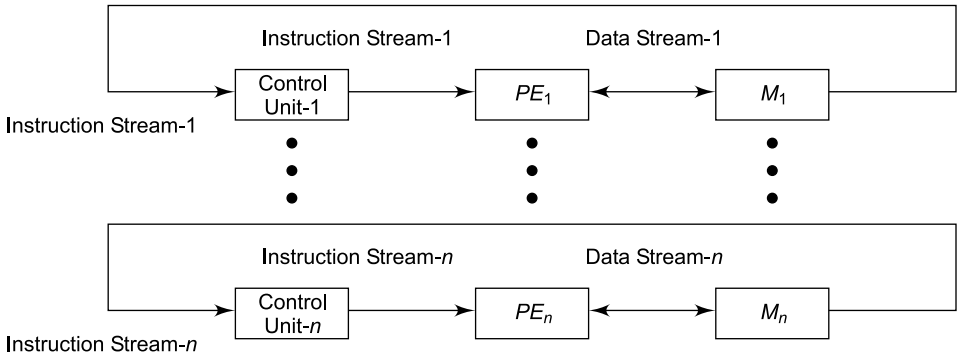


Figure 4 *MIMD computer*

Speed-up S(n) of a parallel processing system is defined as the ratio of total execution time T(1) on serial processing system to the corresponding execution time T(n) on a processing system whose degree of parallelism is n.

Thus, $S(n) = \dfrac{T(1)}{T(n)}$

(b) Speed-up is defined as

$$S_p = \frac{\text{Time to execute K tasks in N-stage non-pipeline processor}}{\text{Time to execute K tasks in N-stage pipeline processor}}$$

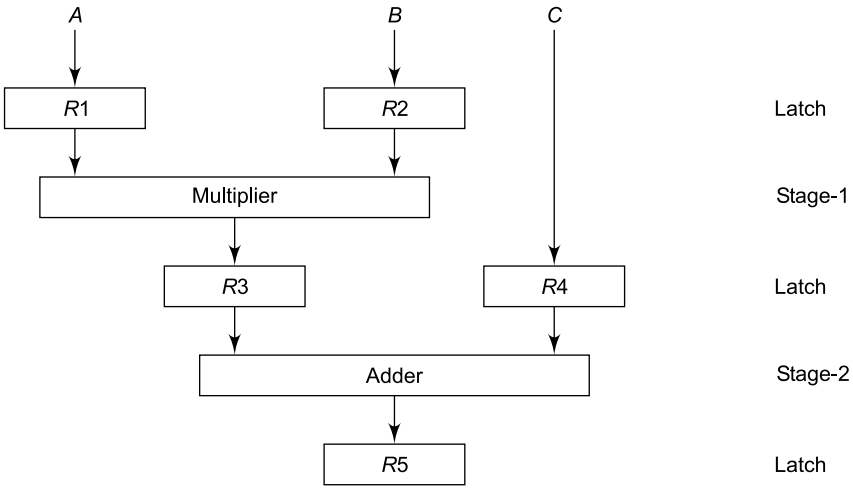
Time to execute K tasks in N-stage pipeline processor is $\tau[N + (K-1)]$ units, where N clock periods (cycles) are needed to complete the execution of the first task and remaining (K–1) tasks require (K–1) cycles. Time to execute K tasks in N-stage non-pipeline processor is $K.N.\tau$, where each task requires N cycles because no new task can enter the pipeline until the previous task finishes. The clock period of the pipeline processor is τ.

Thus $S_p = \dfrac{K \cdot N \cdot \tau}{\tau[N + (K-1)]} = \dfrac{NK}{N + (K-1)}$

(c) The sub-operations to be performed for arithmetic expression $A * B + C$ in each stage of the pipeline are as follows:

Sub-operation-1: $R1 \leftarrow A$, $R2 \leftarrow B$	Input A and B
Sub-operation-2: $R3 \leftarrow R1 * R2$, $R4 \leftarrow C$	Multiply and input C
Sub-operation-3: $R5 \leftarrow R3 + R4$	Add C to product

Five registers are loaded with new data in every clock period. The corresponding pipeline is shown below.



Pipeline processing for $A * B + C$

9. (a) Discuss the principle of carry look ahead adder and design a 4-bit CLA adder and estimate the speed enhancement with respect to ripple carry adder.
(b) Briefly state the relative advantages and disadvantages of parallel adder over serial adder.
(c) $X = (A + B) \times C$

Write down the zero address, one address and three addresses instruction for the expression. $(4 + 3) + 2 + 6$

Answer

(a) See answer of question no. 9(a) of 2007 (CS-404).
(b) See answer of question no. 9(b) of 2007 (CS-404).
(c) See answer of question no. 9(c) of 2007 (CS-404).

10. (a) Why DMA based I/O is better than other I/O techniques? 3
(b) Differentiate between isolated I/O and memory mapped I/O. 3
(c) Explain DMA based data transfer operation between memory and other peripherals. 6
(d) What is the difference between vectored and non-vectored interrupt? 3

Answer

(a) A special controlling unit called DMA controller is provided to allow transfer a block of data directly between a high-speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA).*

DMA is useful and better than other I/O techniques, because it has following advantages:

1. High speed data transfer is possible, since CPU is not involved during actual transfer, which occurs between I/O device and the main memory.
2. Parallel processing can be achieved between CPU processing and DMA controller's I/O operation.

In DMA transfer, I/O devices can directly access the main memory without intervention by the processor.

(b) 1. In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

2. The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.
3. Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.
4. Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

(c) See answer of question no. 9(a) of 2007 (CS-303).

(d) Interrupt is a special signal to the CPU generated by an external device that causes the CPU to suspend the execution of one program and start the execution of another.

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially, the set-up time is quite large.

11. Write short notes any *three* of the following: 3×5

(a) Magnetic recording
(b) Adder-subtractor circuit
(c) Stack organization
(d) Bus organization using tri-state buffer
(e) DMA
(f) Addressing modes.

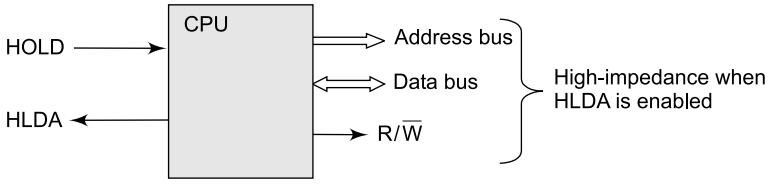
Answer

(a) See answer of question number 10(a) of 2007 (CS-303).
(b) See answer of question number 10(b) of 2007 (CS-303).
(c) See answer of question number 10(d) of 2007 (CS-303).
(d) See answer of question number 10(e) of 2007 (CS-303).

(e) *DMA processing:* To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA).*

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure below shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.



CPU bus signals for DMA transfer

To communicate with the CPU and I/O device the DMA controller needs the usual circuits of an interface. In addition to that, it needs an address register, a word count register, a status register and a set of address lines. Three registers are selected by the controller's register select (RS) line. The address register and address lines are used for direct communication with the memory. The address register is used to store the starting address of the data block to be transferred. The word count register contains the number of words that must be transferred. This register is decremented by one after each word transfer and internally tested for zero after each transfer. Between the device and memory under control of the DMA, the data transfer can be done directly. The status register contains information such as completion of DMA transfer. All registers in the DMA controller appear to the CPU as I/O interface registers. Thus, the CPU can read from or write into the DMA registers under program control via the data bus.

When executing the program for I/O transfer, the CPU first initializes the DMA controller. After that, the DMA controller starts and continues to transfer data between memory and peripheral unit until an entire block is transferred. The DMA controller is initialized by the CPU by sending the following information through the data bus:

1. The starting address of the memory blocks where data are available for read or where data are to be stored for write.
2. The number of words in the memory block (word count) to be read or written.
3. Read or write control to specify the mode of transfer.
4. A control to start the DMA transfer.

(f) See answer of question number 10(c) of 2007 (CS-303).

2009
Computer Organization
(CS-303)

Time Alloted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple Choice Type Questions)

1. Choose the correct alternatives of the following: $10 \times 1 = 10$

(i) When signed numbers are used in binary arithmetic, then which one of the following notations would have unique representation for zero?

(a) Magnitude (b) 1's complement (c) 2's complement (d) None of these

Answer

(c) 2's complement

(ii) Maximum n bit 2's complement number is

(a) 2^n (b) $2^n - 1$ (c) $2^{n-1} - 1$ (d) cannot be said

Answer

(c) $2^{n-1} - 1$

(iii) For BIOS (Basic Input/Output System) and IOCS (Input/Output Control System), which one of the following is true?

(a) BIOS and IOCS are same.

(b) BIOS controls all devices and IOCS controls only certain devices.

(c) BIOS is not a part of the operating system and IOCS is a part of the operating system.

(d) BIOS is stored in ROM and IOCS is stored in RAM.

Answer

(c) BIOS is not a part of the operating system and IOCS is a part of the operating system.

(iv) Which logic gate has the highest speed ?

(a) ECL (b) TTL (c) RTL (d) DTL

Answer

(a) ECL

(v) Booth's algorithm for computer arithmetic is used for

(a) multiplication of numbers in sign magnitude form
(b) multiplication of numbers in 2's complement form
(c) division of numbers in sign magnitude form
(d) division of numbers in 2's complement form

Answer

(b) multiplication of numbers in 2's complement form

(vi) How many RAM chips of size (256 K × 1 bit) are required to build 1 M byte memory?

(a) 32 (b) 10 (c) 8 (d) 24

Answer

(a) 32

(vii) The conversion $(FAFAFB)_{16}$ into octal form is

(a) 76767676 (b) 76575372 (c) 76737672 (d) none of these

Answer

(d) none of these

(viii) A decimal number has 30 digits. Approximately, haw many would the binary representation have?

(a) 30 (b) 32 (c) 60 (d) 90

Answer

(d) 90

(ix) The logic circuit in the ALU is

(a) entirely combinational (b) entirely sequential
(c) combinational cum sequential (d) none of these

Answer

(a) entirely combinational

(x) State True or *False:*

Adding $0110\ 1101_2$ to $1010\ 0010_2$ in 8-bit 2's complement binary will cause an overflow

(a) True (b) False

Answer

(a) True

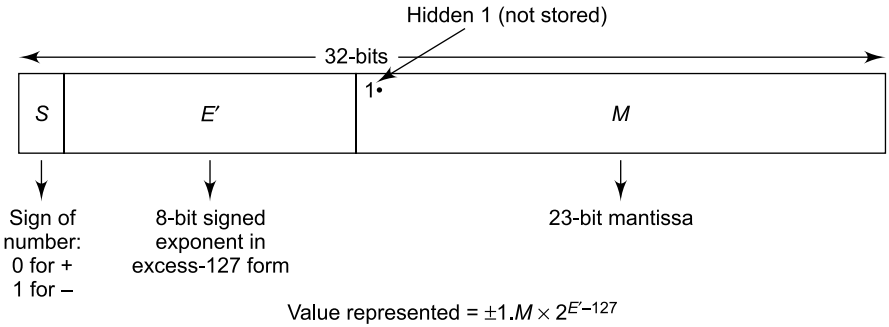
Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. (a) Briefly explain IEEE 754 standard format for floating point representation in single precision.
(b) Write $+7_{10}$ in IEEE 754 floating point representation in double precision. 3 + 2

Answer

(a) *IEEE 754 Single precision format* It is 32-bit format, in which 8-bit is for exponent, 23-bit for mantissa, 1-bit for sign of the number, as shown in figure below. Here, the implied base 2 and original signed exponent E are not stored in register. The value actually stored in the exponent field is an unsigned integer E' called *biased exponent*, which is calculated by the relation $E' = E + 127$. This is referred to as the *excess-127* format. Thus, E' is in the range $0 \leq E' \leq 255$. The end values of this range, 0 and 255, are used to represent special values. Therefore, the range of E' is $1 \leq E' \leq 254$, for normal values. This means that the actual exponent (E) is in the range $-126 \leq E \leq 127$.



(b) See answer of question no. 3(a) of 2008 (CS-404(EI)).

3. What is interrupt ? What are the differences between vectored and non-vectored interrupts? 1 + 4

Answer

Interrupt is a special signal to the CPU generated by an external device that causes the CPU to suspend the execution of one program and start the execution of another.

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially, the set-up time is quite large.

4. (a) Where does DMA mode of data transfer find its use?
   (b) What are the different types of DMA controllers and how do they differ in their functioning? 2 + 3

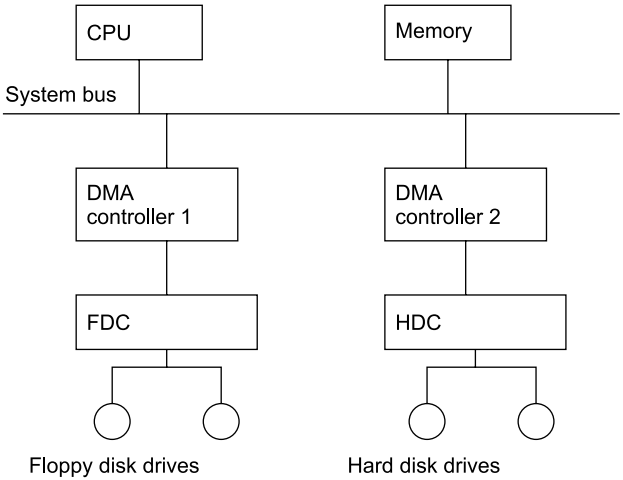
Answer

(a) To transfer large blocks of data at high speed, the DMA method is used. A special controlling unit is provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU.

(b) DMA controllers are of two types:
   – Independent DMA controller
   – DMA controller having multiple DMA-channels

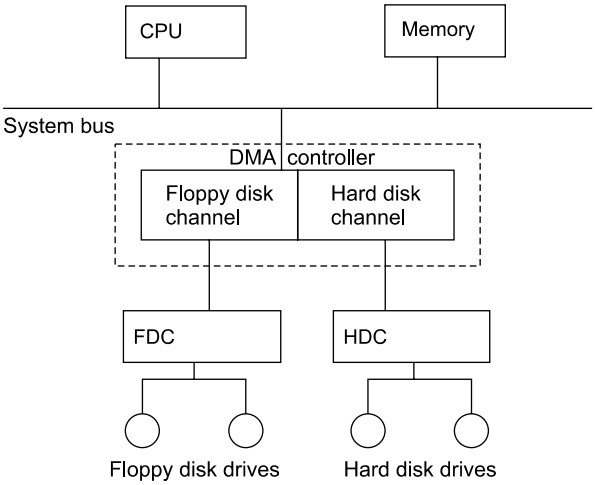
Independent DMA controller:

For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.



DMA controller having multiple DMA-channels:

In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in figure below for floppy disk controller (FDC) and hard disk controller (HDC).



5. Explain the difference between full associative and direct mapped cache mapping approaches.

Answer

See answer of question no. 3(a) of 2007 (CS-303).

6. Compare and contrast RISC and CISC architecture.

Answer

See answer of question no. 6 of 2007 (CS-404).

Group-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. (a) Give the Booth's algorithm for multiplication of signed 2's complement numbers in flowchart and explain.
   (b) Multiply –5 and –3 using Booth's algorithm.
   (c) What is von Neumann architecture? What is von Neumann bottleneck?
   (d) What is the necessity of guard bits? $5 + 4 + 4 + 2$

Answer

(a) See answer of question no. 8(a) of 2007 (CS-303).
(b) See answer of question no. 8(b) of 2007 (CS-303).
(c) See answer of question no. 8(c) of 2007 (CS-303).
(d) See answer of question no. 7(d) of 2008 (CS-303).

8. (a) Define "latency time" of a memory.
   (b) Can a ROM be also a RAM? Justify your answer.
   (c) Explain the memory hierarchy pyramid, also explain the relationship of cost, speed and capacity.
   (d) A hierarchical cache-main memory subsystem has the following specification :
      (i) Cache access time of 160 ns
      (ii) Main memory access time 960 ns
      (iii) Hit ratio of cache memory is 0.9

   Calculate the following:
      (a) Average access time of the memory system
      (b) Efficiency of the memory system $1 + 3 + 5 + (3 + 3)$

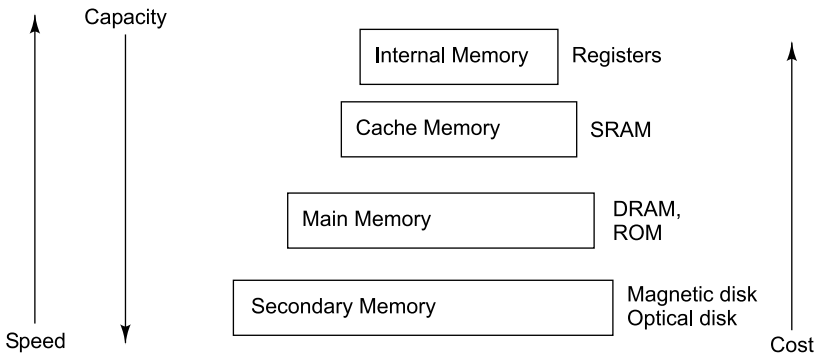
Answer

(a) The latency time is measured in terms of two parameters: access time, t_A and cycle time, t_C. Some times, latency is measured in terms of access time and some times in terms of cycle time. To perform a read operation, first the address of the location is sent to memory followed by the 'read' control signal. The memory decodes the address, selects the location and reads out the contents of the location. The access time is the time taken by the memory to complete a read operation from the moment of receiving the 'read' control signal. Generally, access time for read and write is equal. Suppose two successive memory read operations have to be performed. During the first read operation, the information read from memory is available after the access time. This

data can be immediately used by CPU. However, the memory is still busy with some internal operation for some more time called recovery time, t_R. During this time, another memory access, read or write cannot be initiated. Only after the recovery time, next operation can be started. The cycle time is the total time including access time and recovery time: $t_C = t_A + t_R$. This recovery time varies with memory technology.

(b) Yes. The ROM is also random access memory. In random mode of access, any location of the memory can be accessed randomly.
Also, ROM memories are available in variety of modes, such as PROM, EPROM, and EEPROM, in which both read and write operations can be performed.

(c) The total memory capacity of a computer can be considered as being a hierarchy of components. The memory hierarchy system consists of all storage devices used in a computer system and are broadly divided into following four groups, shown in figure below.
- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory



Memory hierarchy

Secondary Memory: The slow-speed and low-cost devices that provide backup storage are called secondary memory. The most commonly used secondary memories are magnetic disks, such as hard disk, floppy disk and magnetic tapes. This type of memory is used for storing all programs and data, as this is used in bulk size. When a program not residing in main memory is needed to execute, it is transferred from secondary memory to main memory. Programs not currently needed in main memory (in other words, the programs are not currently executed by the processor) are transferred into secondary memory to provide space for currently used programs and data.

Main Memory: This is the memory that communicates directly with CPU. Only programs and data currently needed by the CPU for execution reside in the main memory. Main memory occupies central position in hierarchy by being able to communicate directly with CPU and with secondary memory devices through an I/O processor.

Cache Memory: This is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as a buffer between the CPU and main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of computer.

Internal memory: This memory refers to the high-speed registers used inside the CPU. These registers hold temporary results when a computation is in progress. There is no speed disparity between these registers and the CPU because they are fabricated with the same technology. However, since registers are very expensive, only a few registers are used as internal memory.

(d) Given,

Cache access time, t_c = 160 ns

Main memory access time, t_m = 960 ns

Hit ratio, h = 0.9

(i) Now we have,

Average access time of the memory system, $t_{av} = ht_c + (1 - h)(t_c + t_m)$

$= 0.9 \times 160 + 0.1 \times (160 + 960)$

$= 256$ ns

(ii) Efficiency of the memory system $= (t_c / t_{av}) \times 100\ \%$

$= (160/256) \times 100\ \%$

$= 62.5\ \%$

9. (a) What is locality of reference? Explain the concept of cache memory with it.
   (b) Briefly explain write-through and write-back policies.
   (c) State L1 and L2 cache policies with suitable figure.
   (d) Discuss the role of OS.
   (e) How many 256 × 4 RAM chips are needed to provide a memory capacity of 2048 bytes? Show also the corresponding interconnection diagram. 2 + 2 + 2 + 2 + 7

Answer

(a) Analysis of a large number of typical programs shows that the CPU references to main memory during some time period tend to be confined within a few localized areas in memory. In other words, few instructions in the localized areas in memory are executed repeatedly for some time duration and other instructions are accessed infrequently. This phenomenon is known as the property of *locality of reference*.

Consider that when a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitutes the loop. Thus loop tends to localize the references to memory for fetching the instructions. If this program loop is placed in fast cache memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Because we know that the cache memory's speed is almost same as that of CPU.

(b) *Write-Through Policy:* This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated.

Advantage: The main memory always contains the same data as the cache.

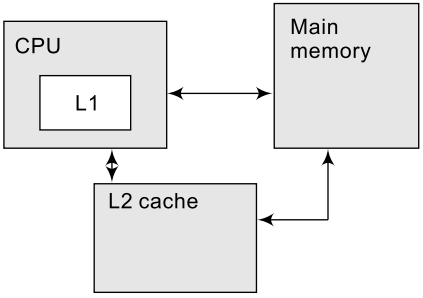
Disadvantage: It is a slow process, since each time main memory needs to be accessed.

Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified or dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set. The philosophy of this method is based on the fact that during a write operation, the word residing in cache may be accessed several times (temporal locality of reference).

Advantage: The method is faster than write-through, because this method reduces the number of references to main memory.

Disadvantage: This method may encounter the problem of inconsistency due to two different copies of the same data, one in cache and other in main memory.

(c) Cache memory has been used in two or even three levels in modern-day computers. For two level cache organization, level 1, simply L1 is used as on-chip cache and level 2, simply L2 is used as off-chip cache. This is shown in following figure. Obviously, L1 cache access is faster than L2, but capacity of L2 is more than that of L1. Since L1 is embedded in the processor, processor first accesses L1 then L2 in case of miss on L1.



(d) See answer of question no. 7(b) of 2009 (CS-404).

(e) See answer of question no. 10(c) of 2007 (CS-404).

10. (a) A 32-bit floating-point binary number has a bit plus a sign for the exponent. The mantissa is assumed to be a normalized fraction. Numbers in the mantissa and exponent are in signed-magnitude representation. What are the largest and smallest positive quantities that can be represented, excluding zero ? Explain with example.

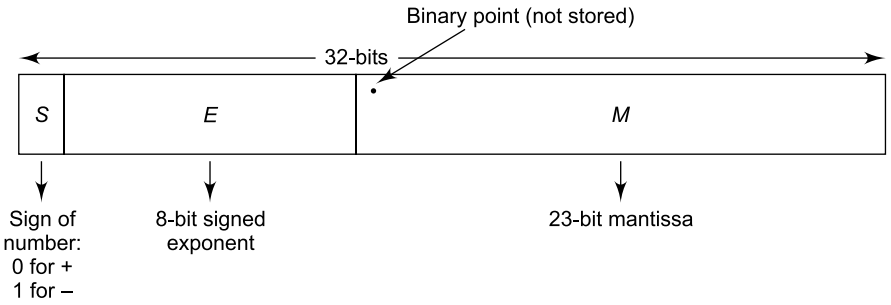
(b) Explain with diagrams, Serial and Parallel Adders.

(c) ADD $A + B$, where $A = 63.11236589 \times 10^{15}$ and $B = 0.002365991 \times 10^{-29}$.

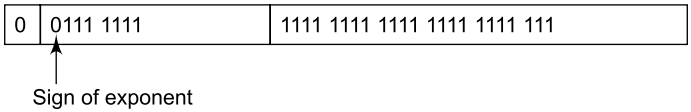
4 + 4 + 4 + 3

Answer

(a) As some information are missing in the problem, we assume that the format of the 32-bit floating-point representation as



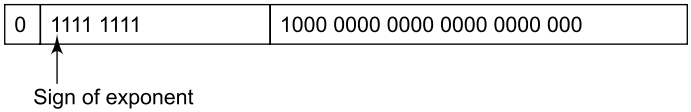
The maximum value of the number which can be stored is shown in the figure below:



Therefore, the maximum value = $(1 - 2^{-23}) \times 2^{127}$

$\approx 2^{127}$

The minimum value of the number which can be stored is shown in the figure below:



Therefore, the minimum value = $(2^{-1}) \times 2^{-127}$

$\approx 2^{128}$

(a) A binary adder is a digital circuit that generates the arithmetic sum of two binary numbers of any lengths. The binary adder is basically constructed with full adders. Binary adders are of two types:

1. Serial Adder
2. Parallel Adder.

Serial Adder: A serial adder is an adder, which performs the addition of two binary numbers serially bit by bit starting with lsb. Addition of one bit position takes one clock cycle. The circuit for this adder is shown in figure below. The operands are provided bit by bit starting with lsbs. Thus, for an *n*-bit serial adder, *n* clock cycles are needed to complete the n-bit numbers' addition. At each cycle, the carry produced by a bit position should be stored in a D-flip-flop and it is given as input during the next cycle through carry-in. Therefore, serial adder is a sequential circuit.

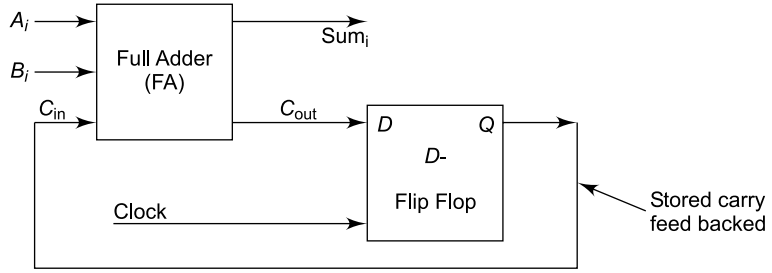


Figure *A Serial Adder*

Advantage: The serial adder circuit is small and hence, it is very inexpensive irrespective of the number of bits to be added.

Disadvantage: The serial is very slow since it takes *n* clock cycles for addition of *n*-bit numbers.

Parallel Adder: A parallel adder is an adder, which adds all bits of two numbers in one clock cycle. It has separate adder circuit for each bit. Therefore, to add two *n*-bit numbers, parallel adder needs *n* separate adder circuits.

For addition of two *n*-bit numbers, *n* full adders (FAs) are required. Each full adder's carry output will be the input of the next higher bit full adder. Each full adder performs addition for same position bits of two numbers. An *n*-bit adder circuit is shown in the figure below.

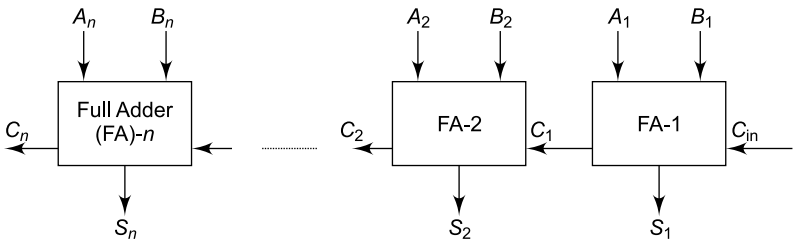


Figure *An n-bit Parallel Adder*

The addition time is decided by the delay introduced by the carry. In worst case, the carry from the first full adder stage has to propagate through all the full adder stages. This is why this type adder is called *Carry-Propagate Adder (CPA)*. Therefore, the maximum propagation delay for n-bit adder is D x n, where D is the time delay for each full adder stage and n is the number of bits in each operand.

There are basically two types of parallel adders, depending on the way of carry generation.

(a) Carry-Propagate Adder (CPA) or Ripple Carry Adder (RCA)

(b) Carry Look-ahead Adder (CLA).

Advantage: This adder, being a combinational circuit, is faster than serial adder. In one clock period all bits of two numbers are added.

Disadvantages:

(a) The addition delay becomes large, if the size of numbers to be added is increased.

(b) The hardware cost is more than that of serial adder. Because, number of full adders needed is equal to the number of bits in operands.

(c) $A = 63.11236589 \times 10^{15}$ and $B = 0.002365991 \times 10^{-29}$

We have to perform $A + B$.

The rule for the ADD operation of two floating-point numbers is summarized below:

Steps

1. Choose the number with the smaller exponent and shift its mantissa right a number of positions equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition on the mantissas and determine the sign of the result.
4. Normalize the result, if necessary.

Therefore, the mantissa 0.002365991 is to be shifted right 44 positions (by step-1).
So, the modified mantissa is:
0.0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 002365991
Now, we have to add this shifted mantissa with the mantissa in no. A (by step -3).
Therefore, we get the resultant mantissa as
63.1123 6589 0000 0000 0000 0000 0000 0000 0000 0000 0000 002365991
As this number is normalized one, we get the final result as
63.1123 6589 0000 0000 0000 0000 0000 0000 0000 0000 0000 002365991 x 10^{15}

11. Write short notes on any *three of* the following: 3×5
    (a) Pipeline hazards
    (b) Adder–subtractor circuit
    (c) Data flow architecture
    (d) Bus organization using tri-state buffer
    (e) Virtual address to real address mapping.

Answer

(a) *Pipeline hazards*: Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. The instruction is said to be *stalled*. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall.

Types of pipeline hazards are:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards:

They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Solution of control hazards:

In order to cope with the adverse effects of branch instructions, an important technique called *prefetching* is used. *Prefetching technique* states that: Instruction words ahead of the one currently being decoded in the instruction-decoding (ID) stage are fetched from the memory system before the ID stage requests them.

Structural hazards

Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Example: Instruction ADD R4, X fetches operand X from memory in the OF stage at 3rd clock period. The memory doesn't accept another access during that period. For this, (i+2)th instruction cannot be initiated at 3rd clock period to fetch the instruction from memory. Thus, one clock cycle is stalled in the pipeline for all subsequent instructions. This is illustrated next.

Clock cycles →	1	2	3	4	5	6	7	8	9	10	11	12	13
ADD R4,X	IF	ID	OF	EX	WB								
Instr. i+1		IF	ID	OF	EX	WB							
Instr. i+2			stall	IF	ID	OF	EX	WB					
Instr. i+3					IF	ID	OF	EX	WB				

Penalty: 1 cycle.

Structural hazard in instruction pipeline

Solution of structural hazards:

Certain resources are duplicated in order to avoid structural hazards. Functional units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time. A classical way to avoid hazards at memory access is by providing separate data and instruction caches.

Data hazards

Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard.*

Example: We have two instructions, I1 and I2. In a pipeline the execution of I2 can start before I1 has terminated. If in a certain stage of the pipeline, I2 needs the result produced by I1, but this result has not yet been generated, we have a data hazard.

According to various data update patterns in instruction pipeline, there are three classes of data hazards exist:

- Write After Read (WAR) hazards
- Read After Write (RAW) hazards
- Write After Write (WAW) hazards

Solution of data hazards:

The system must resolve the interlock situation when a hazard is detected. Consider the sequence of instructions {… I, I + 1, …, J, J + 1, …} in which a hazard has been detected between the current instruction J and a previous instruction I. This hazardous situation can be resolved in one of the two following ways:

- One simple solution is to stall the pipeline and to ignore the execution of instructions J, J + 1, …, down the pipeline until the instruction I has passed the point of resource conflict.
- A more advanced approach is to ignore only instruction J and continue the flow of instructions J+1, J+2,…, down the pipeline. However, the potential hazards due to the suspension of J must be continuously tested as instructions J+1, J+2,… execute prior to J. Thus, multilevel

of hazard detection may be encountered, which requires much more complex control policies to resolve such multilevel of hazards.

(b) See answer of question no. 10(b) of 2007 (CS-303).

(c) *Data flow architecture*:

Data flow computers are based on the concept of *data-driven* computation, which is drastically different from the operation of a conventional von Neumann machine. The fundamental difference is that instruction execution in a conventional computer is under program-flow control, whereas that in a data flow computer is driven by the data (operand) availability.

Jack Dennis (1974) of MIT has identified three basic issues towards development of an ideal architecture for future computers. The first is to achieve a high performance/cost ratio; the second is to match the ratio with technological progress; and the third is to offer better programmability in application areas. The data flow model offers an approach to meet these demands. The recent progress in the VLSI microelectronic area has provided the technological basis for developing data flow computers.

The concept of data flow computing is illustrated by the control of computation sequences for the statement $a = (b + 1) * (b - c)$ in the following figure. In a data flow computing environment, instructions are activated by the availability of data tokens as indicated by the ( ) in the figure. Data flow programs are represented by directed graphs, which show the flow of data between instructions. Each instruction consists of an operator, one or two operands and one or more destinations to which the result (data token) will be sent.

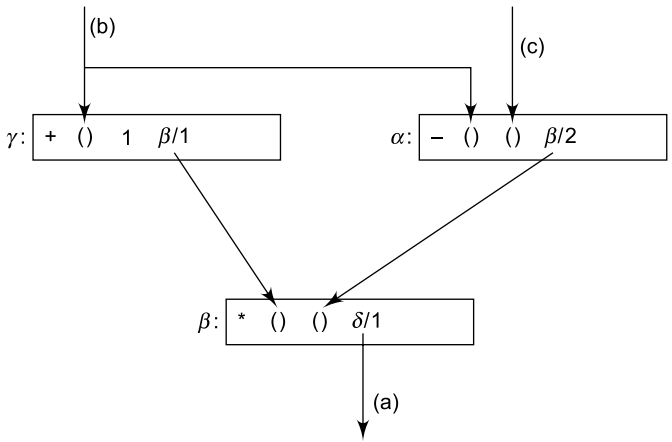


Figure *Instruction execution in a data flow computer for the computation of a = (b + 1) * (b – c) by direct data forwarding*

Major features are identified below for the data flow computers:

1. Data flow computers have a *data-driven* organization that is characterized by a passive examine stage. Instructions are examined to reveal the operand availability; upon which they are executed immediately if the functional units are available.
2. Intermediate or final results are passed directly as data token between instructions.
3. There is no concept of shared data storage as embodied in notion of a variable.

4. Program sequencing is constrained only by data dependency among instructions.
5. The data-driven concept means asynchrony, which means that many instructions can be executed simultaneously and asynchronously.
6. A high degree of implicit parallelism is expected in a data flow computer.

(d) See answer of question no. 10(e) of 2007 (CS-303).

(e) *Virtual address to real address mapping*:

When a program needs to be executed, the CPU would generate addresses, called *logical* addresses. The corresponding addresses in the physical memory, as occupied by the executing program, are called *physical* addresses. The set of all logical addresses generated by the CPU or program is called *logical-address space* and the set of all physical addresses corresponding to these logical addresses is called *physical-address space*. The memory-management unit (MMU) maps each logical address to a physical address during program execution. The figure below illustrates this mapping method, which uses a special register called base register or relocation register. The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

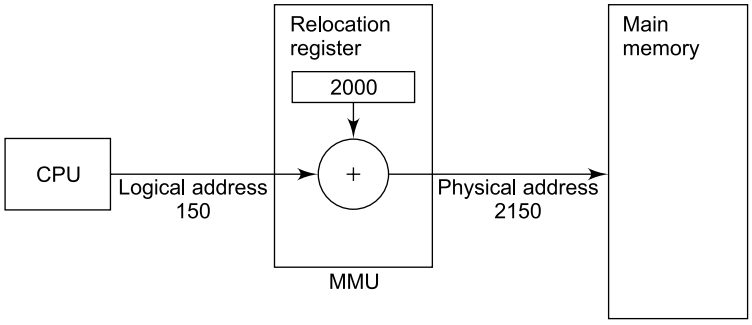


Figure *A simple memory-management scheme*

A virtual memory system may be configured in one of the following ways:

1. Paging technique
2. Segmentation technique

Paging:

Paging is a non-contiguous memory allocation method. In other words, the program is divided into small blocks in paging and these blocks are loaded into else where in main memory. In paging, the virtual address space is divided into equal size blocks called *pages* and the physical (main) memory is divided into equal size blocks called *frames*. The size of a page and size of a frame are equal. The size of a page or a frame is dependent on the operating system and is generally 4 KB.

In paging, operating system maintains a data structure called *page table*, which is used for mapping from logical address to physical address. The page table generally contains two fields, one is page number and other is frame number. The table specifies the information that which page would be

mapped to which frame. Each operating system has its own way of maintaining the page tables; most allocate a page table for each program.

Segmentation:

Segmentation is a memory management scheme that supports the user view of memory. A logical-address space of a program is a collection of segments. A *segment* is defined as a logical grouping of instructions, such as subroutine, array or data area. Each segment has a name and a length. The address of the segment specifies both segment name and offset within the segment. For simplicity of implementation, segments are referred to by a segment number rather than by a segment name. Thus, a logical address consists of two tuples: *(segment number (s), offset (d)*).

The mapping of logical address to corresponding physical address is done using *segment table*. Each entry of the segment table has a segment *base* and a segment *limit*. The segment base indicates the starting physical address where the segment resides in main memory and the segment limit specifies the length of the segment.

2010
Computer Architecture and Organization (EC-503)

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) The logic circuit in ALU is
(a) entirely combinational (b) entirely sequential
(c) combinational cum sequential (d) none of these

Answer
(a) entirely combinational

(ii) In a microprocessor the address of the next instruction to be executed is stored in
(a) stack pointer (b) address latch
(c) program counter register (d) general purpose

Answer
(c) program counter register

(iii) Physical memory broken down into groups of equal size is called
(a) page (b) tag (c) block (d) index

Answer
(c) block

(iv) The basic principle of a Von Neumann computer is
(a) storing program and data in separate memory
(b) using pipeline concept
(c) storing both program and data in the same memory
(d) using a large number of registers

Answer

(c) storing both program and data in the same memory

(v) The principle of locality justifies the use of

(a) interrupts (b) DMA (c) polling (d) cache memory

Answer

(d) cache memory

(vi) Instruction cycle is

(a) fetch-decode-execution (b) fetch-execution decode

(c) decode-fetch-execution (d) none of these

Answer

(a) fetch-decode-execution

(vii) Associative memory is a

(a) pointer addressable memory (b) very cheap memory

(c) content addressable memory (d) slow memory

Answer

(c) content addressable memory

(viii) Conversion of $(FAFAFA)_{16}$ into octal form is

(a) 76767676 (b) 76575372 (c) 76737672 (d) 76727672

Answer

(b) 76575372

(ix) The technique of placing software in a ROM semiconductor chip is called

(a) PROM (b) EPROM (c) Firmware (d) Microprocessor

Answer

(c) Firmware

(x) How many address bits required for a 512×4 memory?

(a) 512 (b) 4 (c) 9 (d) $A_0 - A_6$

Answer

(c) 9

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. Explain the reading and writing operations of a basic static MOS cell.

Answer

One SRAM cell using CMOS is shown in the Figure 1. Four transistors (T_3, T_4, T_5 and T_6) are cross connected in such a way that they produce a stable state. In the state 1, the voltage at the point A is maintained high and voltage at point at B is low by keeping transistors T_3 and T_6 on (i.e. closed), while T_4 and T_5 off (i.e. open). Similarly, in state 0, the voltage at A is low and at point B is high by keeping transistors T_3 and T_6 off, while T_4 and T_5 on. Both these states are stable as long as the power is applied on it. Thus, for state 1, if T_1 and T_2 are turned on (closed), bit lines b and b' will have high and low signals, respectively.

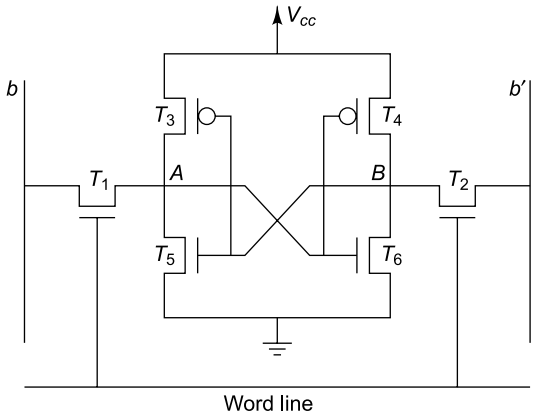


Figure 1 *A CMOS SRAM cell*

Read Operation:

For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation:

Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.

3. Give Booth's algorithm for multiplication of signed 2's complement number in flowchart and explain.

Answer

The algorithm inspects two lower-order multiplier bits at a time to take the next step of action. The algorithm is described by the flowchart in the figure 2. A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contain the product. Ignore the right-end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier.

4. Explain the concept of virtual memory.

Answer

Parts of programs and data are brought into main memory from secondary memory, as the CPU needs them. Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, although which may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual (logical) address is mapped to a physical address in the main memory. This mapping is done during run-time and is performed by a hardware device called *Memory-Management Unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

The virtual memory makes the task of programming much easier, because the programmer no longer needs to bother about the amount of physical memory available. For example, suppose a program size is 18 MB and the available user part of the main memory is 15 MB (other part of the main memory is occupied by the operating system). First, 15 MB of the program is loaded into main memory and then remaining 3 MB is still in the secondary memory. When the remaining 3 MB code is needed for execution, swap out 3 MB code from main memory to secondary memory and swap in new 3 MB code from secondary memory to main memory.

The advantage of virtual memory is efficient utilization of main memory, because the larger size program is divided into blocks and partially each block is loaded in the main memory whenever it is required. Thus multiple programs can be executed simultaneously. The technique of virtual memory has other advantages of efficient CPU utilization and improved throughput.

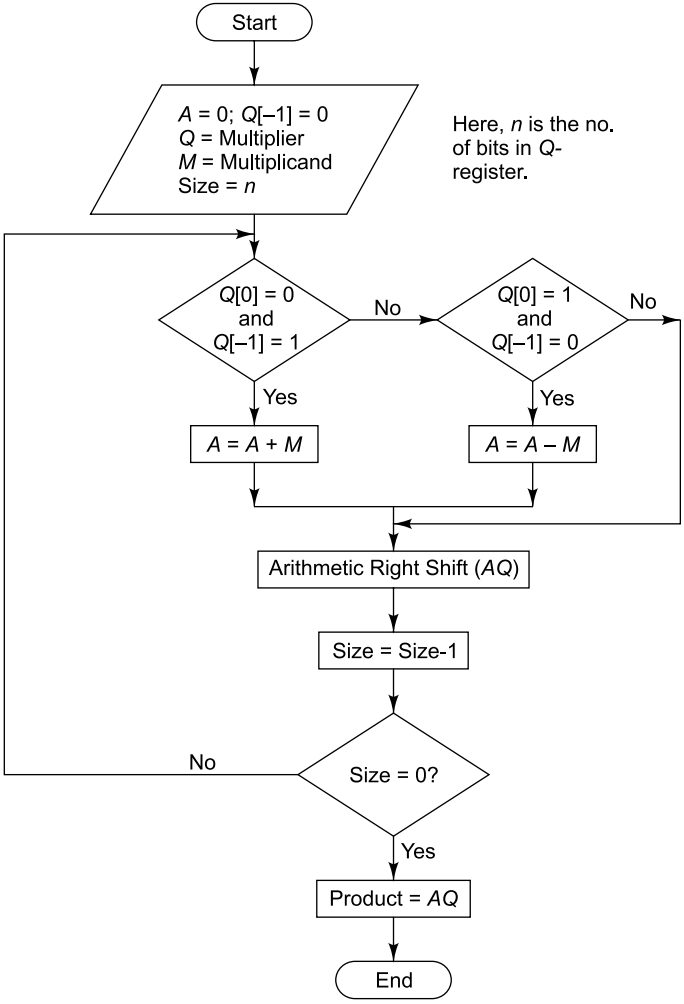


Figure 2 *Booth's multiplication algorithm*

The memory-management unit (MMU) maps each logical address to a physical address during program execution. Figure 3 illustrates this mapping method, which uses a special register called base register or relocation register. The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

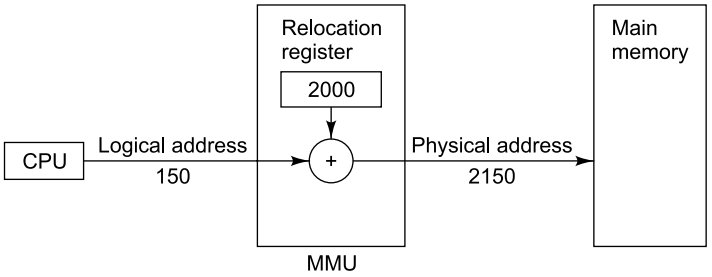


Figure 3 *A simple memory-management scheme*

A virtual memory system may be configured in one of the following ways:

(i) Paging technique
(ii) Segmentation technique

5. What is Von Neumann architecture? What is Von Neumann bottleneck?

Answer

In 1946, Von Neumann and his colleagues began the design of a new stored-program computer, now referred to as the IAS computer, at the Institute for Advanced Studies, Princeton. Nearly, all modern computers still use this *stored-program* concept. This concept has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

Each instruction contains only one memory address and has the format:

OPCODE ADDRESS

The 8-bit op-code specifies the operation to be performed by the CPU and 12-bit address specifies the operand's memory address. Thus length of each instruction is 20 bits.

Von-Neumann Bottleneck:

One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck. This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required

data-word to come. Another way to reduce the problem is by using special type computers known as *Reduced Instruction Set Computers (RISC)*. This class of computers generally uses a large number of registers, through which the most of the instructions are executed. This computer usually limits access to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

6. (a) What are the widths of data bus and address bus for 4096 × 8 memory?
(b) What do you mean by program status word? 2 + 3

Answer

(a) Given memory size is 4096 × 8. The width of data bus = 8-bit and that of address bus = 12-bit (because $4096 = 2^{12}$).

(b) *Program status word (PSW):* It is a special-purpose register that holds the condition code flags and other information that describe the status of the currently executing program. This register is also known as status register. Generally, there are five flags (i.e., status bits), namely, Carry (CY), Parity (P), Auxiliary Carry (AC), Zero (Z) and Sign (S) flags. The processor uses these flags to test data conditions.

Group-C
(Long-Answer Type Questions)

Answer any *three of* the following. 3 × 15 = 45

7. (a) Compare parallel adder with serial adder.
(b) Briefly describe Carry-Look-Ahead adder.
(c) Multiply –5 by –3 using Booth's algorithm. 4 + 6 + 5

Answer

(a)

Parallel Adder	*Serial Adder*
1. This adder is a combinational circuit, which adds all bits of two numbers in one clock cycle.	1. This adder is a sequential circuit, which performs the addition of two binary numbers serially bit by bit starting with lsb.
2. This adder, being a combinational circuit, is faster than serial adder. In one clock period all bits of two numbers are added.	2. The serial adder is very slow since it takes *n* clock cycles for addition of *n*-bit numbers.
3. The hardware cost is more than that of serial adder. Because, number of adder blocks needed is equal to the number of bits in operands.	3. The serial adder circuit is small and hence, it is very inexpensive irrespective of the number of bits to be added.

(b) A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1}=A_iB_i + (A_i+B_i)C_i$$

This result can be rewritten as:

$$C_{i+1}= G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in the figure 4. The sum bit S_i is = $A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

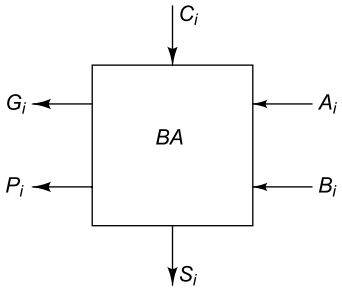


Figure 4 *Basic adder*

For example, we want to design a 4-bit CLA, for which four carries C_1, C_2, C_3 and C_4 are to be generated. Using equation number (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2$$
$$C_4 = G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$
$$= G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$
$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$
$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \quad (5)$$

The Eqs (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in the figure 5.

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

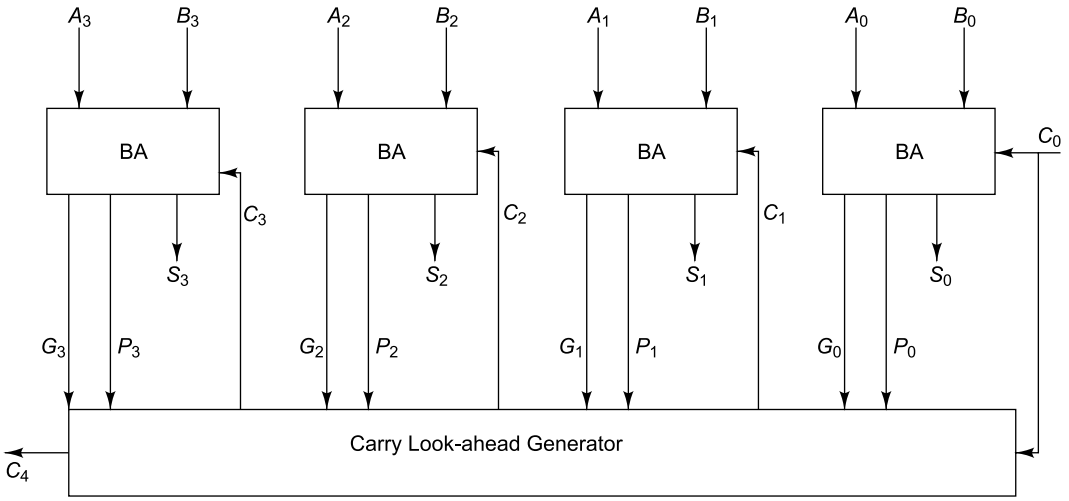


Figure 5 *4-bit Carry Look-ahead Adder (CLA)*

(c) $M = -5 = 1011$ and $Q = -3 = 1101$.

	M	A	Q	Size
Initial Configuration	1011	0000	1101 0	4
Step-1				
As $Q[0] = 1$and $Q[-1] = 0$				
$A = A - M$	1011	0101	1101 0	–
And ARS(AQ)	1011	0010	1110 1	3
Step-2				
As $Q[0] = 0$ and $Q[-1] = 1$				
$A = A + M$	1011	1101	1110 1	–
ARS(AQ)	1011	1110	1111 0	2
Step-3				
As $Q[0] = 1$ and $Q[-1] = 0$				
$A = A - M$	1011	0011	1111 0	–
ARS(AQ)	1011	0001	1111 1	1
Step-4				
As $Q[0] = 1$ and $Q[-1] = 1$				
ARS(AQ)	1011	0000	1111 1	0

Since the Size register becomes 0, the algorithm is terminated and the product is = AQ = 0000 1111, which shows that the product is a positive number. The result is 15 in decimal.

8. (a) What is pipelining?
(b) What are speed-up, throughput and efficiency of a pipelined architecture?
(c) Describe pipeline hazards.
(d) What do you mean by paging?
(e) What are instruction pipeline and arithmetic pipeline? 2+3+5+2+3

Answer

(a) *Pipelining* is a technique of decomposing a sequential task into subtasks, with each subtask being executed in a special dedicated stage (segment) that operates concurrently with all other stages. Each stage performs partial processing dictated by the way the task is partitioned. Result obtained from a stage is transferred to the next stage in the pipeline. The final result is obtained after the instruction has passed through all the stages. All stages are synchronized by a common clock. Stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high-speed interface latches (i.e., collection of registers). The Figure 6 shows the pipeline concept with k stages.

(b) *Speed-up*: It is defined as

$$S_k = \frac{\text{Time to execute } n \text{ tasks in } k\text{-stage non-pipeline processor}}{\text{Time to execute } n \text{ tasks in } k\text{-stage pipeline processor}}$$

$$= \frac{n.k.\tau}{\tau[k+(n-1)]} \quad \text{where, } \tau = \text{clock period of the pipeline processor.}$$

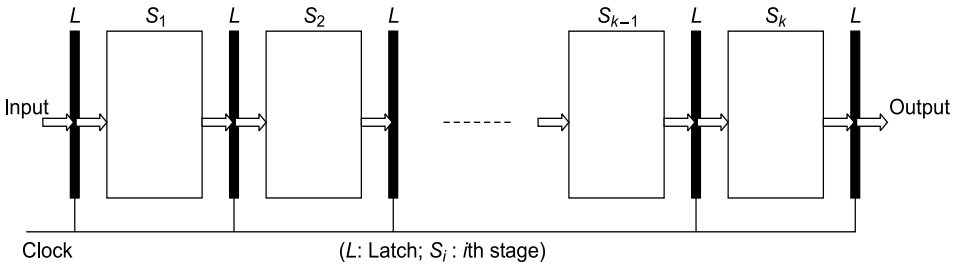


Figure 6 *Concept of pipelining*

Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n–1)]$ units, where k clock periods (cycles) are needed to complete the execution of the first task and remaining $(n – 1)$ tasks require $(n – 1)$ cycles. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can enter the pipeline until the previous task finishes.

It can be noted that the maximum speed-up is k, for $n >> k$. But this maximum speed-up is never fully achievable because of data dependencies between instructions, interrupts, program branches, etc.

Efficiency: To define it, we need to define another term 'time-space span'. It is the product (area) of a time interval and a stage space in the space-time diagram. A given time-space span can be in either a busy state or an idle state, but not both.

The *efficiency* of a linear pipeline is measured by the percentage of busy time-space spans over the total time-space span, which equals the sum of all busy and idle time-space spans. Let n, k, τ be the number of tasks (instructions), the number of stages and the clock period of a linear pipeline, respectively. Then the efficiency is defined by

$$\eta = \frac{n.k.\tau}{k.[k.\tau + (n-1).\tau]} = \frac{n}{k + (n-1)}$$

Note that $\eta \to 1$ (i.e., 100%) as $n \to \infty$. This means that the larger the number of tasks flowing through the pipeline, the better is its efficiency. For the same reason as speed-up, this ideal efficiency is achievable.

Throughput: The number of tasks that can be completed by a pipeline per unit time is called its throughput. Mathematically, it is defined as

$$\omega = \frac{n}{k.\tau + (n-1).\tau} = \frac{\eta}{\tau}$$

Note that in ideal case, $\omega = 1/\tau = f$, frequency, when $\eta \to 1$. This means that the maximum throughput of a linear pipeline is equal to its frequency, which corresponds to one output result per clock period.

(c) Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. There are three types of pipeline hazards:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards:

They arise from the pipelining of branches and other instructions that change the content of program counter (PC) register.

Structural hazards:

Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Data hazards:

Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard.*

(d) Paging is a non-contiguous memory allocation method. In other words, the program is divided into small blocks in paging and these blocks are loaded into else where in main memory. In paging, the virtual address space is divided into equal size blocks called *pages* and the physical (main) memory is divided into equal size blocks called *frames*. The size of a page and size of a frame are equal. The size of a page or a frame is dependent on the operating system and is generally 4 KB.

In paging, operating system maintains a data structure called *page table*, which is used for mapping from logical address to physical address. The page table generally contains two fields, one is page number and other is frame number. The table specifies the information that which page would be mapped to which frame. Each operating system has its own way of maintaining the page tables; most allocate a page table for each program.

(e) *Arithmetic pipeline*: An arithmetic pipeline divides an arithmetic operation, such as a multiply, into multiple arithmetic steps each of which is executed one-by-one in different arithmetic stages in the ALU. Examples include 4-stage pipeline used in Star-100, 8-stage pipeline used in TI-ASC.

Instruction pipeline: The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions. All high-performance computers are now equipped with this pipeline.

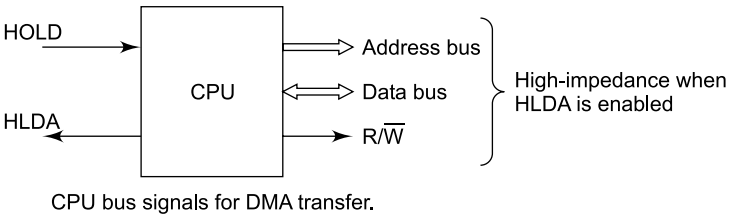
9. (a) Explain the basic Direct Memory Access (DMA) operation for transfer of data bytes between memory and peripherals.
   (b) Give the main reason why DMA based I/O is better in some circumstances than interrupt driven I/O.
   (c) What is programmed I/O technique? Why is it not very useful?
   (d) According to the following information, determine size of the subfields (in bits) in the address for Direct Mapping and Set Associative Mapping cache schemes:
   - We have 256 MB main memory and 1 MB cache memory
   - The address space of the processer is 256 MB
   - The block size is 128 bytes
   - There are 8 blocks in a cache set

   5 + 3 + 3 + 4

Answer

(a) DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure 7 shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the



CPU bus signals for DMA transfer.

Figure 7 *CPU bus signals for DMA transfer*

execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external

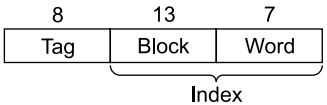
DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.

(b) To transfer large blocks of data at high speed, DMA method is used. A special DMA controller is provided to allow transfer a block of data directly between a high-speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. The data transmission cannot be stopped or slowed down until an entire block is transferred. This mode of DMA transfer is known as burst transfer.

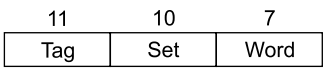
(c) This is the software method where CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU.

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. Generally the CPU is 5-7 times faster than an I/O device. Thus, the difference in data transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

(d) The address space of the processor is 256 MB. So, the processor generates an address of 28-bit.
The cache memory size = 1 MB
Therefore, the size of index field of cache = 20-bit (1MB = 2^{20})
The tag-field uses 28 – 20 = 8 bits.
The number of blocks in cache = Size of cache / size of a block = $2^{20}/2^7$ = 8192.
Therefore the number of bits required to select each block = 13 (since 8192 = 2^{13})
The size of each block is 128 bytes.
So, the number of bits required to select a word (byte) in a block = 7.
Thus, the address format for direct mapped cache is as follows:



The number of blocks in a set is = 8
Number of bits required to select a block in a set is = 3 (because 8 = 2^3).
Number of sets in the set-associative cache is = 8192/8 = 1024.
To select each set, number of bits required is = 10 (because 1024 = 2^{10}).
Therefore, tag field requires (28 – (10 + 7)) = 11 bits.
Thus, the address format for set-associative cache is as follows:



10. (a) Evaluate the following arithmetic expression using 0, l, 2, 3 address instruction:

$$X = (A+B)/(C*D).$$

(b) Why do we require memory hierarchy? Show the memory hierarchy diagram indicating speed and cost.

(c) Distinguish between SRAM and DRAM. 8 + (2 + 2) + 3

Answer

(a) To evaluate the statement $X = (A + B) / (C * D)$ in zero, one, two and three address machines, we assume the following assumptions:

LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD, DIV and MULT are used for the arithmetic operations addition, division and multiplication respectively. Assume that the respective operands are in memory addresses *A*, *B*, *C* and *D* and the result must be stored in the memory at address *X*.

For zero-address machine:

The assembly-language program using zero-address instructions is written next. In the comment field, the symbol TOS is used, which means the top of stack.

```
PUSH A      ; TOS ← A
PUSH B      ; TOS ← B
ADD         ; TOS ← (A + B)
PUSH C      ; TOS ← C
PUSH D      ; TOS ← D
MULT        ; TOS ← (C * D)
DIV         ; TOS ← (A + B) / (C * D)
POP X       ; X ← TOS
```

For one-address machine:

The assembly-language program using one-address instructions is written below.

```
LOAD C      ; AC ← M[C]
MULT D      ; AC ← AC * M[D]
STORE T     ; T ← AC
LOAD A      ; AC ← M[A]
ADD B       ; AC ← AC + M[B]
DIV T       ; AC ← AC/M[T]
STORE X     ; X ← AC
```

For two-address machine:

The assembly-language program using two-address instructions is written below.

```
LOAD R1, A  ; R1 ← M[A]
ADD R1, B   ; R1 ← R1 + M[B]
LOAD R2, C  ; R2 ← M[C]
MULT R2, D  ; R2 ← R2 * M[D]
DIV R1, R2  ; R1 ← R1 / R2
STORE X, R1 ; X ← R1
```

For three-address machine:

The assembly-language program using three-address instructions is written below.

```
ADD R1, A, B        ; R1 ← M[A] + M[B]
MULT R2, C, D       ; R2 ← M[C] * M[D]
DIV X, R1, R2       ; X ← R1/R2
```

(b) Ideally, we would like to have the memory which would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three requirements simultaneously. If we increase the speed and capacity, then cost will increase. We can achieve these goals at optimum level by using several types of memories, which collectively give a memory hierarchy.

A memory hierarchy system is broadly divided into following four groups, shown in Fig. 8.

- Secondary (auxiliary) memory
- Main (primary) memory
- Cache memory
- Internal memory

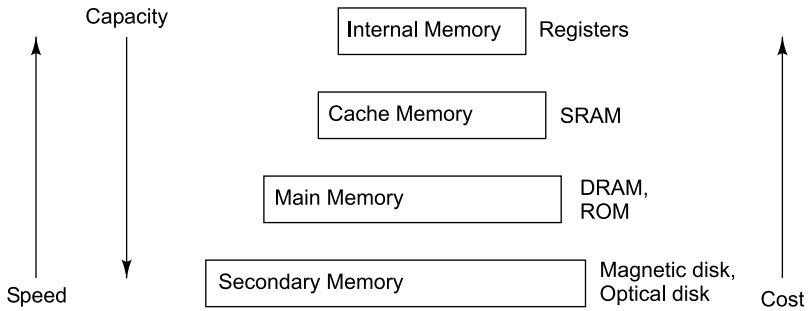


Figure 8 *Memory hierarchy*

(c) (i) The SRAM has lower access time, which means it is faster compared to the DRAM.

(ii) The SRAM requires constant power supply, which means this type of memory consumes more power; whereas, the DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.

(iii) Due to the relatively small internal circuitry in the one-bit memory cell of DRAMs, the large storage capacity in a single DRAM memory chip is available compared to the same physical size SRAM memory chip. In other words, DRAM has high packaging density compared to the SRAM.

(iv) SRAM is costlier than DRAM.

11. Write short notes on any *three of* the following: 3×5

(a) Bus organization using Tri-state Buffer

(b) Cache replacement policies

(c) Restoring Division method

(d) Comparison between RISC and CISC

(e) Instruction format.

Answer

(a) A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The gate is controlled by one separate control input C. If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low the gate does not produce any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown in Fig. 9.

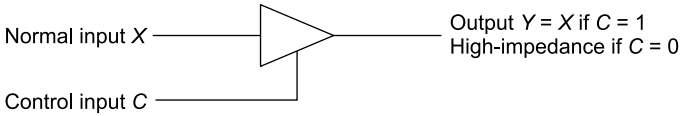


Figure 9 *Symbol of a tri-state buffer gate*

The common bus is used to transfer a register's content to other register or memory at a single time. A common bus system with tri-state buffers is described in Fig. 10, where one line of the common bus is shown.

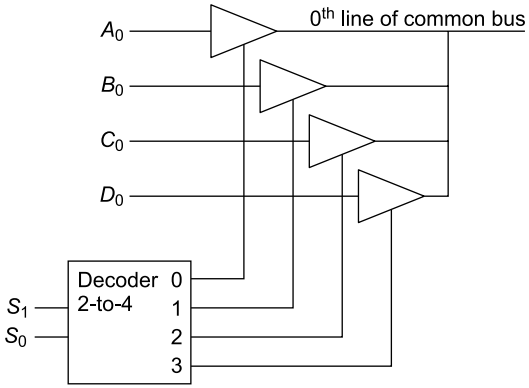


Figure 10 *A single line of a bus system with tri-state buffers*

Assume that there are four registers A, B, C and D. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

(b) In case a miss occurs in cache memory, then a new data from main memory needs to be placed over old data in the selected location of cache memory. In case of direct mapping cache, we have no choice and thus no replacement algorithm is required. The new data has to be stored only in a specified cache location as per the mapping rule for the direct mapping cache. For associative mapping and set-associative mapping, we need a replacement algorithm since we have multiple choices for locations. Some most used replacement algorithms are given below.

First-In First-Out (FIFO) Algorithm: This algorithm chooses the word that has been in the cache for a long time. In other words, the word which entered the cache first, gets pushed out first.

Least Recently Used (LRU) Algorithm: This algorithm chooses the item for replacement that has been used by the CPU minimum number of times in the recent past.

(c) The restoring division method uses three n-bit registers A, M, Q for dividing two n-bit numbers. The register M is used to hold the divisor. Initially, A contains 0 and Q holds the n-bit dividend. In each iteration, the contents of register-pair AQ are shifted to the left first. The content of M is then subtracted from A. If the result of subtraction is positive, a 1 is placed into the vacant position created in lsb position of Q by the left shift operation; otherwise a 0 is put into this position and before beginning the next iteration, restore the content of A by adding the current content of A register with M. For this step, the algorithm is referred to as a restoring division algorithm. When, the algorithm terminates, the A register contains the remainder result and the Q register contains the quotient result.
The restoring division algorithm to divide two n-bit numbers is described using the flowchart shown in Fig. 11.

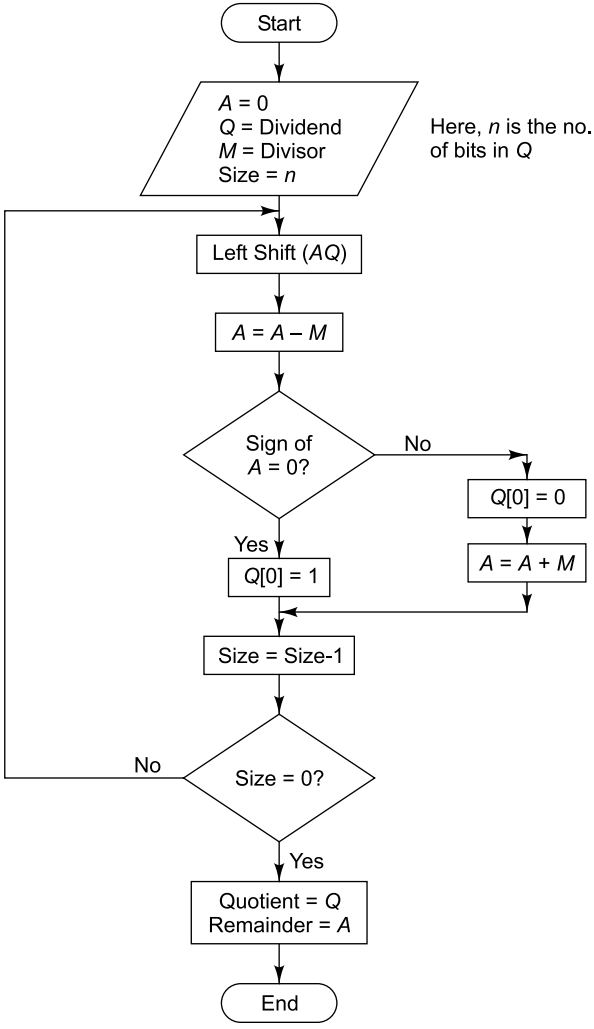


Figure 11 *Restoring division algorithm*

(d)

CISC	*RISC*
1. A large number of instruction types used—typically from 100 to 250 instructions.	1. Relatively few number of instruction types—typically less than100 instructions.
2. A large number of addressing modes used—typically from 5 to 15 different modes.	2. Relatively few addressing modes—typically less than or equal to 5.
3. Variable-length instruction formats.	3. Fixed-length, easily decoded instruction formats.
4. Small number of general-purpose registers (GPRs)—typically 8–24 GPRs.	4. Large number of general-purpose registers (GPRs)—typically 32–192 GPRs.
5. Clock per instruction (CPI) lies between 2 and 15.	5. Clock per instruction (CPI) lies between 1 and 2.
6. Mostly microprogrammed control units.	6. Mostly hardwired control units.
7. Most instructions manipulate operands in memory.	7. All operations are executed within registers of the CPU.

(e) In broad sense, the superiority of a computer is decided on the basis of its instruction set. Since the total number of instructions and their powerfulness has contributed to the efficiency of the computer, these two factors are given highest priority. An efficient program is the one which is short, hence fast execution and occupies less memory space. The size of a program depends largely on the formats of instructions used.

A computer usually has a variety of instruction formats. It is the task of the control unit within CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction. The most common format followed by instructions is depicted in Fig. 12.

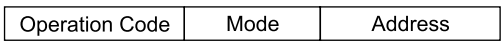


Figure 12 *Different fields of instructions*

The bits of the instruction are divided into groups called fields. The commonly used fields found in instruction formats are given below.

1. *Operation Code* (or, simply *Op-code*): This field states the operation to be performed. This field defines various processor operations, such as add, subtract, complement, etc.
2. *Address:* An address field designates a memory address or a processor register or an operand value.
3. *Mode*: This field specifies the method to get the operand or effective address of operand. In instruction set of some computers, the op-code itself explicitly specifies the addressing mode used in the instruction. A computer has various addressing modes.

For example, in the instruction ADD R1, R0; ADD is the op-code to indicate the addition operation and R1, R0 are the address fields for operands.

In certain situations, other special fields are sometimes used. For example, a field that gives the number of shifts in a shift-type micro-operation or, a label field is used to process unconditional branch instruction.

The memory or processor registers store the operand values on which operation codes specified by computer instructions are executed. Memory addresses are used to specify operands stored in

memory. A register address specifies an operand stored in processor register. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 32 processor registers R0 to R31 has a register address field of 5 bits. For example, processor register R7 is specified by the binary number 00111. The internal organization of processor registers determines the number of address fields in the instruction.

2010
Computer Organization
(CS-303)

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) From a source code, a compiler can detect
(a) run-time error (b) logical errors (c) syntax error (d) none of these

Answer
(c) Syntax error

(ii) The purpose of ROM in a computer system is
(a) to store constant data required for computers own use
(b) to help reading from memory
(c) to store application program
(d) to store 0s in memory.

Answer
(a) to store constant data required for computers own use

(iii) Which one does not possess the characteristics of a memory element?
(a) A toggle switch (b) A lamp (c) An AND gate (d) None of these

Answer
(c) An AND gate

(iv) Data from memory location after fetching is deposited by memory in
(a) MAR (b) MBR (c) IR (d) Status Register.

Answer
(b) MBR

(v) How many minimum NAND gates are required to make a flip flop?
(a) 4 (b) 3 (c) 2 (d) 5.

Answer

(a) 4

(vi) Virtual memory system allows the employment of
(a) more than address space (b) the full address space
(c) more than hard disk capacity (d) none of these.

Answer

(a) more than address space

(vii) A system has 48-bit virtual address, 36-bit physical address and 128 MB main memory. How many virtual and physical pages can the address space support?
(a) 2^{36}, 2^{24} (b) 2^{12}, 2^{36} (c) 2^{24}, 2^{34} (d) 2^{34}, 2^{36}

Answer

(a) 2^{36}, 2^{24}

(viii) A UART is an example of
(a) serial asynchronous data transmission ship (b) PIO
(c) DMA controller (d) none of these

Answer

(a) serial asynchronous data transmission ship

(ix) A priority interrupt may be accomplished by
(a) polling (b) daisy chain
(c) parallel method of priority interrupt (d) all of these.

Answer

(d) all of these

(x) Control program memory can be reduced by
(a) horizontal format (b) vertical format micro-program
(c) hardwired control unit (d) none of these.

Answer

(b) vertical format micro-program

GROUP-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. (a) A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.
(i) How many selection inputs are there in each multiplexer?
(ii) What size of multiplexers is needed?
(iii) How many multiplexers are there in the bus?
(b) Why do most computers have a common bus system? 3 + 2

Answer

(a) (i) and (ii) The size of each multiplexer must be 16×1, because it multiplexes 16 data lines. Therefore, the number of selection inputs in each multiplexer is 4 (because $16 = 2^4$)

(iii) The number of multiplexers needed to construct the bus is equal to 32, the number of bits in each register.

(b) Many registers are provided in the CPU of a computer for fast execution. Therefore, several paths must be provided to transfer information from one register to another. If a separate communication line is used between each register pair in the system, the number of lines will be excessive and thus cost of communication will be huge. Thus it is economical to have a common bus system for transferring information between registers in a multiple-register configuration.

3. Draw the logic diagram of a common bus which connects 4 registers of 4-bit each using tri-state buffers. 5

Answer

A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that no output is produced though there is an input signal and does not have logic significance. The gate is controlled by one separate control input *C*. If *C* is high the gate behaves like a normal logic gate having output 1 or 0. When *C* is low the gate does not produce any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown in Fig. 1.

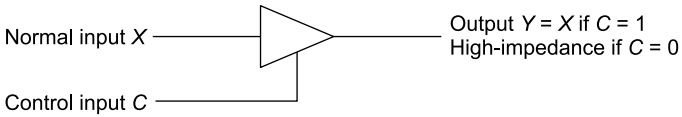


Figure 1 *Graphic symbol for a tri-state buffer gate*

A common bus system with tri-state buffers is described in Fig. 2. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the

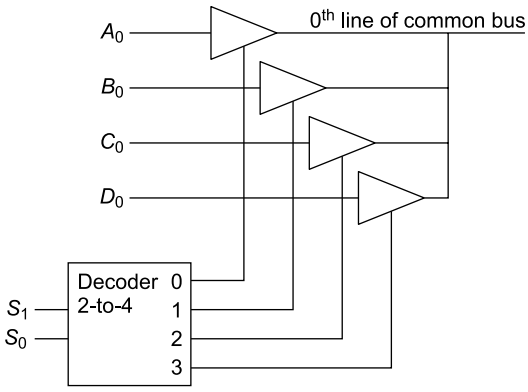


Figure 2 *A single line of a bus system with tri-state buffers*

decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

4. What is virtual memory? Why is it called virtual? Write the advantages of virtual memory. 2 + 1 + 2

Answer

Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, although which may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual(logical) address is mapped to a physical address in main memory. This mapping is done during run-time and is performed by a hardware device called *memory-management unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

Virtual memory is not a physical memory, is actually a technique. That is why it is called virtual memory.

The advantage of virtual memory is efficient utilization of main memory, because the larger size program is divided into blocks and partially each block is loaded in the main memory whenever it is required. Thus multiple programs can be executed simultaneously. The technique of virtual memory has other advantages of efficient CPU utilization and improved throughput.

5. What is programmed I/O technique? Why is it not very useful? 3 + 2

Answer

Program I/O technique is the software method where CPU is needed all the times during data transfer between any two devices. Programmed I/O operations are the result of I/O instructions written in the computer program or I/O routine. Each data item transfer is initiated by an instruction in the program or I/O routine. Generally, the transfer is to and from a CPU register and peripheral. Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. In other words, the CPU polls the devices for next data transfer. This is why the programmed I/O is sometimes called *polled I/O*.

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. Generally the CPU is 5-7 times faster than an I/O device. Thus, the difference in data transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

6. Draw the block diagram and explain the functionality of microprogrammed control unit.

Answer

In the microprogrammed approach, all control functions that can be simultaneously activated are grouped to form control words stored in a separate ROM memory called the control memory. From the control memory, the control words are fetched one at a time and the individual control fields are routed to various functional units to activate their appropriate circuits. The desired task is performed by activating these circuits sequentially.

Figure 3 depicts the general structure of a microprogrammed control unit.

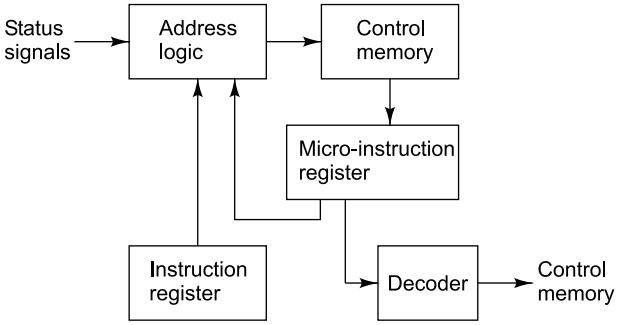


Figure 3 *Microprogrammed control unit*

Microprogramming is a modern concept used for designing a control unit. It can be used for designing control logic for any digital system. As stated earlier, a microprogrammed control unit's control words are held in a separate ROM memory called the *control memory* (CM). Each control word contains signals to activate one or more micro-operations. When these words are retrieved in a sequence, a set of micro-operations are activated that will complete the desired task.

GROUP-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. (a) Explain the difference between associative and set-associative cache mapping technique.
   (b) With the help of following information, determine the size of sub-fields (in bits) in the address for direct mapping, associative mapping and Set-associative mapping:
   - 512 MB main memory and 2 MB cache memory
   - The address space of this processor is 256 MB
   - The block size is 256 bytes
   - There are 16 blocks in a cache set.

   (c) Briefly explain the two write policies, write through and write back for cache design. What are we getting the advantages and disadvantages of both the methods? 4 + 6 + 5

Answer

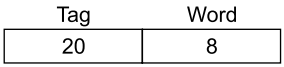
(a) (i) The associative cache memory uses the fastest and most flexible mapping method, in which both address and data of the memory word are stored; whereas, in set-associative cache, two or more words can be stored under the same index address, which is not stored in the memory. Each data word is stored together with its tag. The number of tag-data words under an index is said to form a *set.*

(ii) The set-associative cache has higher hit ratio compared to associative cache.

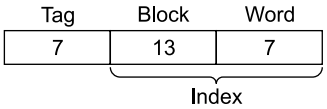
(iii) The set-associative cache is the most expensive memory. The cost increases as set size increases.

(b) Given,
The capacity of main memory = 512 MB
The capacity of cache memory = 2 MB

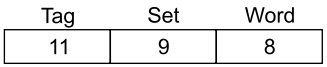
Block size = 256 bytes.
A set contains 16 blocks.
Since the address space of the processor is 256 MB.
The processor generates address of 28-bit to access a byte (word).
The number of blocks main memory contains = 256 MB/256 bytes = 2^{20}.
Therefore, no. of bits required to specify one block in main memory = 20.
Since the block size is 256 bytes.
The no. of bits required to access each word (byte) = 8.
For associative cache, the address format is



The number of blocks cache memory contains = 2 MB / 256 bytes = 2^{13}.
Therefore, no. of bits required to specify one block in cache memory = 13.
The tag field of address = 28 – (13 + 8) = 7-bit.
For direct cache, the address format is



In case of set-associative cache,
A set contains 16 blocks.
Therefore, the number of sets in cache = 2^{13} / 16 = 2^9.
Thus, the number of bits required to specify each set = 9.
The tag field of address = 28 – (9 + 8) = 11-bit.
For set-associative cache, the address format is



(c) *Write-Through Policy*: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated.
Advantage: The main memory always contains the same data as the cache.
Disadvantage: It is a slow process, since each time main memory needs to be accessed.
Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified or dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set. The philosophy of this method is based on the fact that during a write operation, the word residing in cache may be accessed several times (temporal locality of reference).
Advantage: The method is faster than write-through, because this method reduces the number of references to main memory.
Disadvantage: This method may encounter the problem of inconsistency due to two different copies of the same data, one in cache and other in main memory.

8. (a) With the help of suitable diagram, explain the advantage of carry look ahead adder over conventional parallel adder.
   (b) If a CPU has 8-bit data bus and 16-bit address bus, draw the connection diagram for this CPU with four 256 × 8 RAM and one 512 × 8 ROM.
   (c) Show the bus connection with a CPU to connect four RAM chips of size 256 × 8-bit each and a ROM chip of 512 × 8-bit size. Assume the CPU has 8-bit data bus and 16-bit address bus. Clearly specify generation of chip select signals.
   (d) What is dirty bit? 5 + 4 + 4 + 9

Answer

(a) A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1} = A_iB_i + (A_i+B_i)C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in Fig. 4. The sum bit S_i is = $A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

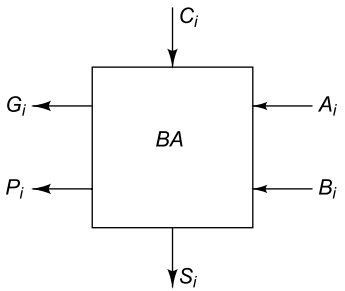


Figure 4 *Basic Adder*

For example, we want to design a 4-bit CLA, for which four carries C_1, C_2, C_3 and C_4 are to be generated. Using equation number (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2$$
$$C_4 = G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$
$$= G_1 + P_1(G_0 + P_0C_0)$$

$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$

$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$

$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \quad (5)$$

The equations (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in Fig. 5.

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

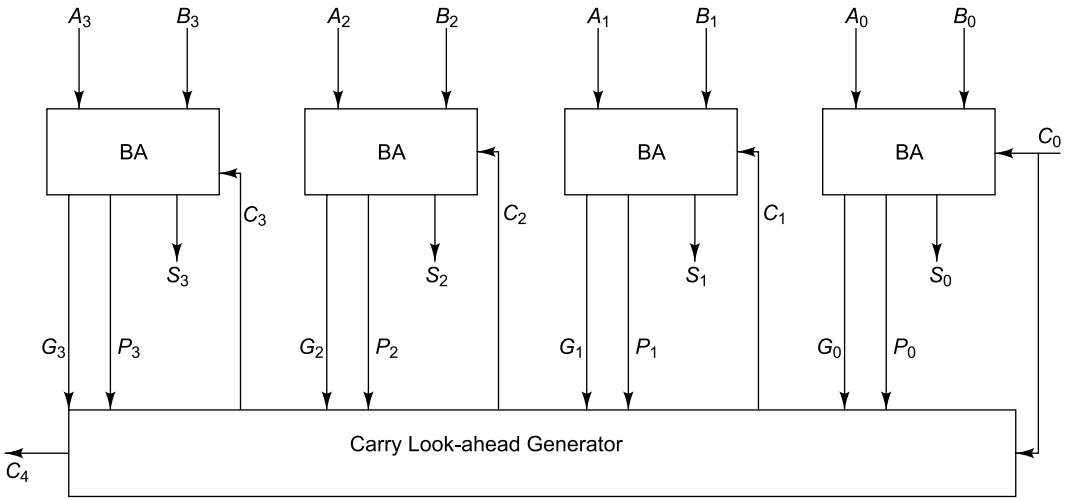


Figure 5 *4-bit Carry Look-ahead Adder (CLA)*

(b) Figure 6 shows the interconnection diagram of these memories with the CPU having 16 address lines.

The address lines 1 to 8 are connected to each memory and address line 9 is used in dual purposes. In case of a RAM selection out of four RAMs, the line no. 9 and line no. 10 are used through a 2-to-4 decoder. The line no. 9 is also connected to the ROM as address line along with lines 1 to 8 giving a total of 9 address lines in the ROM, since the ROM has 512 locations. The CPU address line number 11 is used for separation between RAM and ROM. The other 12 to 16 lines of CPU are unused and for simplicity we assume that they carry 0s as address signals. For ROM, 10^{th} line is unused and thus it can be assumed that this line carries signal 0.

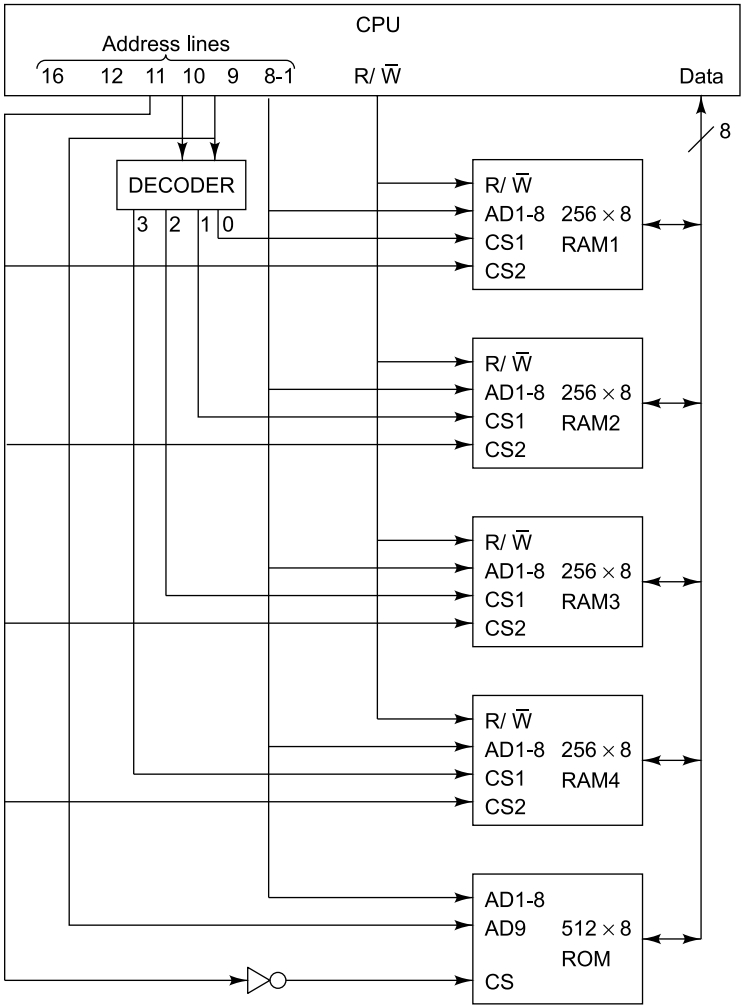


Figure 6 *Interconnection diagram*

The address map table for the memory connection to the CPU shown in Fig. 6 is constructed in the following table.

Chip selected	*Address space (in HEX)*	*Address bus*									
		11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
RAM1	0400 – 04FF	1	0	0	x	x	x	x	x	x	x	x
RAM2	0500 – 05FF	1	0	1	x	x	x	x	x	x	x	x
RAM3	0600 – 06FF	1	1	0	x	x	x	x	x	x	x	x
RAM4	0700 – 07FF	1	1	1	x	x	x	x	x	x	x	x
ROM	0000 – 01FF	0	0	x	x	x	x	x	x	x	x	x

(c) Same as question no. (b) above.

(d) In write-back cache update policy, only cache location is updated and it is marked as updated with an associated tag bit, known as *dirty bit.* The main memory location of the word is updated later, when the block containing this marked word is to be removed from the cache to make room for the new block.

9. (a) Explain Booth's Multiplication Algorithm for signed 2's complement numbers with proper flow-chart. Illustrate this with an example by multiplying $(-9) \times (-13)$.

(b) Explain destructive read out and non-destructive read out of memory system. (5 + 5) + 5

Answer

(a) The algorithm inspects two lower-order multiplier bits at time to take the next step of action. The algorithm is described by the flowchart shown in Fig. 7. A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contain the product. Ignore the right end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier.

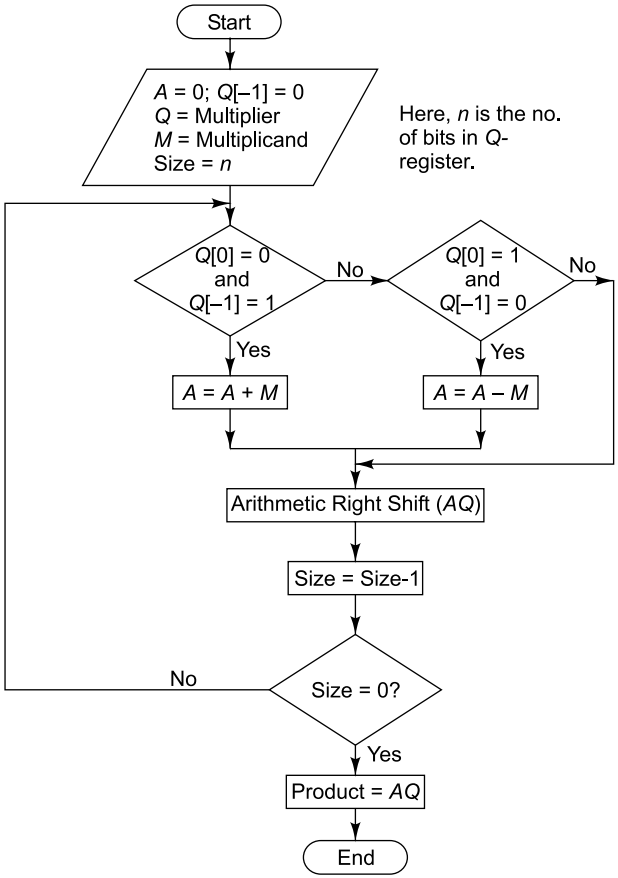


Figure 7 *Booth's multiplication algorithm*

Multiplication of numbers $(-9)_{10}$ and $(-13)_{10}$:
Multiplicand, $M = -9 = 10111$ and multiplier, $Q = -13 = 10011$.

	M	A	Q	Size
Initial Configuration	10111	00000	10011 0	5
Step-1				
As $Q[0] = 1$ and $Q[-1] = 0$				
$A = A - M$	10111	01001	10011 0	–
ARS(AQ)	10111	00100	11001 1	4
Step-2				
As $Q[0] = 1$ and $Q[-1] = 1$				
ARS(AQ)	10111	00010	01100 1	3
Step-3				
As $Q[0] = 0$ and $Q[-1] = 1$				
$A = A + M$	10111	11001	01100 1	–
ARS(AQ)	10111	11100	10110 0	2
Step-4				
As $Q[0] = 0$ and $Q[-1] = 0$				
ARS(AQ)	10111	11110	01011 0	1
Step-5				
As $Q[0] = 1$ and $Q[-1] = 0$				
$A = A - M$	10111	00111	01011 0	–
ARS (AQ)	10111	00011	10101 1	0

Since the Size register becomes 0, the algorithm is terminated and the product is = AQ = 00011 10101, which shows that the product is a positive number and the result is + 117 in decimal.

(b) In some memories, the reading the memory word destroys the stored word, this fact is known as *destructive readout* and memory is known as *destructive readout memory*. In these memories, each read operation must be followed by a write operation that restores the memory's original state. Example includes dynamic RAM.
In some memories, the reading the memory word does not destroy the stored word, this fact is known as *non-destructive readout* and memory is known as *non-destructive readout memory*. Examples include static RAM and magnetic memory.

10. (a) Explain non-restoring division algorithm and explain the hardware diagram. Perform the Restoring division operation with 19 divided by 8.

(b) What is Belady anomaly for page replacement technique? Explain with example.

(3 + 2 + 5) + 5

Answer

(a) In the restoring method, some extra additions are required to restore the number, when A is negative. Proper restructuring of the restoring division algorithm can eliminate that restoration step. This is known as the non-restoring division algorithm.

The three main steps in restoring division method were:

1. Shift AQ register pair to the left one position.
2. $A = A - M$.
3. If the sign of A is positive after the step 2, set $Q[0] = 1$; otherwise, set $Q[0] = 0$ and restore A.

Now, assume that the step 3 is performed first and then step 1 followed by step 2. Under this condition, the following two cases may arise.

Case 1: When A is positive

Note that shifting A register to the left one position is equivalent to the computation of $2A$ and then subtraction. This gives the net effect on A as $2A - M$.

Case 2: When A is negative

First restore A by adding the content of M register and then shift A to the left one position. After that A will be subtracted from M register. So, all together they give rise the value of A as $2(A + M) - M = 2A + M$.

Basis on these two observations, the non-restoring division method can be designed.

This algorithm removes the restoration step, though it may require a restoration step at the end of algorithm for remainder A, if A is negative.

The hardware block diagram of non-restoring division algorithm is shown in Fig. 8.

Restoring division operation with 19 divided by 8:

Dividend $Q = 19 = 010011$ and divisor $M = 8 = 001000$.

	M	A	Q	Size
Initial Configuration	001000	000000	010011	6
Step-1				
LS(AQ)	001000	000000	10011–	–
$A = A - M$	001000	111000	10011–	–
As Sign of A = –ve				
Set $Q[0] = 0$ and Restore A	001000	000000	100110	5

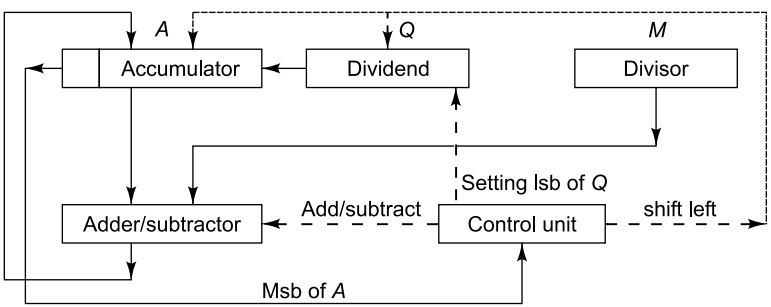


Figure 8 *Block diagram of non-restoring division algorithm*

Step-2				
LS(AQ)	001000	000001	00110 –	–
$A = A - M$	001000	111001	00110 –	–
As Sign of A = –ve				
Set Q[0] = 0				
Restore A	001000	000001	001100	4
Step-3				
LS(AQ)	001000	000010	01100 –	–
$A = A - M$	001000	111010	01100 –	–
As Sign of A = –ve				
Set Q[0] = 0				
Restore A	001000	000010	011000	3
Step-4				
LS(AQ)	001000	000100	11000 –	–
$A = A - M$	001000	111100	11000 –	–
As Sign of A = –ve				
Set Q[0] = 0				
Restore A	001000	000100	110000	2
Step-5				
LS(AQ)	001000	001001	10000 –	–
$A = A - M$	001000	000001	10000 –	–
As Sign of A = +ve				
Set Q[0] = 1	001000	000001	100001	1
Step-6				
LS(AQ)	001000	000011	00001 –	–
$A = A - M$	001000	111011	00001 –	–
As Sign of A = –ve				
Set Q[0] = 0				
Restore A	001000	000011	000010	0

From the above result, we see that the quotient = Q = 000010 = 2 and remainder = A = 000011 = 3.

(b) The general fact for any page replacement technique is that as the number of frames available increases, the number of page faults will decrease.

However, for some page replacement algorithms, the page fault rate may *increase* as the number of allocated frames increases. This fact is most unexpected and is known as *Belady's anomaly.*

To illustrate the problem that is possible with a FIFO page replacement algorithm, we consider the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Then it can be noticed that the number of faults for four frames (10) is *greater* than the number of faults for three frames (9). This is very much unexpected result and is known as *Belady's anomaly.*

11. (a) What are the different types of DMA controllers and how do they differ in their functioning?
    (b) Explain the basic DMA operations for transfer of data between memory and peripherals.
    (c) Differentiate between memory mapped I/O and I/O mapped I/O. 5 + 5 + 5

Answer

(a) DMA controllers are of two types:
- Independent DMA controller
- DMA controller having multiple DMA-channels

Independent DMA controller:
For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in Fig. 9 for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.

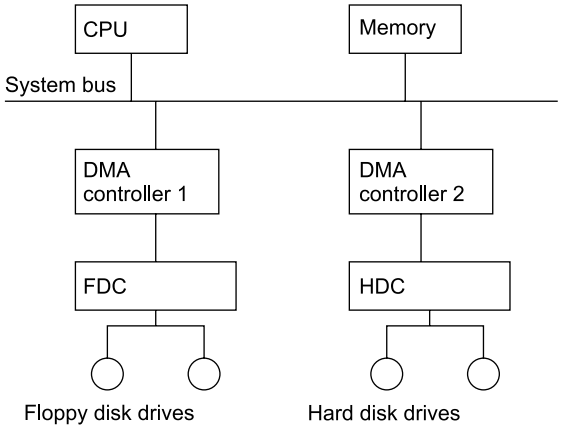


Figure 9 *Independent DMA controller*

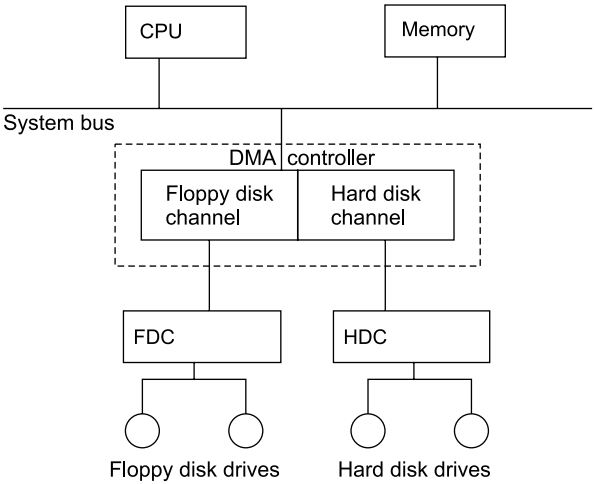


Figure 10 *DMA controller having multiple DMA channels*

DMA controller having multiple DMA-channels:
In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in Fig. 10 for floppy disk controller (FDC) and hard disk controller (HDC).

(b) In DMA transfer, I/O devices can directly access the main memory without intervention by the processor. Figure 11 shows a typical DMA system. The sequences of events involved in a DMA transfer between an I/O device and the main memory are discussed next.

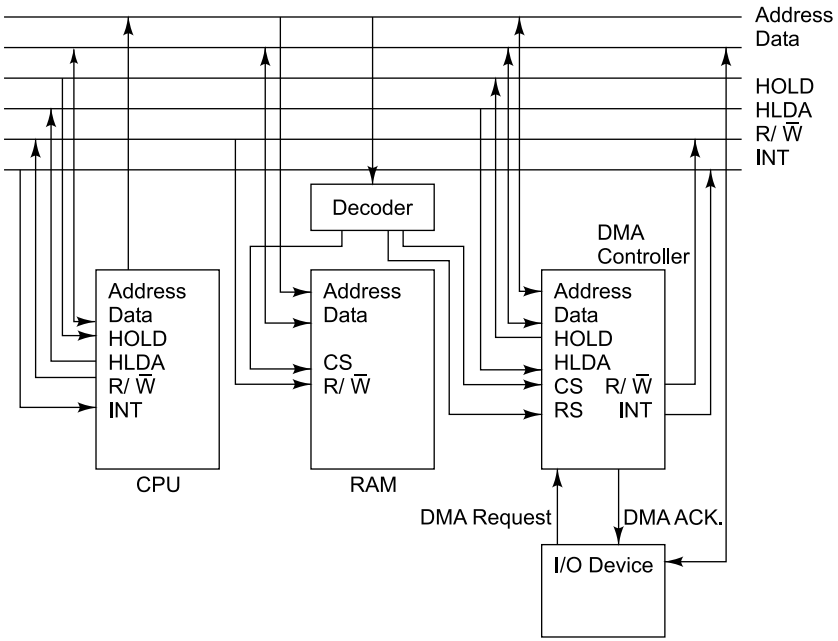


Figure 11 *Typical DMA System*

A DMA request signal from an I/O device starts the DMA sequence. DMA controller activates the HOLD line. It then waits for the HLDA signal from the CPU. On receipt of HLDA, the controller sends a DMA ACK (acknowledgement) signal to the I/O device. The DMA controller takes the control of the memory buses from the CPU. Before releasing the control of the buses to the controller, the CPU initializes the address register for starting memory address of the block of data, word-count register for number of words to be transferred and the operation type (read or write). The I/O device can then communicate with memory through the data bus for direct data transfer. For each word transferred, the DMA controller increments its address-register and decrements its word count register. After each word transfer, the controller checks the DMA request line. If this line is high, next word of the block transfer is initiated and the process continues until word count register reaches zero

(i.e., the entire block is transferred). If the word count register reaches zero, the DMA controller stops any further transfer and removes its HOLD signal. It also informs the CPU of the termination by means of an interrupt through INT line. The CPU then gains the control of the memory buses and resumes the operations on the program which initiated the I/O operations.

(c) (i) In the isolated (I/O mapped) I/O, computers use one common address bus and data bus to transfer information between memory or I/O and the CPU; but use separate read-write control lines one for memory and another for I/O. Whereas, in memory mapped I/O, computers use only one set of read and write lines along with same set of address and data buses for both memory and I/O devices.

(ii) The isolated I/O technique isolates all I/O interface addresses from the addresses assigned to memory. Whereas, the memory mapped I/O does not distinguish between memory and I/O addresses.

(iii) Processors use different instructions for accessing memory and I/O devices in isolated I/O. In memory mapped I/O, processors use same set of instructions for accessing memory and I/O.

(iv) Thus, the hardware cost is more in isolated I/O relative to the memory mapped I/O, because two separate read-write lines are required in first technique.

2010
Computer Organization and Architecture (CS-404)

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) Periodic refreshing is needed in
(a) ROM (b) EPROM (c) SRAM (d) DRAM

Answer
(d) DRAM

(ii) The 2's complement representation of (–24) in a 16-bit micro-computer is
(a) 0000 0000 0001 1000 (b) 1111 1111 1110 0111
(c) 1111 1111 1110 1000 (d) 0001 0001 1111 0011

Answer
(c) 1111 1111 1110 1000

(iii) Which of the following addressing modes is used in 'Push B'?
(a) Immediate (b) Register (c) Direct (d) Register Direct

Answer
(a) Immediate

(iv) Which of the following I/O mechanisms requires the least hardware support?
(a) Polled (b) Interrupt driven (c) DMA (d) Memory-mapped

Answer
(a) Polled

(v) The basic principle of a Von Neumann computer is
(a) storing program and data in separate memory
(b) using pipeline concept

(c) storing program and data in the same memory
(d) using a large number of register.

Answer

(c) storing program and data in the same memory

(vi) The performance of a pipelined processor suffers if
(a) the pipeline stages have different delays
(b) consecutive instructions are depends on each other
(c) the pipeline stages share H/W resource
(d) all of these.

Answer

(d) all of these

(vii) Associative memory is a
(a) very cheap memory (b) pointer addressable memory
(c) content addressable memory (d) slow memory

Answer

(c) content addressable memory

(viii) How many RAM chips of size (256 K × 1 bit) are required to build 1 M byte memory?
(a) 24 (b) 10 (c) 32 (d) 8

Answer

(c) 32

(ix) A ripple carry adder requires time in the order of
(a) linear time (O(N)) (b) constant (c) (O(log (N))) (d) (O (N log (N)))

Answer

(a) linear time (O(N))

(x) How many address bits are required for a 1024 × 8 memory?
(a) 5 (b) 10 (c) 1024 (d) None of these

Answer

(b) 10

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. 3 × 5 = 15

2. Using 8-bit 2's complement integers, perform the following computations:
(i) 26 – (–4) (ii) 1 – 7

Answer

(i) Binary representation of 26 = 0001 1010
Binary representation of 4 = 0000 0100
Binary representation of (–4) = 1111 1100
2's complement of (–4) = 0000 0100
Therefore, 26 – (–4) = 0001 1010 + 0000 0100 = 0001 1110, which is the answer.

(ii) Binary representation of 1 = 0000 0001
Binary representation of 7 = 0000 0111
2's complement of (7) = 1111 1001

Therefore, 1 – 7 = 0000 0001 + 1111 1001 = 1111 1010, which shows that the result is negative (because of bit '1' in leftmost position). To get the result in familiar form, take 2's complement of the 7-bit magnitude of this result and this becomes 000 0110. This is binary equivalent of +6. Therefore, the result is –6.

3. Explain the following with respect to pipelined architecture:
Speed-up, throughput, efficiency.

Answer

Speed-up: It is defined as

$$S_k = \frac{\text{Time to execute } n \text{ tasks in } k\text{-stage non-pipeline processor}}{\text{Time to execute } n \text{ tasks in } k\text{-stage pipeline processor}}$$

$$= \frac{n.k.\tau}{\tau[k+(n-1)]} \quad \text{where, } \tau = \text{clock period of the pipeline processor.}$$

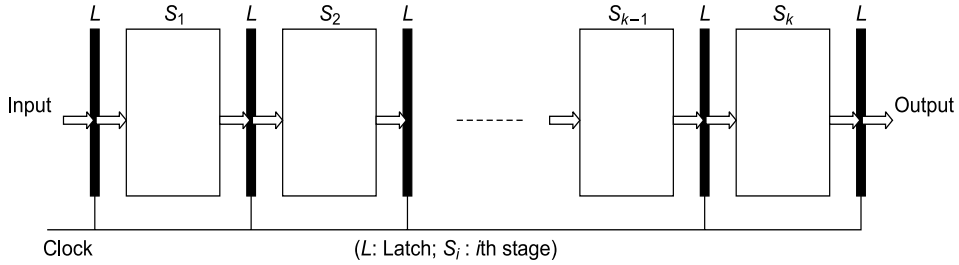


Figure 1 *Concept of pipelining*

Time to execute n tasks in k-stage pipeline processor is $\tau[k + (n – 1)]$ units, where k clock periods (cycles) are needed to complete the execution of the first task and remaining $(n – 1)$ tasks require $(n–1)$ cycles. Time to execute n tasks in k-stage non-pipeline processor is $n.k.\tau$, where each task requires k cycles because no new task can enter the pipeline until the previous task finishes.

It can be noted that the maximum speed-up is k, for $n >> k$. But this maximum speed-up is never fully achievable because of data dependencies between instructions, interrupts, program branches, etc.

Efficiency: To define it, we need to define another term 'time-space span'. It is the product (area) of a time interval and a stage space in the space-time diagram. A given time-space span can be in either a busy state or an idle state, but not both.

The *efficiency* of a linear pipeline is measured by the percentage of busy time-space spans over the total time-space span, which equals the sum of all busy and idle time-space spans. Let n, k, τ be the number of tasks (instructions), the number of stages and the clock period of a linear pipeline, respectively. Then the efficiency is defined by

$$\eta = \frac{n.k.\tau}{k.[k.\tau + (n-1).\tau]} = \frac{n}{k+(n-1)}$$

Note that $\eta \to 1$ (i.e., 100%) as $n \to \infty$. This means that the larger the number of tasks flowing through the pipeline, the better is its efficiency. For the same reason as speed-up, this ideal efficiency is achievable.

Throughput: The number of tasks that can be completed by a pipeline per unit time is called its throughput. Mathematically, it is defined as

$$\omega = \frac{n}{k.\tau + (n-1).\tau} = \frac{\eta}{\tau}$$

Note that in ideal case, $\omega = 1/\tau = f$, frequency, when $\eta = 1$. This means that the maximum throughput of a linear pipeline is equal to its frequency, which corresponds to one output result per clock period.

4. Explain the working (with a suitable example) of a carry look-ahead adder.

Answer

A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1} = A_iB_i + (A_i + B_i)C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in Fig. 2. The sum bit S_i is = $A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

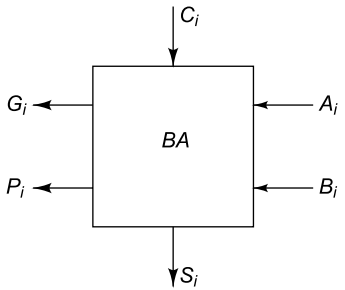


Figure 2 *Basic adder*

For example, we want to design a 4-bit CLA, for which four carries C_1, C_2, C_3 and C_4 are to be generated. Using equation number (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2$$
$$C_4 = G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$

$$= G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$
$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$
$$C_4 = G_3 + P_3C_3$$
$$= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \quad (5)$$

The equations (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in Fig. 3.

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

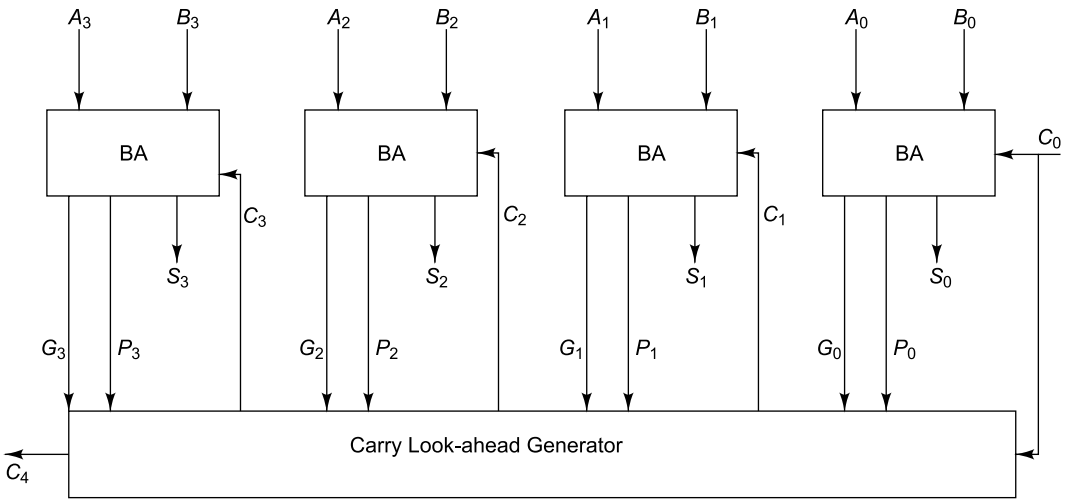


Figure 3 *4-bit Carry Look-ahead Adder (CLA)*

5. Explain how a RAM of capacity $2k$ bytes can be mapped into address space $(1000)_H$ to $(17FF)_H$ of CPU having a 16-bit address lines. Show how the address lines are decoded to generate the chip select condition for the RAM.

Answer

Since the capacity of RAM memory is $2k$ bytes, the memory uses 11 (2 KB = 2^{11}) address lines, say namely A_{10}–A_0, to select one word. Thus, memory's internal address decoder uses 11 lines A_{10}–A_0 to select one word.

To select this memory module, remaining 5 (i.e., 16–11) address lines A_{15}–A_{11} are used. Thus, an external decoding scheme is employed on these higher-order five address bits of processor's address.

The address space of the memory is 1000_H and $17FF_H$.
Therefore, the starting address $(1000)_H$ in memory is as

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Based on the higher-order five bits (00010), external decoding scheme performs a logical AND operation on address values: $\overline{A_{15}}$, $\overline{A_{14}}$, $\overline{A_{13}}$, A_{12} and $\overline{A_{11}}$. The output of AND gate acts as chip select (CS) line. The address decoding scheme is shown in Fig. 4.

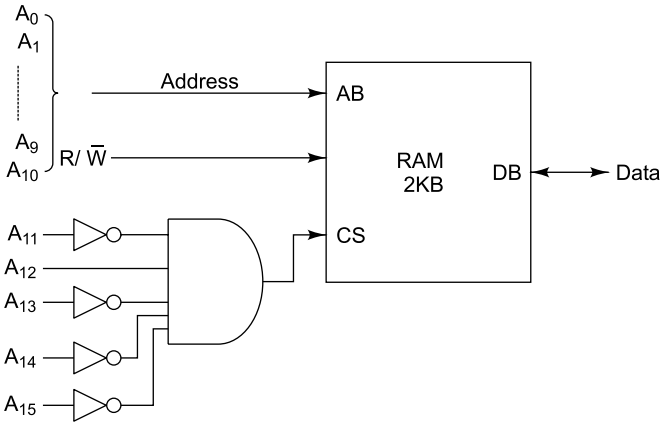


Figure 4 *Address decoding scheme*

6. What is cache memory? What are the different mechanisms of writing into it? Briefly describe.

Answer

Cache memory is a special high-speed main memory, sometimes used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. Generally, the CPU is faster than main memory, thus resulting that processing speed is limited mainly by the speed of main memory. So, a technique used to compensate the speed mismatch between CPU and main memory is to use an extremely fast, small cache between CPU and main memory, whose access time is close to CPU cycle time. The cache is used for storing portions of programs currently being executed in the CPU and temporary data frequently needed in the present computations. Thus, the cache memory acts as buffer between the CPU and main memory. By making programs and data available at a rapid rate, it is possible to increase the performance of computer.

There are two methods in writing into cache memory:

Write-Through Policy: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus, the main memory always contains the same data as the cache. But it is a slow process, since each time main memory needs to be accessed.

Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified or dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set. The philosophy of this method is based on the fact that during a write operation, the word residing in cache may be accessed several times (temporal locality of reference). This method reduces the number of references to the main

memory. However, this method may encounter the problem of inconsistency due to two different copies of the same data, one in cache and other in main memory.

Group-C
(Long-Answer Type Questions)

Answer any *three of* the following. $3 \times 15 = 45$

7. (a) What do you mean by Instruction Cycle, Machine Cycle and T-States?
   (b) Compare RISC with CISC.
   (c) What do you mean by Von Neumann bottleneck?

Specify possible strategies for handling it. $6 + 5 + 4$

Answer

(a) *Instruction cycle:* The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

Machine cycle: A machine cycle consists of necessary steps carried out to perform the memory access operation. Each of the basic operations such as fetch or read or write operation constitutes a machine cycle. An instruction cycle consists of several machine cycles.

T-states: One clock cycle of the system clock is referred to as T-state.

(b)

CISC	*RISC*
1. A large number of instruction types used—typically from 100 to 250 instructions.	1. Relatively few number of instruction types—typically less than100 instructions.
2. A large number of addressing modes used—typically from 5 to 15 different modes.	2. Relatively few addressing modes—typically less than or equal to 5.
3. Variable-length instruction formats.	3. Fixed-length, easily decoded instruction formats.
4. Small number of general-purpose registers (GPRs)—typically 8–24 GPRs.	4. Large number of general-purpose registers (GPRs)—typically 32–192 GPRs.
5. Clock per instruction (CPI) lies between 2 and 15.	5. Clock per instruction (CPI) lies between 1 and 2.
6. Mostly microprogrammed control units.	6. Mostly hardwired control units.
7. Most instructions manipulate operands in memory.	7. All operations are executed within registers of the CPU.

(c) One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck.

This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the

CPU, for which there is almost no waiting time of the CPU for the required data-word to come. Another way to reduce the problem is by using special types of computers known as *Reduced Instruction Set Computers (RISC)*. This class of computers generally uses a large number of registers, through which the most of the instructions are executed. This computer usually limits access to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

8. (a) With the help of a block diagram, describe the components of a microprogrammed control unit. Discuss the advantages and disadvantages of horizontal and vertical micro-instructions. What is a microprogram sequencer/control?

(b) What is bus arbitration? Explain clearly. (4 + 4 + 2) + 5

Answer

(a) The architecture of a typical modern microprogrammed control unit is shown in Fig. 5. This architecture was proposed by Maurice Wilkes in 1953.

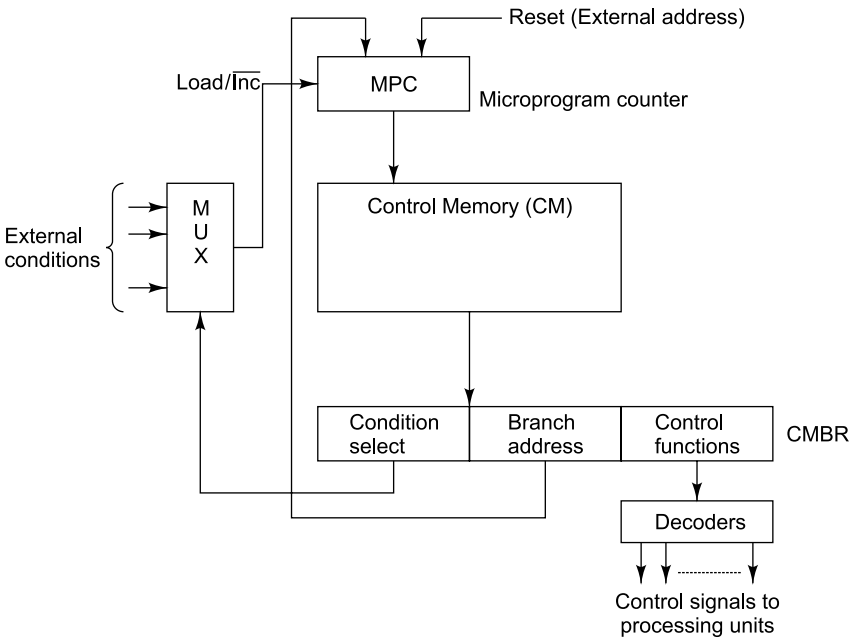


Figure 5 *General-purpose microprogrammed control unit*

The various components used in the figure are summarized next.

Control memory buffer register (CMBR): The function of CMBR is same as the MBR (memory buffer register) of the main memory. It is basically a latch and acts as a buffer for the microinstructions retrieved from the CM. Typically, each micro-instruction has three fields as

Condition select	Branch address	Control functions

The condition select field selects the external condition to be tested. The output of the MUX will be 1, if the selected condition is true. The MPC will be loaded with the address specified in the branch address field of the microinstruction, because the output of the MUX is connected to the load input of the Micro Program Counter (MPC). However, the MPC will point to the next microinstruction to be executed, if the selected external condition is false. Thus, this arrangement allows conditional branching. The control function field of the microinstruction may hold the control information in an encoded form which thus may require decoders.

Microprogram counter (MPC): The task of MPC is same as the PC (program counter) used in the CPU. The address of the next micro-instruction to be executed is held by the MPC. Initially, it is loaded from an external source to point to the starting address of the microprogram to be executed. From then on, the MPC is incremented after each micro-instruction fetch and the instruction fetched is transferred to the CMBR. However, the MPC will be loaded with the contents of the branch address field of the micro-instruction that is held in the CMBR, when a branch instruction is encountered.

External condition select MUX: Based on the contents of the condition select field of the microinstruction, this MUX selects one of the external conditions. Therefore, the condition to be selected must be specified in an encoded form. Any encoding leads to a short micro-instruction, which implies a small control memory; hence the cost is reduced. Suppose two external conditions X_1, X_2 are to be tested; then the condition-select and actions taken are summarized next:

Condition select	*Action taken*
00	No branching
01	Branch if $X_1 = 1$
10	Branch if $X_2 = 1$
11	Always branching (unconditional branching)

The multiplexer has four inputs V_0, V_1, V_2, V_3 where V_i is routed to the multiplexer's output when the condition select field has decimal equivalent i. Hence we require $V_0 = 0$, $V_1 = X_1$, $V_2 = X_2$, $V_3 = 1$ to control the loading of microinstruction branch addresses into MPC.

Advantages of horizontal micro-instructions:

(i) Ability to express a high degree of parallelism.

(ii) Very little encoding of the control information.

Disadvantage of horizontal micro-instructions:

(i) The length of the micro-instruction is large and thus the control memory size is huge.

Advantages of vertical micro-instructions:

(i) Short micro-instruction format and thus the overall of the control memory is small.

(ii) A single field can produce an encoded sequence.

Disadvantages of vertical micro-instructions:

(i) Limited ability to express parallel microoperations.

(ii) Considerable encoding of the control information. Therefore, this technique takes more time for generating the control signals due to the decoding time and also more microinstructions are needed.

The microprogramming approach is systematic, flexible, and less error-prone. Advances in IC technology have made LSI designers think of a general solution for implementing a microprogrammed CPU. A microprogrammed CPU has two major activities to be performed:

1. Fetching and interpreting micro-instructions
2. Generating the next address of the micro-instruction to be retrieved

The first task is assumed by the control memory and the associated circuit elements.

Designers have replaced the next address generation of a microprogrammed control unit with a single LSI component called a *microprogram sequencer,* which checks certain bits in the microinstruction and finds the next address for the control memory. The sequencer contains a microprogrammed counter (MPC) and circuit elements necessary to perform functions such as address incrementing, address sequencing for subroutine calls, returns, and conditional branching.

(c) A conflict may arise if the number of I/O devices or processors or memory unit try to access the common bus at the same time, but access can be given to only one of those. Only one processor or I/O device can be bus master. The bus master is the controller that has access to the bus at an instance. To resolve these conflicts, bus arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers. *Bus arbitration* refers to a process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus. The *bus arbiter* decides who would become current bus master. There are two approaches to bus arbitration:

1. *Centralized bus arbitration*: A single bus arbiter performs the required arbitration.
2. *Distributed bus arbitration*: All devices participate in the selection of the next bus master.

There are three bus arbitration methods:

1. Daisy Chaining Method.
2. Polling or Rotating Priority Method.
3. Fixed Priority or Independent Request Method.

9. (a) A CPU has 32-bit memory address and a 256 KB cache memory. The cache is organized as a 4-way set associative cache with cache block size of 16 bytes.
   (i) What is the number of sets in the cache?
   (ii) What is the size (in bits) of the tag field per cache block?
   (iii) What is the number and size of comparators required for tag matching?
   (iv) How many address bits are required to find the byte offset within a cache block?

   (b) What are the widths of data bus and address bus for (4096 × 8) memory? What do you mean by program status word? Define content addressable memory. What is control word?

$(4 \times 2) + (2 + 2 + 2 + 1)$

Answer

(a) (i) The of blocks in cache = (256 * 1024)/16 = 2^{14} = 16384.
Since the cache is 4-way set associative, the number of sets = $2^{14}/4 = 2^{12}$ = 4096.

(ii) Since cache has 4096 sets, the number bits required to select a set = 12.
Each block consists of 16 bytes.
Therefore, number of bits required to select a byte (word) = 4.

Since the CPU generates 32-bit address to access a byte (word) in memory, the number of bits of tag required in each entry in the tag array = 32 – (12 + 4) = 16.

(iii) As calculated previously, the number of bits required to select a byte (word) = 4.

(b) Given memory size is 4096 × 8. The width of data bus = 8-bit and that of address bus = 12-bit (because $4096 = 2^{12}$).

Program Status Word (PSW): It is a special purpose register that holds the condition code flags and other information that describe the status of the currently executing program. This register is also known as status register. Generally, there are five flags (i.e., status bits), namely, Carry (CY), Parity (P), Auxiliary Carry (AC), Zero (Z) and Sign (S) flags. The processor uses these flags to test data conditions.

Content Addressable Memory: The searching time for desired data stored in memory can be reduced largely if stored data can be searched only by the data value itself rather than by an address. The memory accessed by the data content is known as *associative memory or Content Addressable Memory (CAM)*. When a data is stored in this memory, no address is stored. At any first empty location, the data is stored. When a data word is to be read from the memory, only the data word or part of data called *key* is provided. The memory is sequentially searched thoroughly for match with the specified key and set them for reading next.

Control Word: In the microprogrammed control unit design approach, all control functions that can be simultaneously activated are grouped to form control words stored in a separate ROM memory called the control memory. From the control memory, the control words are fetched one at a time and the individual control fields are routed to various functional units to activate their appropriate circuits. The desired task is performed by activating these circuits sequentially.

10. (a) Draw and explain the flowchart for division of two binary numbers using non-restoring algorithm. Use the example of 8 to be divided by 3.

(b) Explain the difference between instruction pipeline and arithmetic pipeline.

(c) Why is Carry Look-Ahead Adder (CLA) called a fast parallel adder? What will be the delay if you construct a 16-bit CLA using 4-bit CLA blocks?

Answer

(a) In the restoring division method, some extra additions are required to restore the number, when A is negative. Proper restructuring of the restoring division algorithm can eliminate that restoration step. This is known as the non-restoring division algorithm.

The three main steps in restoring division method were

1. Shift AQ register pair to the left one position.
2. $A = A - M$.
3. If the sign of A is positive after the step 2, set $Q[0] = 1$; otherwise, set $Q[0] = 0$ and restore A.

Now, assume that the step 3 is performed first and then step 1 followed by the step 2. Under this condition, the following two cases may arise.

Case 1: When A is positive

Note that shifting A register to the left one position is equivalent to the computation of $2A$ and then subtraction. This gives the net effect on A as $2A - M$.

Case 2: When A is negative

First restore A by adding the content of M register and then shift A to the left one position. After that A will be subtracted from M register. So, all together they give rise the value of A as $2(A + M) - M = 2A + M$.

Basis on these two observations, we can design the non-restoring division method and it is described in the flowchart, as shown in Fig. 6.

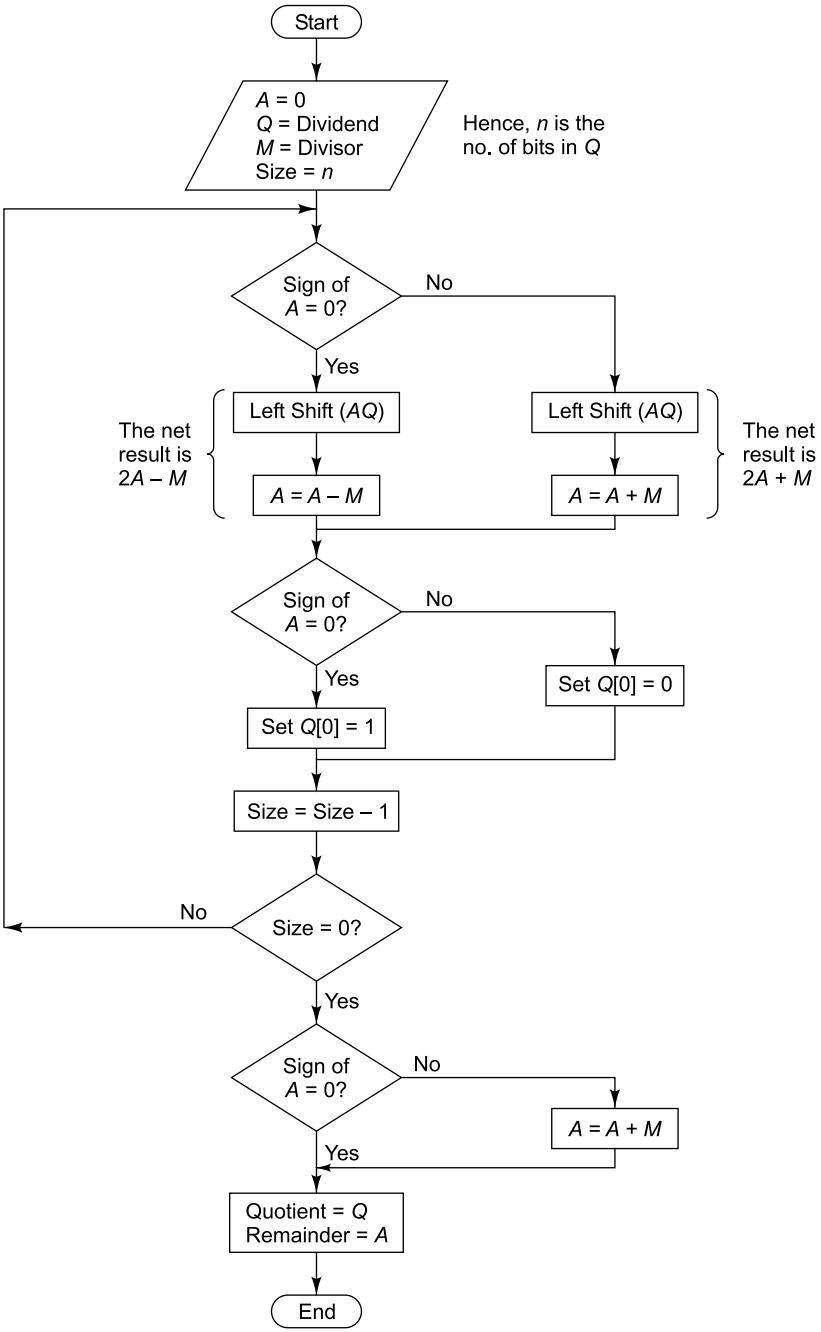


Figure 6 *Non-restoring division method*

This algorithm removes the restoration step, though it may require a restoration step at the end of algorithm for remainder A, if A is negative.

Example: we have dividend $Q = 8 = 1000$ and divisor $M = 3 = 0011$.

	M	A	Q	Size
Initial Configuration	00011	00000	1000	4
Step-1				
As sign of A = +ve				
LS(AQ)	00011	00001	000–	–
$A = A - M$	00011	11110	000–	–
As sign of A = –ve				
Set $Q[0] = 0$	00011	11110	0000	3
Step-2				
As sign of A = –ve				
LS(AQ)	00011	11100	000–	–
$A = A + M$	00011	11111	000–	–
As sign of A = –ve				
Set $Q[0] = 0$	00011	11111	0000	2
Step-3				
As sign of A = –ve				
LS(AQ)	00011	11110	000–	–
$A = A + M$	00011	00001	000–	–
As sign of A = +ve				
Set $Q[0] = 1$	00011	00001	0001	1
Step-4				
As sign of A = +ve				
LS(AQ)	00011	00010	001–	–
$A = A - M$	00011	11111	001–	–
As sign of A = –ve				
Set $Q[0] = 0$	00011	00010	0010	0
Restore A				

From the above last step, we conclude that quotient = 0010 = 2 and remainder = 00010 = 2.

(b)
1. Instruction pipeline is used to process all instructions, whereas arithmetic pipeline is used to process arithmetic type instructions such as addition, subtraction, multiplication, etc.
2. In instruction pipeline, the execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions. An arithmetic pipeline divides an arithmetic operation, such as a multiply, into multiple arithmetic steps each of which is executed one-by-one in different arithmetic stages in the ALU.
3. All high-performance computers are now equipped with instruction pipeline. The number of arithmetic pipelines varies from processors to processors.

(c) In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages. These input carries depend only on the initial carry C_0. For this reason, CLA is fast parallel adder.

The maximum delay for such adder can be computed as:

Δ (for G_i, P_i generation) + 2Δ (for C_4 generation from C_0) + 2Δ (for C_8 generation from C_4) + 2Δ (for C_{12} generation from C_8) + 2Δ (for C_{15} generation from C_{12}) + 3Δ (for S_{15} generation from C_{15}) = 12Δ.

11. (a) What is SRAM?
    (b) What is DMA?
    (c) What is the bandwidth of a memory system that transfers 128-bit data per reference having a speed of 20 nanoseconds per operation?
    (d) How do the following influence the performance of a virtual memory system?
        (i) Size of page
        (ii) Replacement policies of pages.
    (e) What is a floating point number? Write down the steps to subtract 110. 101101 from 10110. 1110 2 + 3 + 3 + 4 + 3

Answer

(a) The SRAM (static RAM) memories consist of circuits capable of retaining the stored information as long as power is applied. That means this type of memory requires constant power. SRAM memories are used to build cache memory.

(b) To transfer large blocks of data at high speed, this method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA)*.

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

(c) The bandwidth is the data transfer rate by the memory. It is expressed as number of bytes (words) per second.

Here, the memory system transfers 128-bit data per reference having a speed of 20 nano sec per operation.

Therefore, 128-bit data word is transferred in 20 ns.

So, the bandwidth of the memory system = 128/20 bits per ns

= $(128 * 10^9)/20$ bits per sec

= $64 * 10^8$ bits per sec

(d) (i) *Page size*: If page size is large, the page fault rate will be less. But, in that case, transfer time of the page will increase.

If page size is small, the memory is better utilized, but number of pages and hence the size of page table will be large.

(ii) *Replacement policy*: When a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several

replacement algorithms such as *FIFO (First-in First-out)*, *LRU (Least Recently Used)* and *optimal page replacement* algorithm available. The algorithm which gives lowest page faults is considered as best algorithm.

The *FIFO algorithm* is simplest and its criterion is "select a page for replacement that has been in main memory for longest period of time".

The *LRU algorithm* states that "select a page for replacement, if the page has not been used often in the past". The LRU algorithm is difficult to implement, because it requires a counter for each page to keep the information about the usage of page.

The *optimal algorithm* generally gives the lowest page faults of all algorithms and its criterion is "replace a page that will not be used for the longest period of time". This algorithm is also difficult to implement, because it requires future knowledge about page references.

An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a *reference string*.

(e) In floating-point representation, a number has two parts. The first part is called *mantissa or fraction*, to represent a signed fixed-point number, which may be a fraction or an integer. The second part is called *exponent or characteristic*, to designate the position of the radix point. For example, in floating-point representation, the decimal number +786.231 is represented with a mantissa and an exponent as follows:

Mantissa	Exponent
+ 0.786231	+ 03

This representation is equivalent to the scientific notation + $0.786231 \times 10^{+03}$.

If the integer system of representation for mantissa is used, the decimal number +786.231 is represented in floating-point with a mantissa and an exponent as follows:

Mantissa	Exponent
+ 786231	– 03

Subtraction of 110. 101101 from 10110. 1110:

110. 101101 = .1101 01101 $\times 2^{+3}$

10110. 1110 = .1011 01110 $\times 2^{+5}$

We select the larger exponent (i.e., +5) as the exponent of the result.

Therefore, the mantissa of the number with smaller exponent, i.e., .1101 01101 is shifted right 2 positions (equals to the difference in the exponents).

So, this becomes .001101 01101.

Now we subtract .001101 01101 from .1011 01110 and this gives rise to .01111 111011

This mantissa indicates that it is non-normalized number, because of a leading 0.

So, after normalization, the resultant mantissa becomes .11111 11011 and its exponent becomes equivalent to +4.

2010
Computer Organization and Architecture
CS-404 (EI)

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) How many bits are needed to represent a digit in hexadecimal notation?
(a) 8 (b) 16 (c) 4 (d) 2.

Answer
(c) 4

(ii) How many RAM chips (each of 128×4) are required to provide a memory capacity of 2048 bytes?
(a) 32 (b) 16 (c) 8 (d) 64

Answer
(a) 32

(iii) Principle of locality is justified in the use of
(a) daisy chaining (b) DMA (c) interrupts (d) Cache memory

Answer
(d) Cache memory

(iv) Range of values on a data bus of an 8-bit microprocessor using 2's complement representation will be
(a) – 128 to + 128 (b) – 128 to + 127 (c) – 127 to + 128 (d) – 127 to + 127

Answer
(b) – 128 to + 127

(v) Bidirectional buses use
(a) two back-to-back connected buffers

(b) two tri-state buffers in cascade
(c) tri-state buffers
(d) two tri-state buffers back-to-back connected in parallel .

Answer

(d) two tri-state buffers back-to-back connected in parallel

(vi) How many memory locations can be accessed by a 32-bit computer?
(a) 64 KB (b) 32 KB (c) 4 GB (d) None of these

Answer

(c) 4 GB

(vii) Highest speed logic gate among the following is
(a) TTL (b) DTL (c) RTL (d) ECL.

Answer

(d) ECL

(viii) A 'hit' is considered when a
(a) word is found in the cache (b) word is not found in the cache
(c) word is found in the virtual memory (d) word is not found in the virtual memory

Answer

(a) word is found in the cache

(ix) Which one of the following is volatile in nature?
(a) ROM (b) DVD-ROM (c) CD-ROM (d) RAM.

Answer

(d) RAM

(x) Using binary arithmetic, the unique representation of zero is
(a) sign magnitude (b) 2's complement (c) 1's complement (d) none of these

Answer

(b) 2's complement

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. Explain an interrupt cycle with a flowchart.

Answer

To process interrupts, an interrupt cycle is added to the instruction cycle (which consists of two major cycles: fetch and execute, here), as shown in Fig. 1. In the interrupt cycle, the CPU checks to verify if any interrupts have occurred, indicated by the presence of an interrupt signal. If no interrupts are pending, the CPU proceeds to the fetch cycle and fetches the next instruction of the current program as usual. If an interrupt is pending, the CPU does the following:

1. It suspends execution of the current program being executed.
2. It saves the context of the current program being executed. This means saving the address of the next instruction to be executed and any other data relevant to current activity of the CPU.

3. It sets PC (program counter) to start the address of interrupt handler (i.e., interrupt service routine) to service the interrupt.
4. Then process interrupt.
5. After the completion of interrupt handler, the CPU resumes execution of the interrupted program.

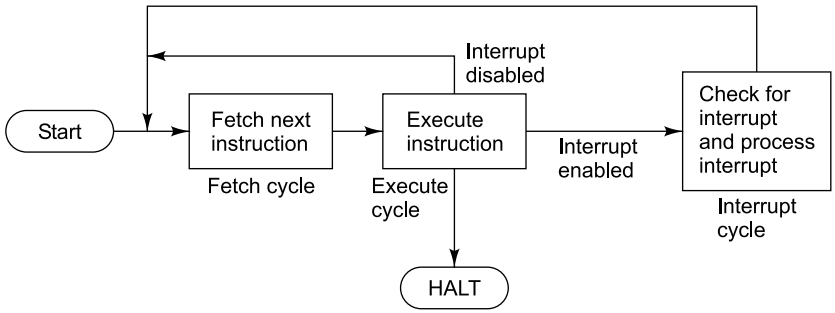


Figure 1 *Instruction cycle with interrupts*

Interrupts are not always handled immediately. The CPU has authority to disable all or selected interrupt signals and subsequently enable them. A disabled interrupt simply means that the CPU can and will ignore that interrupt request signal.

For example, it is generally desirable to finish the processing of one interrupt before taking on another. Thus, interrupts are often disabled while the CPU is processing an interrupt. If an interrupt occurs during this time, it generally remains pending and will be checked by the CPU after the CPU has enabled interrupt.

A simple flowchart of interrupt cycle is shown in Fig. 2.

3. What is the difference between hardwired control and microprogrammed control?

Answer

(i) Microprogrammed control provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions). Logic gates, flip flops, decoders and other digital circuits are used to implement hardwired control organization.

(ii) With microprogramming, many additions and changes are made by simply changing the microprogram in the control memory. A small change in the hardwired approach may lead to redesigning the entire system.

(iii) The microprogramming approach is more expensive than hardwired approach. Since a control ROM memory is needed in the microprogramming approach.

(iv) Microprogrammed control unit design method has been followed in modern-day control units of processors, and the hardwired approach is almost not followed nowadays.

4. What is a multiprocessor? Write briefly about the Harvard architecture. 1 + 4

Answer

A multiprocessor is a single computer consisting of multiple processors, which may communicate and cooperate at different levels in solving a given problem. The communication may occur by sending messages from one processor to the other or by sharing a common memory.

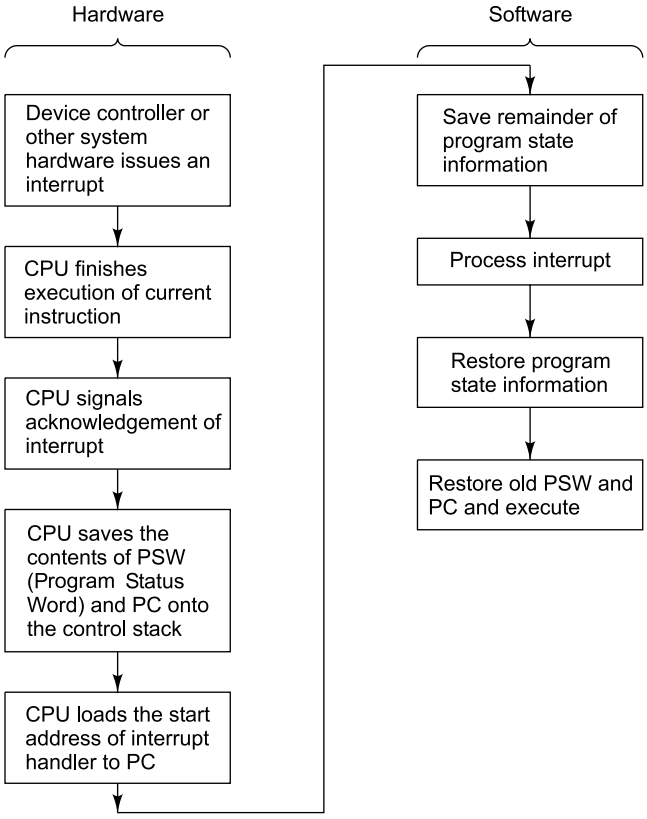


Figure 2 *Flowchart of a simple interrupt cycle*

In 1944, Howard Aiken of Harvard University developed a computer (named Automatic Controlled Calculator and later Harvard Mark I) which used two separate memories, one for program storage (on punched tape) and other for data storage (on relay latches).

Harvard architecture uses physically separate memories for their instructions and data, requiring dedicated buses for each of them (see Fig. 3). Thus instructions and data can be fetched simultaneously.

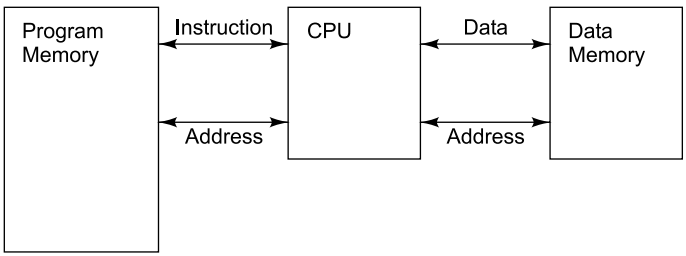


Figure 3 *Harvard architecture*

Program memory and data memory can be of different widths, type, etc. Both memories can be accessed at the same time using separate buses. Thus, Harvard computers are faster

than Von Neumann computers for a given circuit complexity. This architecture has been followed in modern-day systems like Digital Signal Processors (DSP) and microcontrollers.

The instruction format of the Harvard Mark I machine was

$$\text{ADDRESS}_1 \ \text{ADDRESS}_2 \ \text{OPCODE}$$

where ADDRESS_1 and ADDRESS_2 specified the registers storing the operands while ADDRESS_2 also specified the destination register where the result could be stored. OPCODE specified the operation (add, subtract or multiplication, etc.) to be performed. The storage had the capacity to store seventy-two 23-digit decimal numbers

5. With the help of a diagram explain clearly the structure and working of a typical arithmetic pipeline to perform : $X * Y + Z$.

Answer

The sub-operations to be performed for the arithmetic expression $X * Y + Z$ in each stage of the pipeline are as follows:

Sub-operation-1: R1 ← X, R2 ← Y	Input X and Y
Sub-operation-2: R3 ← R1 * R2, R4 ← Z	Multiply and input Z
Sub-operation-3: R5 ← R3 + R4	Add Z to product

Five registers are loaded with the new data in every clock period. The corresponding pipeline is shown in Fig. 4.

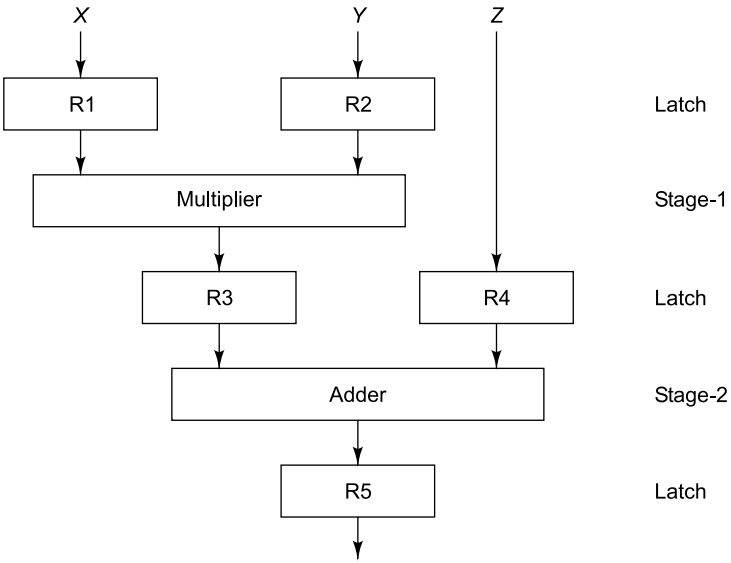


Figure 4 *Pipeline processing for $X * Y + Z$*

Group-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

6. (a) Show the bus connection with a CPU to connect four RAM chips of size 256×8 bits each and a ROM chip of 512×8 bit size. Assume the CPU has 8-bit data bus and 16-bit address bus. Clearly specify generation of chip select signals.

(b) What is an instruction cycle? Describe the steps of an instruction cycle with a suitable diagram.
(c) What are the advantages of interrupt-initiated I/O over programmed I/O? 6 + 6 + 3

Answer

(a) Figure 5 shows the interconnection diagram of these memories with the CPU having 16 address lines.

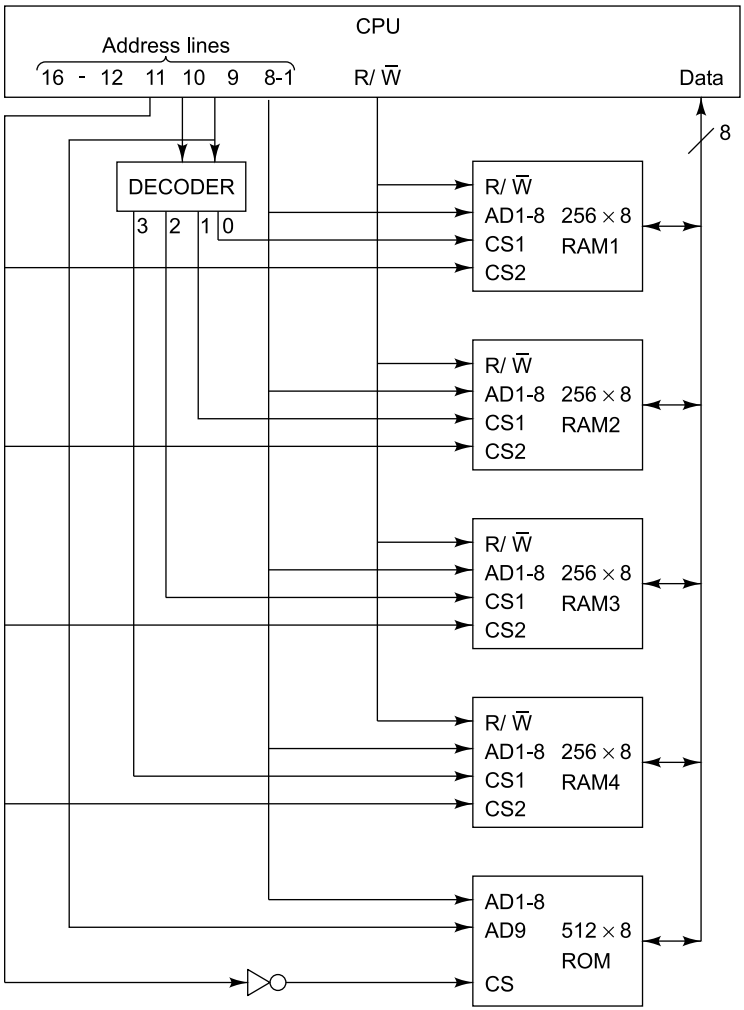


Figure 5 *Interconnection diagram*

The address lines 1 to 8 are connected to each memory and address line 9 is used in dual purposes. In case of a RAM selection out of four RAMs, the line no. 9 and line no. 10 are used through a 2-to-4 decoder. The line no. 9 is also connected to the ROM as address line along with lines 1 to 8 giving a total of 9 address lines in the ROM, since the ROM has 512 locations. The CPU address line number 11 is used for separation between RAM and ROM. The other 12 to 16 lines of CPU are unused and for simplicity we assume that they carry 0s as address signals. For ROM, 10^{th} line is unused and thus it can be assumed that this line carries signal 0.

The address map table for the memory connection to the CPU shown in Fig. 6 is constructed in the following table.

Chip selected	*Address space (in HEX)*	*Address bus*									
		11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
RAM1	0400 – 04FF	1	0	0	x	x	x	x	x	x	x	x
RAM2	0500 – 05FF	1	0	1	x	x	x	x	x	x	x	x
RAM3	0600 – 06FF	1	1	0	x	x	x	x	x	x	x	x
RAM4	0700 – 07FF	1	1	1	x	x	x	x	x	x	x	x
ROM	0000 – 01FF	0	0	x	x	x	x	x	x	x	x	x

(b) The processing required for each instruction of a program is called *instruction cycle*. The control unit's task is to go through an instruction cycle (see Fig. 6) that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

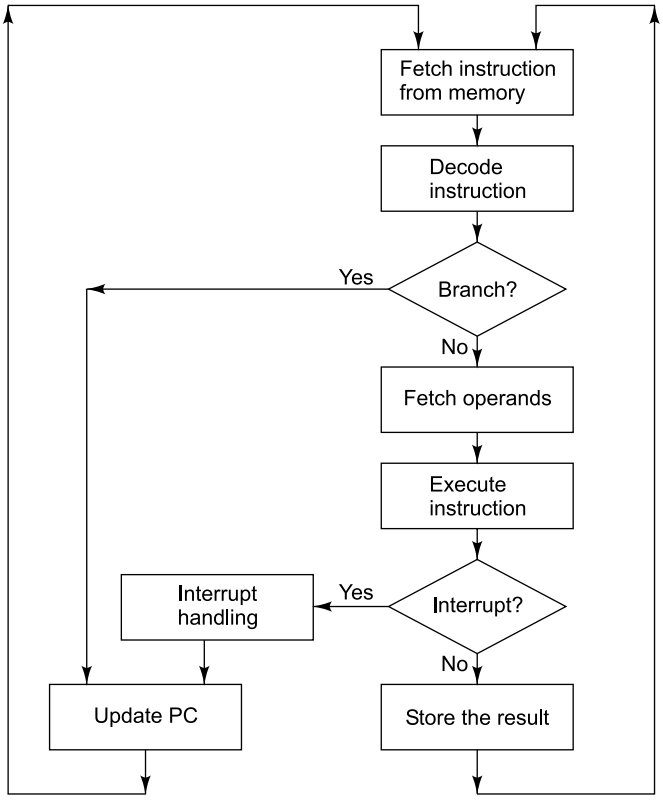


Figure 6 *Instruction cycle*

The step 1 is basically performed using a special register in the CPU called *program counter* (PC) that holds the address of the next instruction to be executed. If the current instruction is simple arithmetic/logic or load/store type, the PC is automatically incremented. Otherwise, the PC is loaded with the address dictated by the currently executing instruction. The decoding done in the step 2 determines the operation to be performed and the addressing mode of the instruction for calculation of address of operands. After getting the information about the addresses of operands, the CPU fetches the operands in the step 3 from memory or registers and stores them in its registers. In the step 4, the ALU of the processor executes the instruction on the stored operands in the registers. After the execution of instruction, in the phase 5, the result is stored back in the memory or register and returns to the step 1 to fetch the next instruction in sequence. All these sub-operations are controlled and synchronized by the control unit.

(c) In the programmed I/O method, the program constantly monitors the device status. Thus, the CPU stays in the program until the I/O device indicates that it is ready for data transfer. This is a time-consuming process since it keeps the CPU busy needlessly. It can be avoided by letting the device controller continuously monitor the device status and raise an interrupt to the CPU as soon as the device is ready for data transfer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to an *interrupt-service-routine (ISR)* or *I/O routine* or *interrupt handler* to process the I/O transfer, and then returns to the task it was originally performing. Thus, in the interrupt-initiated mode, the ISR software (i.e., CPU) performs data transfer but is not involved in checking whether the device is ready for data transfer or not. Therefore, the execution time of the CPU can be optimized by employing it to execute normal program, when no data transfer is required.

7. (a) What are the different types of DMA controllers and how do they function?
   (b) Briefly describe pipeline hazards.
   (c) What is the difference between a carry-look ahead adder and a ripple carry adder ?
   (d) What are the bottlenecks of Von Neumann concept? 5 + 5 + 3 + 2

Answer

(a) DMA controllers are of two types:
   – Independent DMA controller
   – DMA controller having multiple DMA-channels

Independent DMA controller:
For each I/O device a separate DMA controller is used. Each DMA controller takes care of supporting one of the I/O controllers. A set of registers to hold several DMA parameters is kept in each DMA controller. Such arrangement is shown in Fig. 7 for floppy disk controller (FDC) and hard disk controller (HDC). DMA controllers are controlled by the software.

DMA controller having multiple DMA-channels:
In this type of DMA controller, only one DMA controller exists in the system, but this DMA controller has multiple sections or channels each channel is for one I/O device. In this case, the software deals each channel in the same way. Multiple DMA channels in a DMA controller work in overlapped fashion, but not in fully parallel mode since they are embedded in a single DMA controller. Such DMA controller design technique is adopted in most of the computer system and is shown in Fig. 8 for floppy disk controller (FDC) and hard disk controller (HDC).

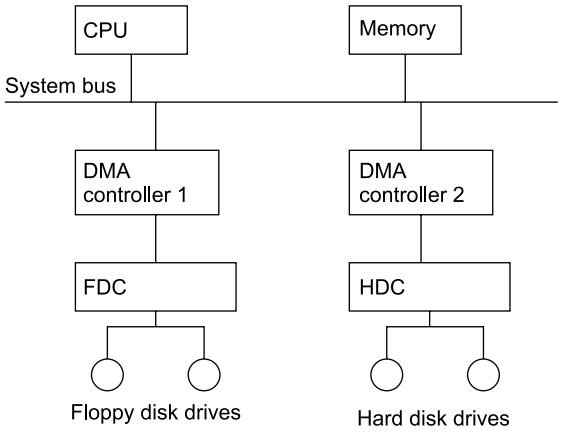


Figure 7 *Independent DMA controller*

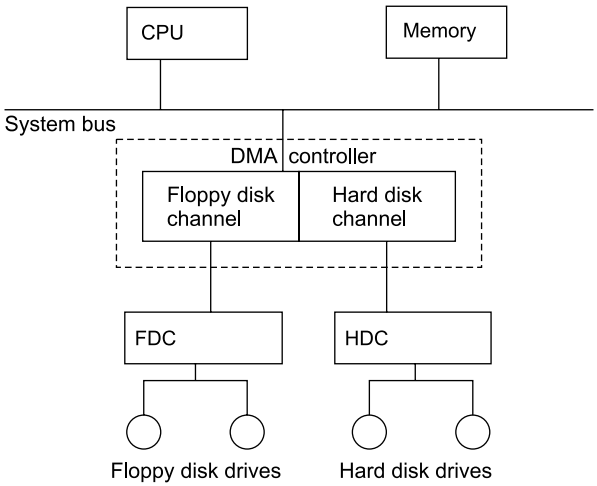


Figure 8 *DMA controller having multiple DMA channels*

(b) Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. There are three types of pipeline hazards:

1. Control hazards
2. Structural hazards
3. Data hazards

Control hazards:

They arise from the pipelining of branches and other instructions that change the content of the program counter (PC) register.

Structural Hazards:

Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.

Data Hazards:
Inter-instruction dependencies may arise to prevent the sequential (in-order) data flow in the pipeline, when successive instructions overlap their fetch, decode and execution through a pipeline processor. This situation due to inter-instruction dependencies is called *data hazard*.

(c) (i) In the Carry Look-ahead Adder (CLA), carry inputs of all stages are generated simultaneously, without using carries from the previous stages. In the Ripple Carry Adder (RCA), carry input of any stage depends on carry out from the previous stage.

(ii) Generally, the CLA is faster than the RCA. Because, the maximum delay of the CLA is $6 \times \delta$, where δ is the average gate delay and this holds good for any size numbers. However, the maximum propagation delay for n-bit RCA is $\delta \times n$, where n is the number of bits in each operand.

(iii) The RCA becomes slow once the sizes of operands are increased, which is not true for CLA. The speed of CLA remains same irrespective of sizes of operands.

(iv) The hardware cost of the CLA is more than that of RCA for same-size number addition.

(d) One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU memory speed disparity is referred to as Von-Neumann bottleneck. This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required data word to come. Another way to reduce the problem is by using special types of computers known as *Reduced Instruction Set Computers (RISC).* This class of computers generally uses a large number of registers, through which most of the instructions are executed. This computer usually limits access to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

8. (a) A hierarchical cache-main memory sub-system has the following specifications: cache access time is 50 ns, main memory access time is 500 ns, 80% of memory request for read, hit ratio 0.9 for read access and write-through scheme is used.

(i) Calculate the average access time of the memory system considering only memory read cycle.

(ii) Calculate the average access time of the memory system both for read and write.

(b) Explain clearly the procedure of virtual address translation into real address in a paged virtual memory system.

(c) What is the difference between associative and set-associative mappings? 6 + 6 + 3

Answer

(a) Given,
Cache access time t_c = 50 ns.
Main memory access time t_m = 500 ns
Probability of read p_r = 0.8
Hit ratio for read access h_r = 0.9
Writing scheme: write-through.

(i) Considering only memory read cycle,
The average access time $t_{av-r} = h_r * t_c + (1 - h_r) * (t_c + t_m)$
$= 0.9 * 50 + (1 - 0.9) * 550$
$= 100$ ns

(ii) For both read and write cycles,
The average access time = $p_r * t_{av-r} + (1 - p_r) * t_m$. Since in write-through method, access time for write cycle will be the main memory access time.
$= 0.8 * 100 + (1 - 0.8) * 500$
$= 180$ ns

(b) Paging is a non-contiguous memory allocation method. In other words, the program is divided into small blocks in paging and these blocks are loaded elsewhere in the main memory. In paging, the virtual address space is divided into equal-size blocks called *pages* and the physical (main) memory is divided into equal size blocks called *frames*. The size of a page and size of a frame are equal. The size of a page or a frame is dependent on the operating system and is generally 4 KB.

In paging, operating system maintains a data structure called *page table*, which is used for mapping from logical address to physical address. The page table generally contains two fields, one is page number and other is frame number. The table specifies the information that which page would be mapped to which frame. Each operating system has its own way of maintaining the page tables; most allocate a page table for each program.

Each address generated by the CPU (i.e., virtual address) is divided into two parts: *page number (p)* and *offset or displacement (d)*. The page number *p* is used as index in the page table and the offset *d* is the word number within the page *p*. The structure of paging method is shown in Fig. 9.

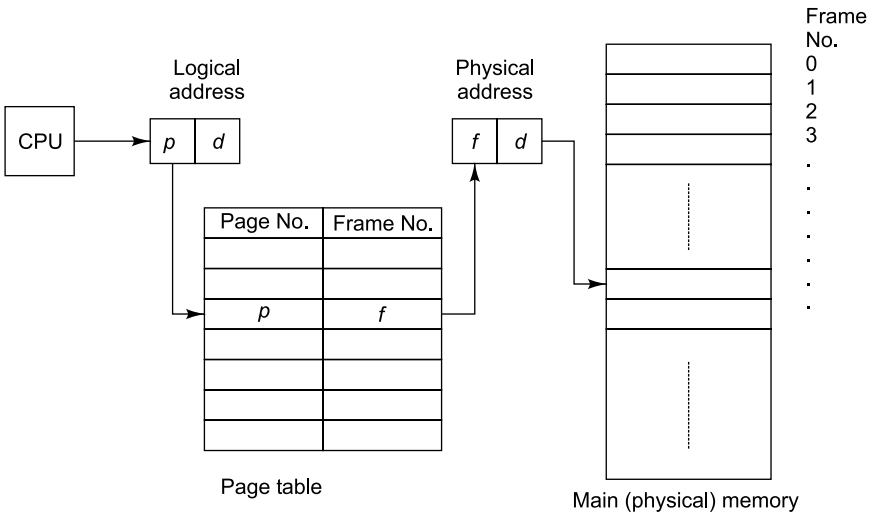


Figure 9 *Paging structure*

(c) (i) The associative cache memory uses the fastest and most flexible mapping method, in which both address and data of the memory word are stored whereas, in set-associative

cache, two or more words can be stored under the same index address, which is not stored in the memory. Each data word is stored together with its tag. The number of tag-data words under an index is said to form a *set.*

(ii) The set-associative cache has a higher hit ratio compared to associative cache.

(iii) The set-associative cache is the most expensive memory. The cost increases as set size increases.

9. (a) Using Booth's algorithm multiply (–3) and (–5) up to five digits. Show every step.

(b) Evaluate the following statement using zero address and two-address machines: $Z = (M + N) * (P + Q)$.

(c) Explain Flynn's classification with respect to computer architecture. 6 + 5 + 4

Answer

(a) $M = -5 = 11011$ and $Q = -3 = 11101$.

	M	A	Q	Size
Initial Configuration	11011	00000	11101 0	5
Step-1				
As $Q[0] = 1$and $Q[-1] = 0$				
$A = A - M$	11011	00101	11101 0	–
And ARS(AQ)	11011	00010	11110 1	4
Step-2				
As $Q[0] = 0$ and $Q[-1] = 1$				
$A = A + M$	11011	11101	11110 1	–
ARS(AQ)	11011	11110	11111 0	3
Step-3				
As $Q[0] = 1$ and $Q[-1] = 0$				
$A = A - M$	11011	00011	11111 0	–
ARS(AQ)	11011	00001	11111 1	2
Step-4				
As $Q[0] = 1$ and $Q[-1] = 1$				
ARS(AQ)	11011	00000	11111 1	1
Step-5				
As $Q[0] = 1$ and $Q[-1] = 1$				
ARS(AQ)	11011	00000	01111 1	0

Since the Size register becomes 0, the algorithm is terminated and the product is = AQ = 00000 01111, which shows that the product is a positive number. The result is 15 in decimal.

(b) To evaluate the statement $Z = (M + N) * (P + Q)$ in zero and two address machines, we assume the following assumptions:

LOAD symbolic op-code is used for transferring data to register from memory. STORE

symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations of addition and multiplication respectively. Assume that the respective operands are in memory addresses *M*, *N*, *P* and *Q* and the result must be stored in the memory at address *Z*.

For zero-address machine

The assembly-language program using zero-address instructions is written next. In the comment field, the symbol TOS is used, which means the top of stack.

```
PUSH M          ; TOS ← M
PUSH N          ; TOS ← N
ADD             ; TOS ← (M + N)
PUSH P          ; TOS ← P
PUSH Q          ; TOS ← Q
ADD             ; TOS ← (P + Q)
MULT            ; TOS ← (M + N) * (P + Q)
POP Z           ; Z ← TOS
```

For two-address machine

The assembly-language program using two-address instructions is written below.

```
LOAD R1, M      ; R1 ← M[M]
ADD R1, N       ; R1 ← R1 + M[N]
LOAD R2, P      ; R2 ← M[P]
ADD R2, Q       ; R2 ← R2 + M[Q]
MULT R1, R2     ; R1 ← R1 * R2
STORE Z, R1     ; Z ← R1
```

(c) Based on the number of simultaneous instruction and data streams used by a CPU during program execution, digital computers can be classified into four categories. This scheme for classifying computer organizations was proposed by Michael J Flynn. The objective of a processor (CPU) is the execution of a sequence of instructions on a set of data. The term *stream is* used here to denote a sequence of items (instructions or data) as executed or operated upon by a single CPU. *Instructions* or *data* are defined with respect to a given processor. An *instruction stream is* a sequence of instructions as executed by the processor; a *data stream is* a sequence of data including input, partial, or temporary results, called for by the instruction stream.

The multiplicity of the ALU and CU provided to service the instruction and data streams controls computer organization. Flynn's four-machine organizations are as follows:

- Single instruction stream-single data stream (SISD) machine
- Single instruction stream-multiple data stream (SIMD) machine
- Multiple instruction stream-single data stream (MISD) machine
- Multiple instruction stream-multiple data stream (MIMD) machine

Both instructions and data are fetched from the *memory modules.* Instructions are decoded by the *control unit,* which sends the decoded instruction stream to the *processor elements* (ALUs) for execution. Data streams flow between the processor elements and the memory bi-directionally. A shared memory subsystem, consisting of multiple memory modules, can be used in a machine. Each instruction stream is generated in the form of control signals by an independent control unit. The

shared memory subsystem generates multiple data streams simultaneously.

SISD Computer

Most serial computers available today fall in this organization as shown in Fig. 10. Instructions are executed sequentially but may be overlapped in their execution stages (In other words the technique of pipelining can be used in the CPU). Modern-day SISD uniprocessor systems are mostly pipelined. A SISD computer may have more than one functional unit in it, but all are under the supervision of one control unit. This type of machines can process only scalar type instructions.

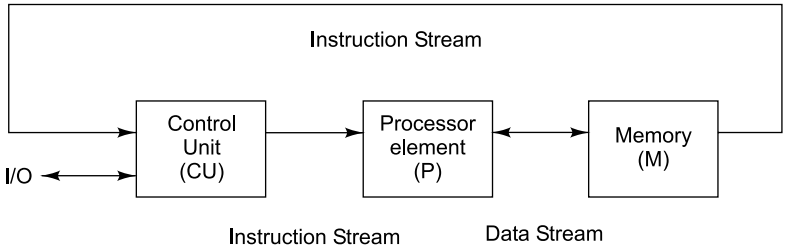


Figure 10 *SISD Computer*

SIMD Computer

Array processors fall into this class. As illustrated in Fig. 11, there are multiple processing elements supervised by the common control unit. All PEs (processing elements, which are essentially ALUs) receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams. The shared memory subsystem containing multiple modules is very essential. This machine generally used to process vector-type data.

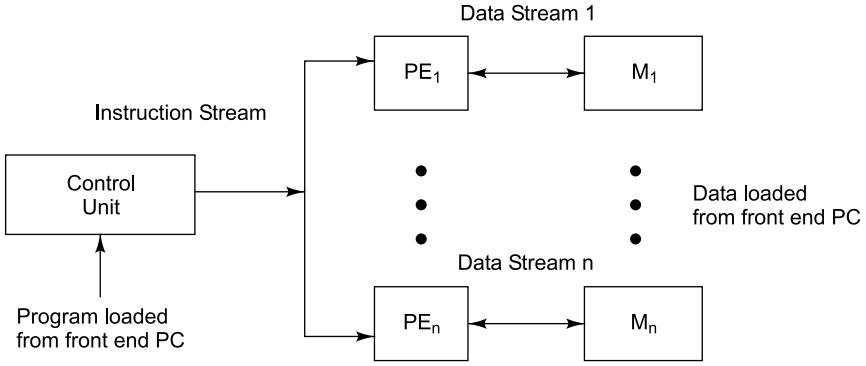


Figure 11 *SIMD computer*

MISD Computer

Very few or no parallel computers fit in this organization, which is conceptually illustrated in Fig. 12. There are *n* processor elements, each receiving distinct instructions to execute on the same data stream and its derivatives. The results (outputs) of one processor element become the inputs (oper-

ands) of the next processor element in the series. This architecture is also known as *systolic arrays*. This structure has received much less attention, though some fault-tolerance machines can be used in this class. Thus, in general no practical machine of this class exists.

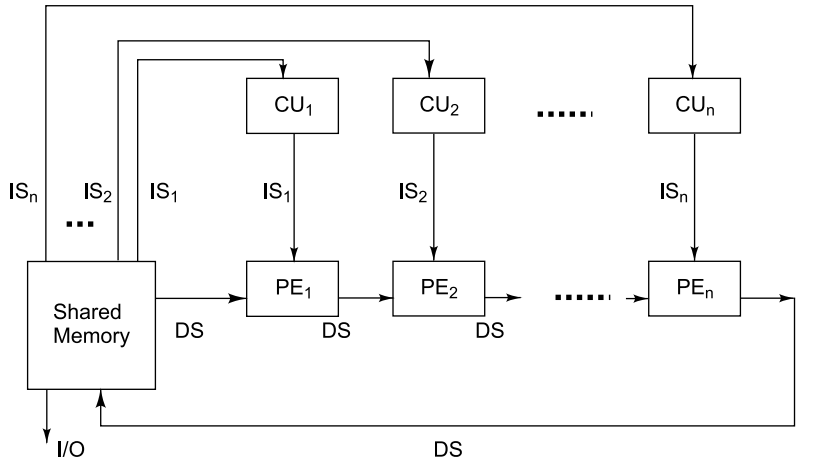


Figure 12 *MISD computer (Systolic array)*

MIMD Computer

This category covers multiprocessor systems and multiple computer systems (Fig. 13). An MIMD computer is called *tightly coupled (or Uniform Memory Access (UMA))* if the degree of interactions among the processors is high. Otherwise, we consider them *loosely coupled (or Non-Uniform Memory Access (NUMA))*. Most commercial MIMD computers are loosely coupled.

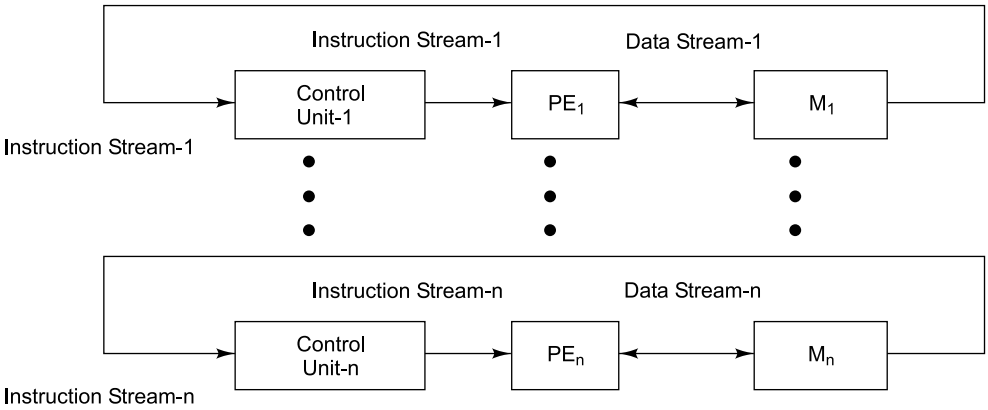


Figure 13 *MIMD computer*

Listed below are several system models under each of the three existing computer organizations under Flynn's classification.

Computer class	*Computer system models*
SISD (using one functional unit)	IBM 701; IBM 1620; IBM 7090; PDP VAX-11/780.
SISD (with multiple functional units)	IBM 360/91; CDC Star-100; TI-ASC; Cray-I; Fujitsu VP-200.
SIMD	Illiac-IV; PEPE; BSP.
MIMD	IBM 370/ 168 MP; Univac 1100 / 80;
(Loosely coupled)	Tandem 16; C.m*.
MIMD	C.mmp; Cray-3; S-1; Cray–X MP;
(Tightly coupled)	Denelcor HEP.

10. (a) Design and describe the function of a control unit with block diagram for a typical computer having 16-bit instruction register.
    (b) Describe briefly the different addressing modes. 9 + 6

Answer

(a) Figure 14 shows a general model of the control unit with all its inputs and outputs. The inputs are as follows:

- *Clock*: In each clock period, the control unit causes one micro-operation or a group of simultaneous micro-operations to be executed.
- *Instruction register*: This register holds the op-code of the current instruction to be executed that determines which micro-operations to perform during the execution cycle.
- *Flags*: The control unit needs these to determine the status of the CPU and the outcome of the previous ALU operations. For example, the control unit will branch to a specific location in memory if the zero flag is set, for the instruction "branch on zero (BZ)".
- *Control signals from control bus*: These are provided to the control unit by the control bus portion of the system bus. These include signals like interrupt signals and acknowledgements.

The outputs are as follows:

- *Control signals within the processor*: These signals are of two types—one that activates specific ALU functions and other that causes data to be moved from one register to another register.
- *Control signals to control bus*: These signals are also of two types—one for memory and other for I/O devices.

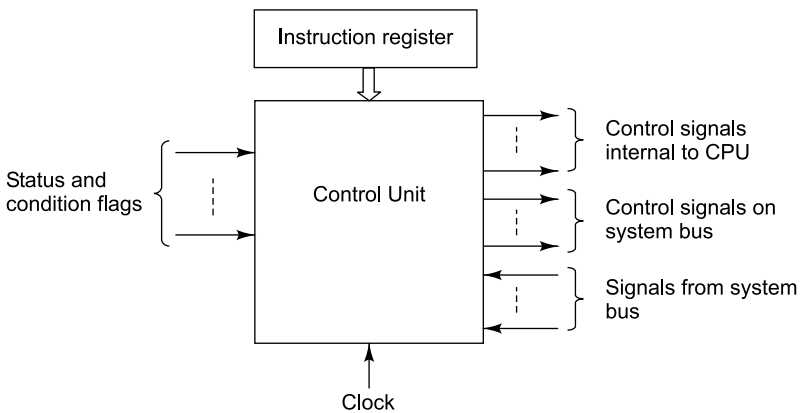


Figure 14 *General model of the control unit*

(b) The ALU of the CPU executes the instructions as dictated by the op-code field of instructions. The instructions are executed on some data stored in registers or memory. The different ways in which the location of an operand is specified in an instruction are referred to as *addressing modes*. A computer uses variety of addressing modes; some of them are described below:

1. *Implied (or Inherent) Mode* In this mode, the operands are indicated implicitly by the instruction. The accumulator register is generally used to hold the operand and after the instruction execution the result is stored in the same register. For example,

   RAL; Rotates the content of the accumulator left through carry

2. *Immediate Mode* In this mode, the operand is mentioned explicitly in the instruction. In other words, an immediate-mode instruction contains an operand value rather than an address of it in the address field. To initialize registers to a constant value, this mode of instructions is useful. For example:

   MVI A, 06; Loads equivalent binary value of 06 to the accumulator

3. *Register (Direct) Mode* In this mode, the processor registers hold the operands. In other words, the address field is now a register field, which contains the operands required for the instruction.
   For example,

   ADD R1, R2; Adds contents of registers R1 and R2 and stores the result in R1

4. *Register Indirect Mode* In this mode, the instruction specifies an address of CPU register that holds the address of the operand in memory.
5. *Direct (or Absolute) Address Mode* In this mode the instruction contains the memory address of the operand explicitly. Example of direct addressing is

   STA 2500H; Stores the content of the accumulator in the memory location 2500H.

6. *Indirect Address Mode*: In this mode, the instruction gives a memory address in its address field which holds the address of the operand.
   For example,

   MOV R1, (X) ; Content of the location whose address is given in X is loaded into register R1.

7. *Relative Address Mode or PC-relative Address Mode* In this mode, the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address.
8. *Indexed Address Mode* In this mode, the effective address is determined by adding the content of index register (XR) with the address part of the instruction. This mode is useful in accessing operand array. The address part of the instruction gives the starting address of an operand array in memory. The index register is a special CPU register that contains an index value for the operand. The index value for operand is the distance between the starting address and the address of the operand. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. For example, an operand array starts at memory address 1000 and assume that the index register XR contains the value 0002. Now consider load instruction

LDA 1000

The effective address of the operand is calculated as

Effective address = 1000 + content of XR = 1002

9. *Base Register Address Mode* This mode is used for relocation of the programs in the memory. *Relocation* is a technique of moving program or data segments from one part of memory to another part of memory. Relocation is an important feature of multiprogramming systems. In this mode the content of the base register (BR) is added to the address part of the instruction to obtain the effective address. This mode is similar to the indexed addressing mode, but exception is in the way they are used. A base register holds the starting address of a memory array of operands and the address part of the instruction gives a displacement or offset relative to this starting address. The base register addressing mode has the advantage over index addressing mode with respect to the size of instructions, because size of instructions in first case is smaller than that of second category.

2011
Computer Organization
(CS-303 (New))

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) How many address bits are required for a 1024×8 memory?

(a) 1024 (b) 5 (c) 10 (d) none of these

Answer

(c) 10

(ii) Micro instructions are kept in the

(a) main memory (b) cache memory

(c) control memory (d) none of these

Answer

(c) control memory

(iii) Booth's algorithm for computer arithmetic is used for

(a) multiplication of numbers in signed magnitude form

(b) division of numbers in signed magnitude form

(c) multiplication of numbers in 2's complement form

(d) division of numbers in 2's complement form

Answer

(c) multiplication of numbers in 2's complement form

(iv) In a microprocessor, the address for the next executable instruction is stored in the

(a) stack pointer (b) program counter

(c) instruction register (d) none of these.

Answer

(b) program counter

(v) A single bus structure is primarily found in
(a) mini and micro computers (b) large mainframe computers
(c) super computers (d) analog computers

Answer

(a) Mini and micro computers

(vi) Cache memory is used to
(a) increase performance (b) increase machine cycles
(c) decrease performance (d) none of these

Answer

(a) increase performance

(vii) Instruction cycle is
(a) fetch-decode-execution (b) fetch-execution-decode
(c) fetch-encode-execution (d) fetch-execution-encode

Answer

(a) fetch-decode-execution

(viii) Equivalent hexadecimal of $(76575372)_8$ will be
(a) FAFAFF (b) FAFAFA (c) FFFAAA (d) FAAFAF

Answer

(b) FAFAFA

(ix) Associative memory is
(a) a very cheap memory (b) pointer addressable memory
(c) content addressable memory (d) all of these

Answer

(c) content addressable memory

(x) Which of the following addressing mode is used for the instruction "Push B"?
(a) Register (b) Register indirect (c) Direct (d) Immediate

Answer

(d) Immediate

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. Explain the difference between full associative and direct mapped cache mapping approaches. Explain "write through" and "write back" policies in cache. 3 + 2

Answer

The fully associative cache memory uses the fastest and most flexible mapping method in which both address and data of the memory word are stored. This memory is expensive because of additional storage of addresses with data in the cache memory.

In the direct cache mapping, instead of storing total address information with data in cache, only parts of address bits are stored along with data. Suppose the cache memory can

hold 2^m words and the main memory can hold 2^n words. The n-bit address generated by the CPU is divided into two fields: lower-order m bits for the index field and the remaining higher-order (n – m) bits for the tag field. The direct mapping cache organization uses the m-bit index to access the cache and higher-order ($n – m$) bits of tag are stored along side the data in cache. This is the simplest type of cache mapping, since only tag field is required to match. That's why it is one of the fastest caches. Also, it is less expensive cache relative to the associative cache.

There are two policies in writing into cache memory: (i) write-through (ii) write-back.

Write-Through Policy: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus the main memory always contains the same data as the cache.

Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified* or *dirty bit*. When the word is replaced from the cache, it is written into the main memory if its flag bit is set.

3. Differentiate among three-, two-, one- and zero- address instructions with suitable examples. Explain base index addressing with example. 3 + 2

Answer

The size of programs consisting of all three-address instructions is small, whereas that of programs using zero-address instructions is large. Three- and two-address instructions are generally used in general-register organized processors, one-address instructions used in single accumulator based processors and zero-address instructions are used in stack-based CPU organizations.

Suppose we have to evaluate the arithmetic statement

$$X = (A + B) * C$$

using zero, one, two or three address instructions. For this, LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations of addition and multiplication respectively. Assume that the respective operands are in memory addresses A, B and C and the result must be stored in the memory at the address X.

Using three-address instructions, the program code in assembly language is as:

```
ADD R1, A, B    ;    R1 ← M[A] + M[B]
MULT X, C, R1   ;    X ← M[C] + R1
```

Using two-address instructions, the program code in assembly language is as:

```
LOAD R1, A      ;    R1 ← M[A]
ADD R1, B       ;    R1 ← R1 + M[B]
LOAD R2, C      ;    R2 ← M[C]
MULT R1, R2     ;    R1 ← R1 * R2
STORE X, R1     ;    X ← R1
```

Using one-address instructions, the program code in assembly language is as:

```
LOAD A      ;    AC ← M[A]
ADD B       ;    AC ← AC + M[B]
STORE T     ;    T ← AC
LOAD C      ;    AC ← M[C]
MULT T      ;    AC ← AC * M[T]
STORE X     ;    X ← AC
```

Using zero-address instructions, the program code in assembly language is as:

```
PUSH A      ;    TOS ← A          [TOS means top of the stack]
PUSH B      ;    TOS ← B
ADD         ;    TOS ← (A + B)
PUSH C      ;    TOS ← C
MULT        ;    TOS ← ((A + B) * C)
POP X       ;    X ← TOS
```

In base index addressing mode, the content of the base register (BR) is added to the address part of the instruction to obtain the effective address. This mode is similar to the indexed addressing mode, but the exception is in the way they are used. A base register holds the starting address of a memory array of operands and the address part of the instruction gives a displacement or offset relative to this starting address. This mode is used for relocation of the programs in the memory.

For example, an operand array starts at memory address 1000 and thus the base register BR contains the value 1000. Now consider the load instruction

LDA 0002

The effective address of the operand is calculated as:

Effective address = 0002 + content of BR
= 1002.

4. What is interrupt? Differentiate between vectored and non-vectored interrupts. 1 + 4

Answer

Interrupt is a special signal to the CPU generated by an external device that causes the CPU to suspend the execution of one program and start the execution of another.

In a vectored interrupt, the source that interrupts supplies the branch information (starting address of ISR) to the CPU. This information is called the interrupt vector, which is not any fixed memory location. The processor identifies individual devices even if they share a single interrupt-request line. So the set-up time is very less.

In a non-vectored interrupt, the branch address (starting address of ISR) is assigned to a fixed location in memory. Since the identities of requesting devices are not known initially, the set-up time is quite large.

5. Compare and contrast RISC and CISC architecture in brief.

Answer

CISC	*RISC*
1. A large number of instruction types used—typically from 100 to 250 instructions.	1. Relatively few number of instruction types—typically less than100 instructions.
2. A large number of addressing modes used—typically from 5 to 15 different modes.	2. Relatively few addressing modes—typically less than or equal to 5.
3. Variable-length instruction formats.	3. Fixed-length, easily decoded instruction formats.
4. Small number of general-purpose registers (GPRs)—typically 8-24 GPRs.	4. Large number of general-purpose registers (GPRs)—typically 32-192 GPRs.
5. Clock per instruction (CPI) lies between 2 and 15.	5. Clock per instruction (CPI) lies between 1 and 2.
6. Mostly micro-programmed control units.	6. Mostly hardwired control units.
7. Most instructions manipulate operands in memory.	7. All operations are executed within registers of the CPU.

6. What are the advantages of micro programming control over hardwired control ? Explain the role of an operating system in brief. 3 + 2

Answer

The main advantage of microprogramming is it provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions). With microprogramming, many additions and changes are made by simply changing the microprogram in the control memory (ROM). Whereas, a small change in the hardwired approach may lead to redesigning the entire system.

The main roles of operating system:

1. Managing the user's programs
2. Managing the memories of computer
3. Managing the I/O operations
4. Controlling the security of computer

Group-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. (a) Describe the major components of a digital computer with a suitable block diagram.
   (b) What are von Neumann concept and its bottlenecks?
   (c) Explain and draw a binary decrement unit.
   (d) Represent the decimal value –7.5 in IEEE-754 single precision floating point formats.

5 + 4 + 3 + 3

Answer

(a)

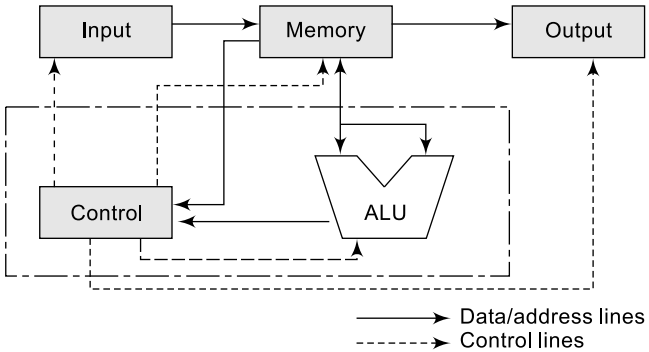


Figure 1 *Block diagram of a computer*

The major units of a computer are described next:

(i) **Arithmetic and Logic Unit (ALU):** It is the main processing unit which performs arithmetic and other data processing tasks as specified by the control unit. The ALU and control unit are the main constituent parts of the Central Processing Unit (CPU). Another component of the CPU is register unit—collection of different registers, used to hold the data or instruction temporarily.

(ii) **Control Unit:** This is the unit that supervises the flow of information between various units. The control unit retrieves the instructions using registers one by one from the program, which is stored in the memory. The instructions are interpreted (or decoded) by the control unit itself and then the decoded instructions are sent to the ALU for processing.

(iii) **Memory:** The memory unit stores programs as well as data. Generally, three types of memories are used: secondary, main and cache memories.

(iv) **Input Unit:** This unit transfers the information as provided by the users into memory. Examples include keyboard, mouse, scanner, etc.

(v) **Output Unit:** The output units receive the result of the computation and display it to the monitor or the user gets the printed results by means of a printer.

(b) The concept is known as *stored-program* concept and has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

Each instruction contains only one memory address and has the format:

OPCODE ADDRESS

The 8-bit op-code specifies the operation to be performed by the CPU and 12-bit address specifies the operand's memory address. Thus length of each instruction is 20-bit.

Von-Neumann bottleneck:

One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck.

(c) The binary decrementer unit performs the decrement micro-operation. The decrement micro-operation subtracts value one from the number stored in a register. For example, if a 4-bit register has a binary value 1001, it will be 1000 after the decrement operation. The subtraction can easily be implemented using combinational circuit half-subtractors or sequential circuit binary down counter. The decrement micro-operation can be realized with combinational circuit full adders. The subtraction of two binary numbers can be performed by taking the 2's complement of the subtrahend and then adding it to the minuend. The diagram of a 4-bit combinational decrementer circuit has been implemented using full adders, shown in the figure below.

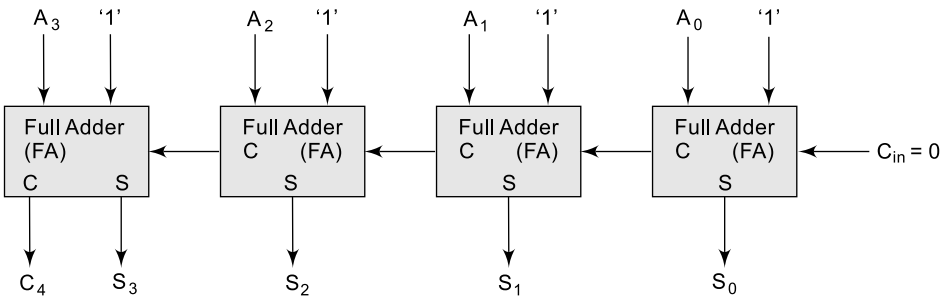


Figure 2 *4-bit Decrementer Circuit*

Here, we are adding a bit 1 as one of the inputs to each full adder. This means that binary number (1111) is added with the operand number A. The binary number (1111) means –1 in decimal, since the negative number is represented in computers using signed 2's complement method. That means, we are adding –1 with the operand value stored in register A.

(d) The decimal number $-7.5 = -111.1$ in binary $= -1.111 \times 2^2$

The 23-bit mantissa $M = 0.111000\ 000000\ 000000\ 00000$

The biased exponent $E' = E + 127 = 2 + 127 = 129 = 1000\ 0001$

Since the number is negative, the sign bit S = 1

Therefore, the IEEE single-precision (32-bit) representation is:

1	1000 0001	111000 000000 000000 00000

8. (a) Compare parallel adder with serial adder.

(b) With a suitable block diagram discuss the construction and working principles of an 8-bit carry- look-ahead adder.

(c) What are the advantages of CLA over ripple carry adder?

(d) Explain the importance of a common bus system in a computer. 4 + 5 + 4 + 2

Answer

(a)

Parallel Adder	*Serial Adder*
1. This adder is a combinational circuit, which adds all bits of two numbers in one clock cycle.	1. This adder is a sequential circuit, which performs the addition of two binary numbers serially bit by bit starting with lsb.
2. This adder, being a combinational circuit, is faster than serial adder. In one clock period all bits of two numbers are added.	2. The serial adder is very slow since it takes *n* clock cycles for addition of *n*-bit numbers.
3. The hardware cost is more than that of serial adder. Because, number of adder blocks needed is equal to the number of bits in operands.	3. The serial adder circuit is small and hence, it is very inexpensive irrespective of the number of bits to be added.

(b) A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1} = A_iB_i + (A_i + B_i)C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in the figure on the right. The sum bit S_i is $= A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

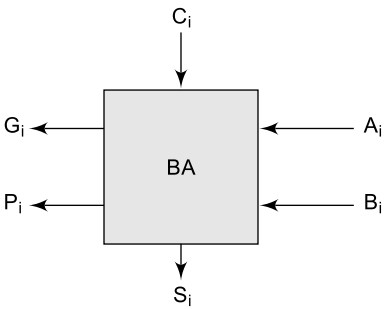


Figure 3 *Basic Adder (BA)*

Now, for an 8-bit CLA, eight carries $C_1, C_2, \ldots, C_8$ are to be generated. Using equation number (1); $C_1, C_2, \ldots, C_8$ can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$\ldots\ldots\ldots\ldots$$
$$C_8 = G_7 + P_7C_7$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$
$$C_2 = G_1 + P_1C_1$$
$$= G_1 + P_1(G_0 + P_0C_0)$$
$$= G_1 + P_1G_0 + P_1P_0C_0 \quad (3)$$
$$C_3 = G_2 + P_2C_2$$
$$= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \quad (4)$$

Similarly, C_4, C_5, C_6, C_7 and C_8 can be expanded to remove the recursion.

The equations (2), (3), (4) and others, if derived, suggest that C_1, C_2, …, C_8 can be generated directly from C_0. In other words, these eight carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. An 8-bit carry look-ahead adder (CLA) is shown in the figure below.

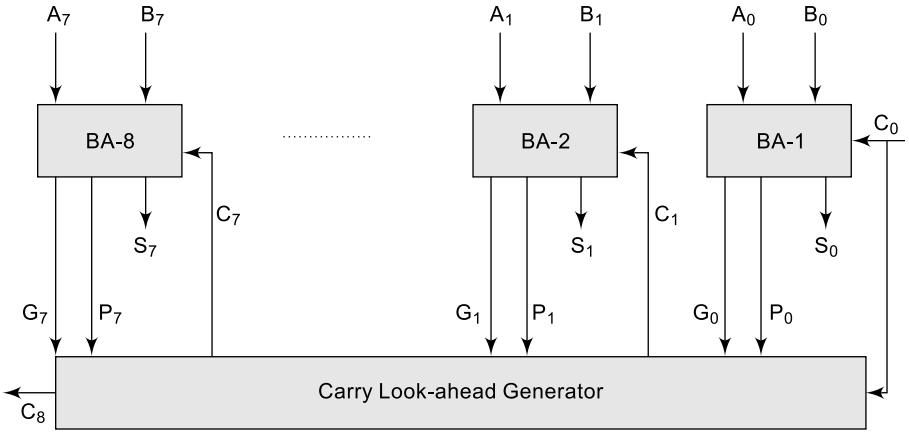


Figure 4 *8-bit Carry Look-ahead Adder (CLA)*

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits. Whereas, the maximum propagation delay for CPA depends on size of inputs and for n-bit CPA it is $\Delta \times n$, where Δ is the time delay for each full adder stage and n is the number of bits in each operand.

(c) The advantages of carry-look-ahead adder over ripple-carry adders:

- Generally, the carry look-ahead adder (CLA) is faster than the ripple carry adder (RCA). Because, the maximum delay of the CLA is $6 \times \delta$, where δ is the average gate delay and this holds good for any size numbers. However, the maximum propagation delay for n-bit RCA is $\delta \times n$, where n is the number of bits in each operand.
- The RCA becomes slow once the sizes of operands are increased, which is not true for CLA. The speed of CLA remains same irrespective of sizes of operands.

(d) Many registers are provided in the CPU of a computer for fast execution. Therefore, several paths must be provided to transfer information from one register to another. If a separate communication line is used between each register pair in the system, the number of lines will be excessive and thus cost of communication will be huge. Thus it is economical to have a common bus system for transferring information between registers in a multiple-register configuration.

9. (a) Explain Booth's multiplication algorithm with a suitable flowchart.

(b) Using Booth's algorithm multiply (–12) and (+6).

(c) What do you mean by "guard bit"? 8 + 5 + 2

Answer

(a)

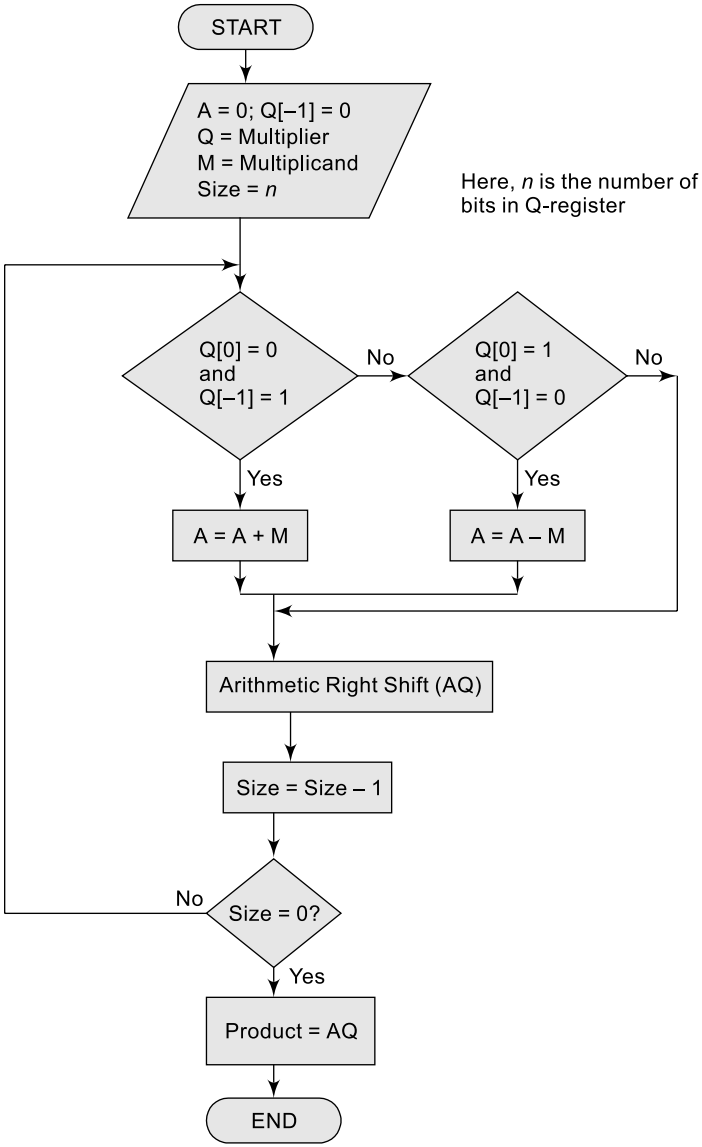


Figure 5 *Booth's multiplication algorithm*

The algorithm inspects two lower-order multiplier bits at time to take the next step of action. The algorithm is described by the flowchart shown above (Fig. 5). A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contain the product. Ignore the right end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier.

(b) Multiplication of numbers $(+6)_{10}$ and $(-12)_{10}$:

Multiplicand, M = +6 = 00110 and multiplier, Q = –12 = 10100.

	M	A	Q	Size
Initial Configuration	00110	00000	10100 0	5
Step-1				
As Q[0] = 0 and Q[–1] = 0				
ARS(AQ)	00110	00000	01010 0	4
Step-2				
As Q[0] = 0 and Q[–1] = 0				
ARS(AQ)	00110	00000	00101 0	3
Step-3				
As Q[0] = 1 and Q[–1] = 0				
A = A – M	00110	11010	00101 0	—
ARS(AQ)	00110	11101	00010 1	2
Step-4				
As Q[0] = 0 and Q[–1] = 1				
A = A + M	00110	00011	00010 1	—
ARS(AQ)	00110	00001	10001 0	1
Step-5				
As Q[0] = 1 and Q[–1] = 0				
A = A – M	00110	11011	10001 0	—
ARS(AQ)	00110	11101	11000 1	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 1110111000, which shows that the product is a negative number. To get the result in familiar form, take the 2's complement of the magnitude of the number and the result is –72 in decimal.

(c) When the mantissa is shifted right, some bits at the right most position (least significant position) are lost. In order to obtain maximum accuracy of the final result; one or more extra bits known as *guard bits*, are included in the intermediate steps. These bits temporarily contain the recently shifted out bits from the right most side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are not stored. However, based on the guard bits, the value of the mantissa can be made more precise by the rounding technique.

10. (a) Explain instruction cycle, machine cycle and T-states with suitable example.

(b) What are the advantages of relative addressing mode over direct address mode?

(c) Draw and explain the timing diagram for memory write operation.

(d) Evaluate the arithmetic statement X = (A * B)/(C + D) in one-, two- and three- address machine. 5 + 4 + 3 + 3

Answer

(a) **Instruction cycle:** The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

Machine cycle: A machine cycle consists of necessary steps carried out to perform the memory access operation. Each of the basic operations such as fetch or read or write operation constitutes a machine cycle. An instruction cycle consists of several machine cycles.

T-states: One clock cycle of the system clock is referred to as T-state.

(b) In relative addressing mode, the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. The instruction specifies the memory address of the operand as the relative position of the current instruction address. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address.

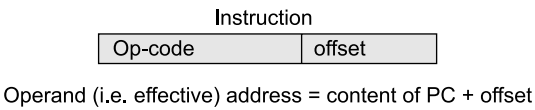


Figure 6 *Relative addressing mode*

In direct addressing mode, the instruction contains the memory address of the operand explicitly. Thus, the address part of the instruction is the effective address.

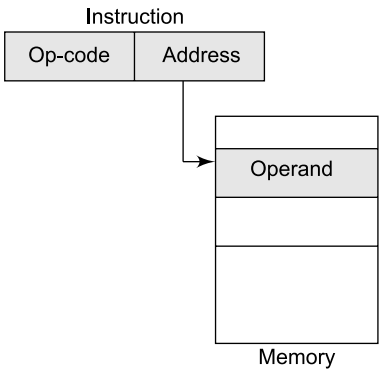


Figure 7 *Direct addressing mode*

The advantages of relative addressing mode over direct addressing mode:

- In relative addressing mode, smaller number of bits are used as the address of the operands compared to the direct addressing mode.

- Since the size of relative addressed mode instructions is shorter than that of direct mode instructions, the relative addressed mode instructions occupy lesser memory space, which decreases the memory requirement.
- Due to the smaller size for relative addressing mode instructions, either data bus width is small or instruction fetch takes less time.

(c) The timing diagram for memory write is given as below.

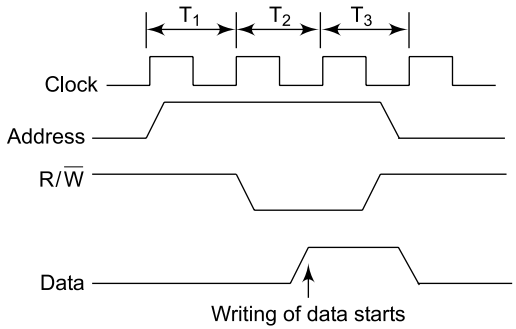


Figure 8 *Timing diagram of memory write*

To write a memory word, the address is specified on the address bus to select the word among many available words at first clock period T_1. At second clock period T_2, memory write signal is activated and after seeing the write signal activated, memory stores data in it from the data bus. The total operation needs three clock periods.

(d) We have to evaluate the arithmetic statement

$$X = (A * B) / (C + D)$$

using one, two or three address instructions. For this, LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD, MULT and DIV are used for the arithmetic operations addition, multiplication and division respectively. Assume that the respective operands are in memory addresses A, B, C and D and the result must be stored in the memory at address X.

Using three-address instructions, the program code in assembly language is as:

```
MULT R1, A, B    ;    R1 ← M[A] * M[B]
ADD R2, C, D     ;    R2 ← M[C] + M[D]
DIV X, R1, R2    ;    X ← R1 / R2
```

Using two-address instructions, the program code in assembly language is as:

```
LOAD R1, A       ;    R1 ← M[A]
MULT R1, B       ;    R1 ← R1 * M[B]
LOAD R2, C       ;    R2 ← M[C]
ADD R2, D        ;    R2 ← R2 + M[D]
DIV R1, R2       ;    R1 ← R1 / R2
STORE X, R1      ;    X ← R1
```

Using one-address instructions, the program code in assembly language is as:

LOAD C	;	AC ← M[C]
ADD D	;	AC ← AC + M[D]
STORE T	;	T ← AC
LOAD A	;	AC ← M[A]
MULT B	;	AC ← AC * M[B]
DIV T	;	AC ← AC/M[T]
STORE X	;	X ← AC

11. Write short notes on any three of the following: 3×5
    (a) IAS computer
    (b) Concept of handshaking in I/O operation
    (c) Static and dynamic memory
    (d) DMA controller
    (e) Classify MRI and non-MRI instructions.

Answer

(a) **IAS Computer:**

In 1946, Von Neumann and his colleagues began the design of a new *stored-program* computer, now referred to as the IAS computer, at the Institute for Advanced Studies, Princeton. Nearly, all modern computers still use this stored-program concept. This concept has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

This machine employed a random-access Cathode-Ray-Tube (CRT) main memory, which permitted an entire word to be accessed in one operation. Parallel binary circuits were employed. Each instruction contained only one memory address and had the format:

OPCODE ADDRESS

The central processing unit (CPU) contained several high-speed (vacuum-tube) registers used as implicit storage locations for operands and results. Its input-output facilities were limited. It can be considered as the prototype of all subsequent general-purpose computers.

Instruction Format: The basic unit of information, i.e. the amount of information that can be transferred between the main memory and CPU in one step is a 40-bit word. The memory has a capacity of $2^{12} = 4096$ words. A word stored in the memory can represent either instruction or data.

Data: The basic data item is a binary number having the format shown in Fig. 9. Leftmost bit represents the sign of number (0 for positive and 1 for negative) while the remaining 39 bits indicate the number's size. The numbers are represented as fixed-point numbers.

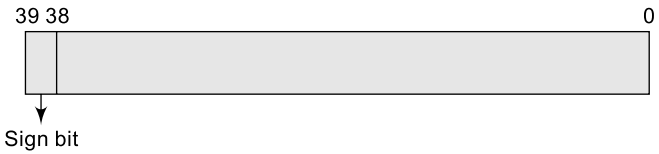


Figure 9 *Number word*

Instruction: IAS instructions are 20 bits long, so that two instructions can be stored in each 40-bit memory location. An instruction consists of two parts, as shown in Figure below: an 8-bit op-code (operation code), which defines the operation to be performed (add, subtract, etc.) and a 12-bit address part, which can identify any of 2^{12} memory locations that may be used to store an operand of the instruction.

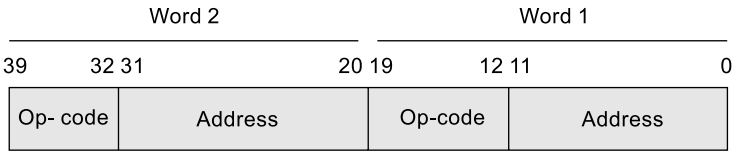


Figure 10 *Instruction word*

(b) Concept of handshaking in I/O operation:

The *disadvantage* of the strobe method is that the source unit that initiates the transfer cannot know whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer cannot know whether the source unit has actually placed the data on the bus.

To overcome this problem of strobe technique, another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as *handshaking* mode of transfer.

Figure 11 below shows the data transfer method when initiated by the source. The two handshaking lines are *data valid*, which is generated by the source unit, and *data accepted*, generated by the destination unit. The source first places data and after some delay, issues data valid signal. On sensing data valid signal, the destination receives data and then issues acknowledgement signal data accepted to indicate the acceptance of data. On sensing data accepted signal, the source removes data and data valid signal. On sensing removal of data valid signal, the destination removes the data accepted signal.

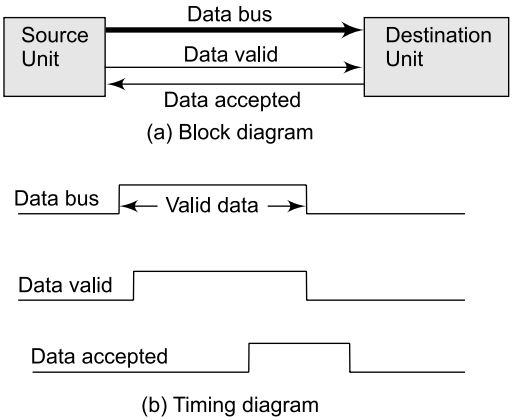


Figure 11 *Source-initiated transfer using handshaking*

Figure below illustrates destination initiated handshaking technique. The destination first sends the *data request* signal. On sensing this signal, the source places data and also issues the *data valid* signal. On sensing data valid signal, the destination acquires data and then removes the data request signal. On sensing this, the source removes both the data and data valid signal.

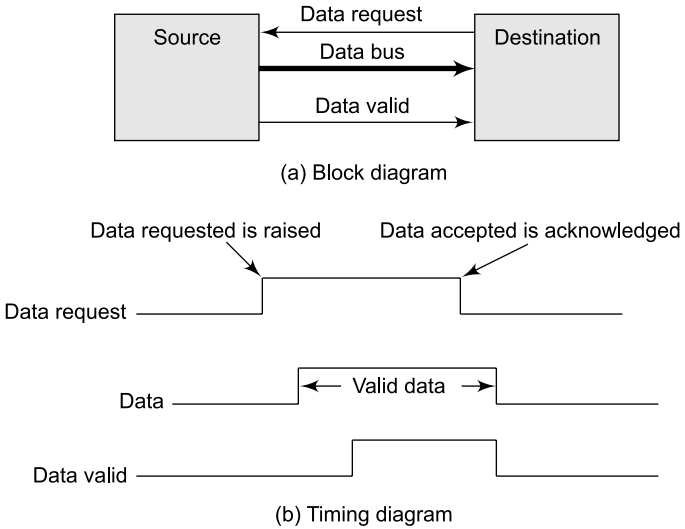


Figure 12 *Destination initiated handshaking technique*

The *advantage* of handshaking scheme is that it provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.

The *disadvantages* of handshaking scheme are:

1. A slow speed destination unit can hold up the bus whenever it gets a chance to communicate.
2. If one of the two communicating devices is faulty, the initiated data transfer cannot be completed.

Examples of asynchronous transfer:

1. The centronics interface follows handshaking scheme.
2. Most microprocessors such as Motorola 88010 and Intel 80286 follow this bus transfer mechanism.

(c) Static and dynamic memory: The central storage unit in a computer system is the main memory which is directly accessible by the CPU. It is a relatively large and fairly fast external memory used to store programs and data during the computer operation. Most of the main memory in a general-purpose computer is made up of RAM (Random Access Memory) integrated circuit chips, which are volatile (i.e. if power goes off, the stored information is lost) in nature. But a small part of the main memory is also constructed with ROM (Read Only Memory) chips, which are non-volatile. Originally, RAM was used to refer to a random-access memory, but now it is used to mean a read-write memory (RWM) to distinguish it from a read-only memory, although ROM's access mechanism is also random.

RAM is used to store the most of the programs and data that are modifiable. Integrated RAM chips are available in two forms: one is *static RAM* (*SRAM*) and another is *dynamic RAM* (*DRAM*). The SRAM memories consist of circuits capable of retaining the stored information as long as power is applied. That means this type of memory requires constant power. SRAM memories are used to build cache memory. On the other hand, DRAM stores the binary information in the form of electric charges that applied to capacitors. The stored information on the capacitors tends to be lost over a period of time and thus the capacitors must be periodically recharged to retain their state. The main memory is generally made up of DRAM chips.

Comparison of SRAM and DRAM

1. The SRAM has lower access time, which means it is faster compared to the DRAM.
2. The SRAM requires constant power supply, which means this type of memory consumes more power; whereas, the DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
3. Due to the relatively small internal circuitry in the one-bit memory cell of DRAMs, the large storage capacity in a single DRAM memory chip is available compared to the same physical size SRAM memory chip. In other words, DRAM has high packaging density compared to the SRAM.
4. SRAM is costlier than DRAM.

(d) DMA controller: To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA)*.

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure 13 shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.

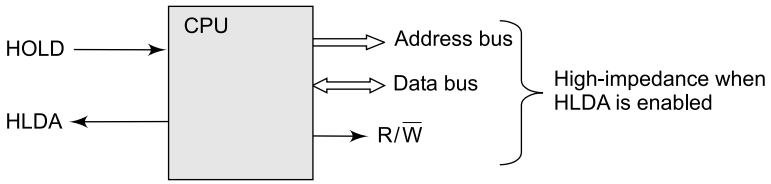


Figure 13 *CPU bus signals for DMA transfer*

To communicate with the CPU and I/O device the DMA controller needs the usual circuits of an interface. In addition to that, it needs an address register, a word-count register, a status register and a set of address lines. Three registers are selected by the controller's register select (RS) line. The address register and address lines are used for direct communication with the memory. The address register is used to store the starting address of the data block to be transferred. The word count register contains the number of words that must be transferred. This register is decremented by one after each word transfer and internally tested for zero after each transfer. Between the device and memory under control of the DMA, the data transfer can be done directly. The status register contains information such as completion of DMA transfer. All registers in the DMA controller appear to the CPU as I/O interface registers. Thus, the CPU can read from or write into the DMA registers under program control via the data bus.

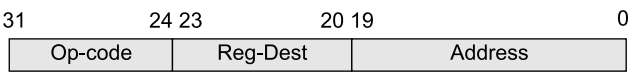
When executing the program for I/O transfer, the CPU first initializes the DMA controller. After that, the DMA controller starts and continues to transfer data between memory and peripheral unit until an entire block is transferred. The DMA controller is initialized by the CPU by sending the following information through the data bus:

1. The starting address of the memory blocks where data are available for read or where data are to be stored for write.
2. The number of words in the memory block (word count) to be read or written.
3. Read or write control to specify the mode of transfer.
4. A control to start the DMA transfer.

(e) Classify MRI and non-MRI instructions:

Memory reference instructions (MRIs) work directly between the registers and main memory. Memory reference instructions load values into and store values from the general registers.

We need to be very precise about our instruction format so that we can easily interpret it in hardware. Instructions are stored in one part of memory and data in another. For a memory unit with 1M words we need 20 bits to specify an address since 2^{20} = 1M. If we store each instruction code in one 32-bit memory word and 16 working registers, we have available 8 bits for the operation code to specify 256 instructions. The instruction format is mentioned below:



There are four basic instructions that reference the memory directly:

LOAD Reg, Address
STORE Reg, Address
JUMP Address
CALL Reg, Address

These all have the same instruction format as provided above.

2011
Computer Organization
(CS-303 (Old))

Time Allotted: 3 hours *Full Marks: 70*

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable.

Group-A
(Multiple-Choice Type Questions)

1. Choose the correct alternatives for the following: $10 \times 1 = 10$

(i) The principle of locality justifies the use of

(a) interrupt (b) polling

(c) DMA (d) cache memory

Answer

(d) cache memory

(ii) Instruction cycle is

(a) fetch-decode-execution (b) fetch-execution-decode

(c) decode-fetch-execution (d) none of these

Answer

(a) fetch-decode-execution

(iii) Subtractor can be implemented using

(a) adder (b) complementer (c) both (a) and (b) (d) none of these

Answer

(c) both (a) and (b)

(iv) How many RAM chips of size 256K $\times$ 1 bit are required to built 1 MB memory?

(a) 24 (b) 10 (c) 32 (d) 8

Answer

(c) 32

(v) Maximum n bit 2's complement number is

(a) 2^n (b) $2^n - 1$ (c) $2^{n-1} - 1$ (d) cannot be said

Answer

(c) $2^{n-1} - 1$

(vi) Micro instructions are kept in

(a) main memory (b) control memory (c) cache memory (d) none of these

Answer

(b) control memory

(vii) Physical memory broken down into groups of equal size is called

(a) page (b) tag (c) block (d) index

Answer

(c) block

(viii) Overflow occurs when

(a) data is out of range (b) data is within range

(c) both (a) and (b) (d) none of these

Answer

(a) data is out of range

(ix) The minimum number of operands with any instruction is

(a) 1 (b) 0 (c) 2 (d) 3

Answer

(b) 0

(x) The basic principle of the von Neumann computer is

(a) storing program and data in separate memory

(b) using pipeline concept

(c) storing both program and data in the same memory

(d) using a large number of register

Answer

(c) storing both program and data in the same memory

Group-B
(Short-Answer Type Questions)

Answer any *three* of the following. $3 \times 5 = 15$

2. (a) What is tri-state buffer? Construct a single line common bus system using tri-state buffer.

(b) What are guard bits? (1 + 2) + 2

Answer

(a) A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that no output is produced though there is an input signal and does not have logic significance. The gate is controlled by one separate control input C, If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low, the gate does not product any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown in Fig. 1.

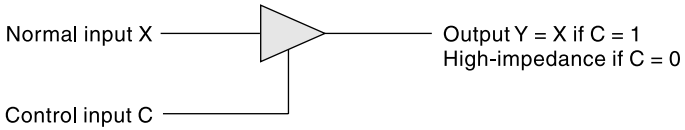


Figure 1 *Graphic symbol for a tri-state buffer gate*

A common bus system with tri-state buffers is described in Fig. 2. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

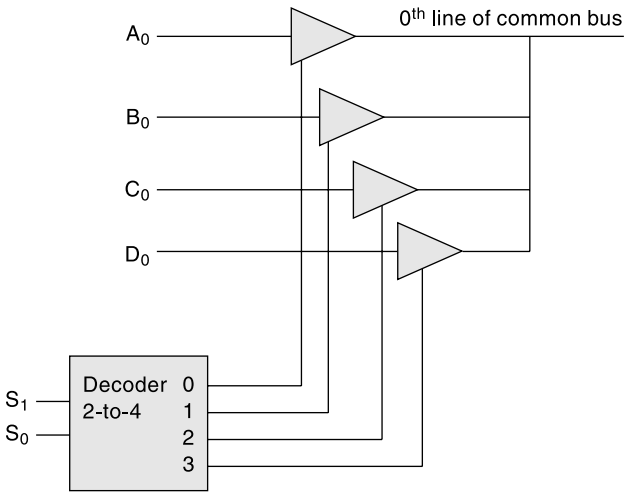


Figure 2 *A single line of a bus system with tri-state buffers*

(b) When the mantissa is shifted right, some bits at the rightmost position (least significant position) are lost. In order to obtain maximum accuracy of the final result; one or more extra bits known as *guard bits*, are included in the intermediate steps. These bits temporarily contain the recently shifted out bits from the rightmost side of the mantissa. When the number has to be finally stored in a register or in a memory as the result, the guard bits are not stored. However, based on the guard bits, the value of the mantissa can be made more precise by the rounding technique.

3. Describe stack based CPU.

Answer

Stack-based computer operates instructions, based on a data structure called stack. A stack is a list of data words with a Last-In, First-Out (LIFO) access method that is included in the

CPU of most computers. A portion of memory unit used to store operands in successive locations can be considered as a stack in computers. The register that holds the address for the top most operand in the stack is called a stack pointer (SP). The two operations performed on the operands stored in a stack are the *PUSH* and *POP*. From one end only, operands are pushed or popped. The PUSH operation results in inserting one operand at the top of stack and it decreases the stack pointer register. The POP operation results in deleting one operand from the top of stack and it increases the stack pointer register.

For example, Fig. 3 shows a stack of four data words in the memory. PUSH and POP instructions require an address field each. The PUSH instruction has the format:

PUSH <memory address>

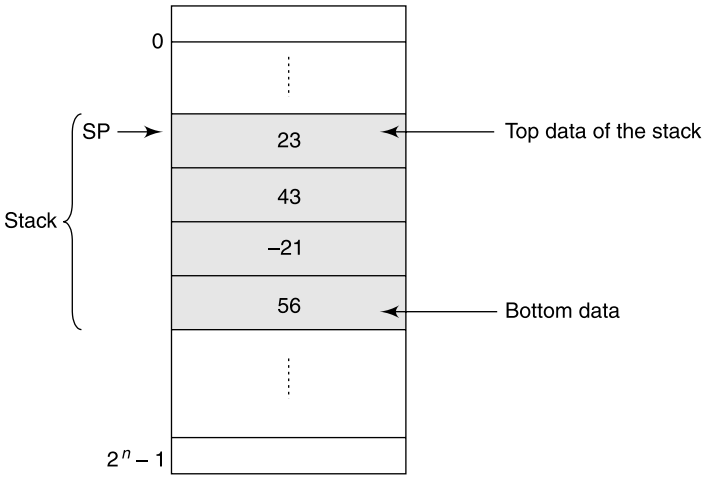


Figure 3 *A stack of words in memory*

The PUSH instruction inserts the data word at specified address to the top of the stack. The POP instruction has the format:

POP <memory address>

The POP instruction deletes the data word at the top of the stack to the specified address. The stack pointer is updated automatically in either case. The PUSH operation can be implemented as

SP ← SP – 1 : decrement the SP by 1
SP ← <memory address> : store the content of specified memory address into SP, i.e. at top of stack

The POP operation can be implemented as

<memory address> ← SP : transfer the content of SP (i.e. top most data) into specified memory location
SP ← SP + 1 : increment the SP by 1

Figure 4 shows the effects of these two operations on the stack in Figure 3.

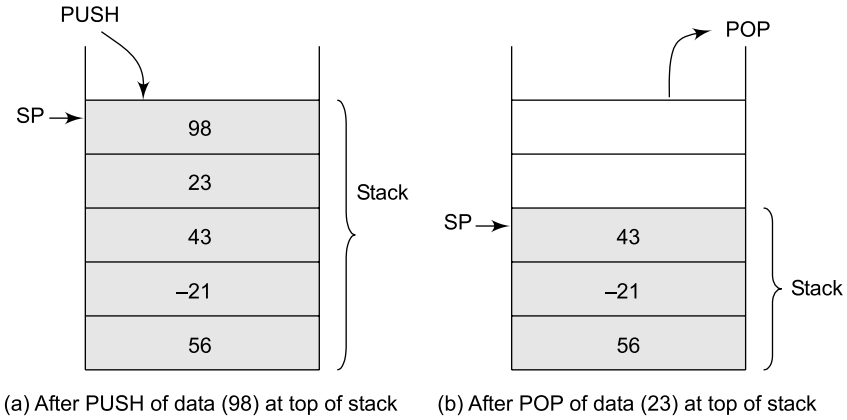


Figure 4 *Effects of stack operations on the stack in Fig. 3*

Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two operands that are on top of the stack. For example, the instruction

SUB

in a stack computer consists of an operation code only with no address field. This operation pops the two top data from the stack, subtracting the data, and pushing the result into the stack at the top. The organization of a stack-based machine is shown in Fig. 5.

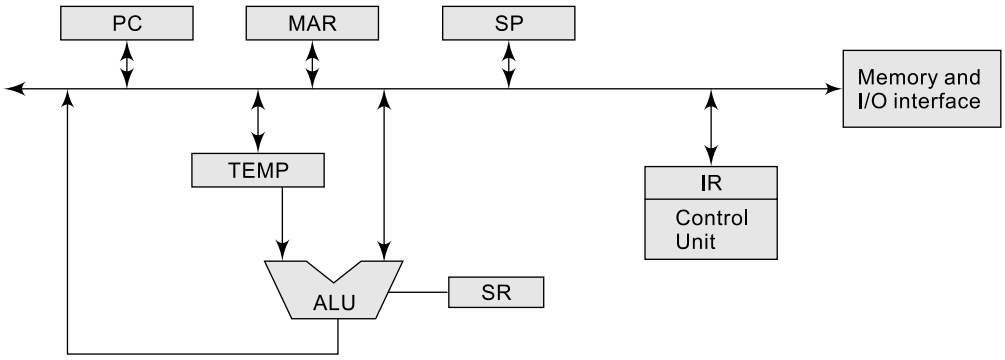


Figure 5 *Typical stack-based CPU organization*

PDP-11, Intel's 8085 and HP 3000 are some of the examples of stack-organized computers. The *advantages* of this organization:

1. Efficient computation of complex arithmetic expressions.
2. Execution of instructions is fast, because operand data are stored in consecutive memory locations.
3. Since instructions do not have address field, the length of instructions is short.

The *disadvantage* of this organization:

1. Program size lengthens.

4. (a) Write $+7_{10}$ in IEEE 32-bit format.
   (b) Convert IEEE 32-bit format 40400000_{16} in decimal value.
   (c) What is the role of an operating system? 2 + 2 + 1

Answer

(a) The decimal number $+7_{10} = +111$ in binary $= +1.11 \times 2^{+2}$
The 23-bit mantissa $M = 0.110000\ 000000\ 000000\ 00000$
The biased exponent $E' = E + 127 = +2 + 127 = 129 = 1000\ 0001$
Since the number is positive, the sign bit S = 0
Therefore, the IEEE single-precision (32-bit) representation is:

0	1000 0001	110000 000000 000000 00000

(b) The given number in IEEE 32-bit format is 40400000_{16}
$= 0100\ 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000_2$
Since the leading bit is 0, the number is positive.
Next higher order 8-bit indicates the biased exponent (E') and it is $(1000\ 00000)_2 = 128$
Therefore, the original exponent $E = E' - 127 = 128 - 127 = 1$
The leading bit in mantissa (after binary point) is 1, so the actual mantissa is $(1.1)_2$
Thus, the decimal number is $= +(1.1)_2 \times 2^{+1} = +(11)_2 = +3_{10}$

(c) An *operating system* (OS) is a set of programs and utilities which acts as the interface between user programs and computer hardware. The purpose of an operating system is to provide an environment in which a user may execute the programs. The following are the main functions of an operating system:
- User's program management
- Memory management
- Secondary storage management
- I/O management
- File management
- Protection
- Networking management
- Command interpretation

5. Evaluate the following arithmetic expression into three-address, two-address, one-address, zero-address instruction format : X = (A + B) * C

Answer

To evaluate this arithmetic expression, we use some op-codes as: LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and MULT are used for the arithmetic operations addition and multiplication respectively. Assume that the respective operands are in memory addresses A, B and C and the result must be stored in the memory at address X.

Using three-address instructions, the program code in assembly language is as:

```
ADD R1, A, B        ;R1 ← M[A] + M[B]
MULT X, C, R1       ;X ← M[C] + R1
```

Using two-address instructions, the program code in assembly language is as:

```
LOAD R1, A          ;R1 ← M[A]
ADD R1, B           ;R1 ← R1 + M[B]
LOAD R2, C          ;R2 ← M[C]
MULT R1, R2         ;R1 ← R1 * R2
STORE X, R1         ;X ← R1
```

Using one-address instructions, the program code in assembly language is as:

```
LOAD A              ;AC ← M[A]
ADD B               ;AC ← AC + M[B]
STORE T             ;T ← AC
LOAD C              ;AC ← M[C]
MULT T              ;AC ← AC * M[T]
STORE X             ;X ← AC
```

Using zero-address instructions, the program code in assembly language is as:

```
PUSH A              ;TOS ← A [TOS means top of the stack]
PUSH B              ;TOS ← B
ADD                 ;TOS ← (A + B)
PUSH C              ;TOS ← C
MULT                ;TOS ← ((A + B) * C)
POP X               ;X ← TOS
```

6. (a) Explain the difference between full associative and direct mapped cache mapping approaches.
   (b) What are "write through" and "write back" policies in cache? 3 + 2

Answer

(a) The fully associative cache memory uses the fastest and most flexible mapping method in which both address and data of the memory word are stored. This memory is expensive because of additional storage of addresses with data in the cache memory.

In the direct cache mapping, instead of storing total address information with data in cache, only part of address bits is stored along with data. Suppose the cache memory can hold 2^m words and main memory can hold 2^n words. The n-bit address generated by the CPU is divided into two fields: lower-order m bits for the index field and the remaining higher-order (n–m) bits for the tag field. The direct mapping cache organization uses the m-bit index to access the cache and higher-order (n–m) bits of tag are stored along side the data in cache. This is the simplest type of cache mapping, since only tag field is required to match. That's why it is one of the fastest caches. Also, it is less expensive cache relative to the associative cache.

(b) There are two policies in writing into cache memory: (i) write-through (ii) write-back.

Write-Through Policy: This is the simplest and most commonly used procedure to update the cache. In this technique, when the cache memory is updated, at the same time the main memory is also updated. Thus the main memory always contains the same data as the cache.

Write-Back Policy: In this method, during a write operation only the cache location is updated. When the update occurs, the location is marked by a flag called *modified* or *dirty bit*. When the word is replaced from cache, it is written into main memory if its flag bit is set.

Group-C
(Long-Answer Type Questions)

Answer any *three* of the following. $3 \times 15 = 45$

7. What is virtual memory? Why is it called virtual? What are the different address spaces? Explain with example how logical address is converted into physical address and also explain how page replacements take place. Explain the instruction cycle with a neat diagram. Explain the disadvantages of stored program computer. $2 + 2 + 4 + 5 + 2$

Answer

Virtual memory is a technique used in some large computer systems, which gives the programmer an illusion of having a large main memory, although which may not be the case. The size of virtual memory is equivalent to the size of secondary memory. Each address referenced by the CPU called the virtual (logical) address is mapped to a physical address in main memory. This mapping is done during run-time and is performed by a hardware device called *memory-management unit* (MMU) with the help of a memory map table, which is maintained by the operating system.

Virtual memory is not a physical memory, is actually a technique. That is why it is called virtual memory.

When a program needs to be executed, the CPU would generate addresses, called *logical* addresses. The corresponding addresses in the physical memory, as occupied by the executing program, are called *physical* addresses. The set of all logical addresses generated by the CPU or program is called *logical-address space* and the set of all physical addresses corresponding to these logical addresses is called *physical-address space*. The memory-management unit (MMU) maps each logical address to a physical address during program execution. Figure 6 illustrates this mapping method, which uses a special register called base register or relocation register. The content of the relocation register is added to every logical address generated by the user program at the beginning of execution. For example, if the relocation register holds an address value 2000, then a reference to the location 0 by the user is dynamically relocated to 2000 address. A reference to the address 150 is mapped to the address 2150.

When a program starts execution, one or more pages are brought to the main memory and the page table is responsible to indicate their positions. When the CPU needs a particular page for execution and that page is not in main (physical) memory (still in the secondary memory), this situation is called *page fault*. When the page fault occurs, the execution of the present program is suspended until the required page is brought into main memory from secondary memory. The required page replaces an existing page in the main memory, when it is brought into main memory. Thus, when a page fault occurs, a page replacement is needed to select one of the existing pages to make the room for the required page. There are several replacement algorithms such as *FIFO (First-in First-out*), *LRU (Least Recently Used*) and *optimal page replacement* algorithm available.

The *FIFO algorithm* is simplest and its criterion is "select a page for replacement that has been in main memory for longest period of time".

The *LRU algorithm* states that "select a page for replacement, if the page has not been used often in the past". The LRU algorithm is difficult to implement, because it requires a counter for each page to keep the information about the usage of page.

The *optimal algorithm* generally gives the lowest page faults of all algorithms and its criterion is "replace a page that will not be used for the longest period of time". This algorithm is also difficult to implement, because it requires future knowledge about page references.

An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a *reference string*.

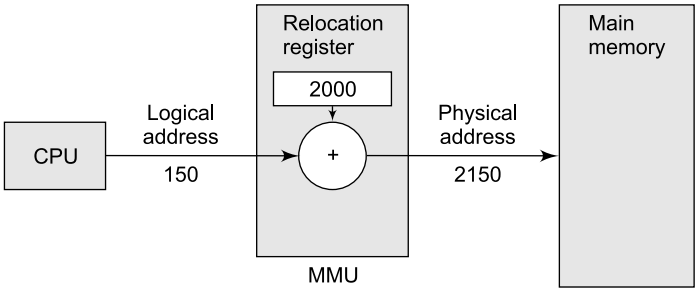


Figure 6 *A simple memory-management scheme*

The processing required for a single instruction is called *instruction cycle*. The control unit's task is to go through an instruction cycle (see Fig. 7) that can be divided into five major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand(s) from memory or register.
4. Execute the whole instruction.
5. Store the output result to the memory or register.

The step 1 is basically performed using a special register in the CPU called *program counter* (PC) that holds the address of the next instruction to be executed. If the current instruction is simple arithmetic/logic or load/store type the PC is automatically incremented. Otherwise, PC is loaded with the address dictated by the currently executing instruction. The decoding done in step 2 determines the operation to be performed and the addressing mode of the instruction for calculation of address of operands. After getting the information about the addresses of operands, the CPU fetches the operands in step 3 from memory or registers and stores them its registers. In step 4, the ALU of processor executes the instruction on the stored operands in registers. After the execution of instruction, in phase 5 the result is stored back in memory or register and returns to step 1 to fetch the next instruction in sequence. All these sub-operations are controlled and synchronized by the control unit.

Nearly all modern computers still use the stored-program concept. This concept has three main principles:

1. Program and data can be stored in the same memory.
2. The computer executes the program in sequence as directed by the instructions in the program.
3. A program can modify itself when the computer executes the program.

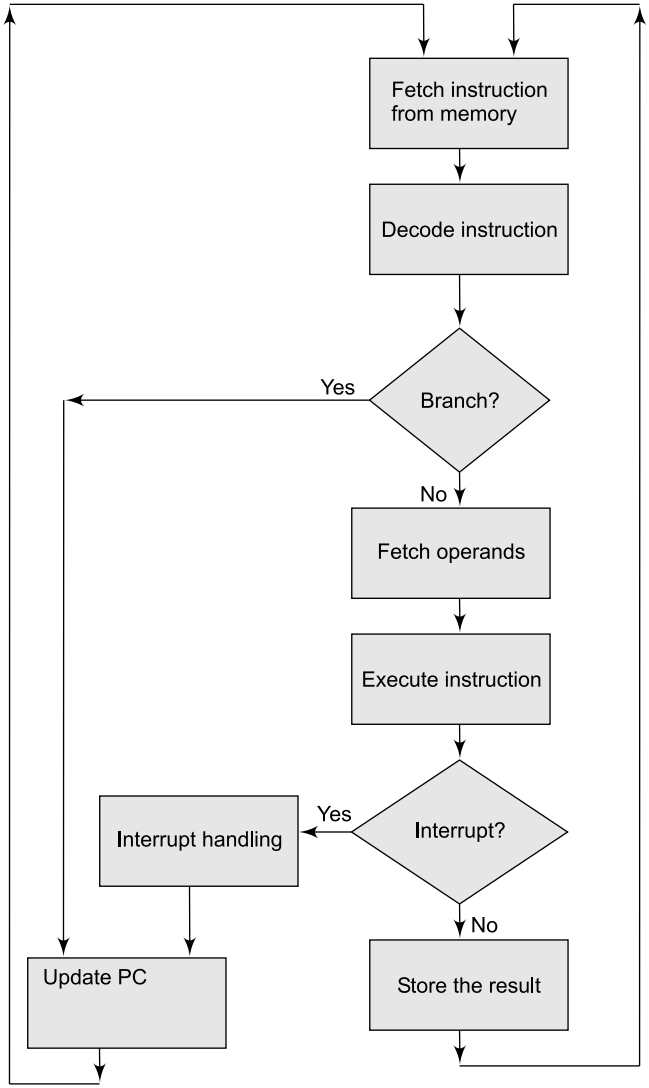


Figure 7 *Instruction cycle*

Disadvantages of the stored-program concept: One of the major factors contributing for a computer's performance is the time required to move instructions and data between the CPU and main memory. The CPU has to wait longer to obtain a data-word from the memory than from its registers, because the registers are very fast and are logically placed inside the processor (CPU). This CPU-memory speed disparity is referred to as Von-Neumann bottleneck. This performance problem is reduced by using a special type memory called *cache memory* between the CPU and main memory. The speed of cache memory is almost same as the CPU, for which there is almost no waiting time of the CPU for the required data-word to come. Another way to reduce the problem is by using special types of computers known as *Reduced Instruction Set Computers (RISC)*. This class of computers generally uses

a large number of registers, through which most of the instructions are executed. This computer usually limits access to main memory to a few load and store instructions. This architecture is designed to reduce the impact of the bottleneck by reducing the total number of the memory accesses made by the CPU and by increasing the number of register accesses.

8. Show the memory map with a CPU having 8 bit data bus and 16 bit address bus requiring four RAM chips of size 256 × 8 bit each and a ROM chip of 512 × 8 bit size. Explain the memory map. Among dynamic MOS cell and static MOS cell which one is used for the construction of cache memory and which one for main memory? What is destructive readout and non-destructive readout memory? 7 + 4 + 4

Answer

The addressing of memory can be designed by means of a table, known as *memory address map*, which specifies the memory address space assigned to each chip. The address map table for the memory connection to the CPU shown in Fig. 8 is constructed in Table 1. The CPU generates 16-bit address for memory read or write operation. The address lines 1 to 8 are connected to each memory and address line 9 is used in dual purposes. In case of a RAM selection out of four RAMs, the line no. 9 and line no. 10 are used through a 2-to-4 decoder. The line no. 9 is also connected to the ROM as address line along with lines 1 to 8 giving a total of 9 address lines in the ROM, since the ROM has 512 locations. The CPU address line number 11 is used for separation between RAM and ROM. The other 12 to 16 lines of CPU are unused and for simplicity we assume that they carry 0s as address signals. For ROM, 10^{th} line is unused and thus it can be assumed that this line carries signal 0.

Table 1 *Memory address map table for Fig. 8*

Chip selected	*Address space (in HEX)*	*Address bus*										
		11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*
RAM1	0400 – 04FF	1	0	0	x	x	x	x	x	x	x	x
RAM2	0500 – 05FF	1	0	1	x	x	x	x	x	x	x	x
RAM3	0600 – 06FF	1	1	0	x	x	x	x	x	x	x	x
RAM4	0700 – 07FF	1	1	1	x	x	x	x	x	x	x	x
ROM	0000 – 01FF	0	0	x	x	x	x	x	x	x	x	x

Dynamic MOS cell is used to construct main memory and static MOS cell is used to construct cache memory because static memory is faster than dynamic memory.

In some memories, the reading the memory word destroys the stored word, this fact is known as *destructive readout* and memory is known as *destructive readout memory*. In these memories, each read operation must be followed by a write operation that restores the memory's original state. Example includes dynamic RAM.

In some memories, the reading the memory word does not destroy the stored word, this fact is known as *non-destructive readout* and memory is known as *non-destructive readout memory*. Examples include static RAM and magnetic memory.

9. Explain with a neat circuit diagram of static MOS cell and dynamic MOS cell. Describe memory reading and writing process. What is daisy chaining ? Discuss the data transfer using the DMA. 2 + 7 + 2 + 4

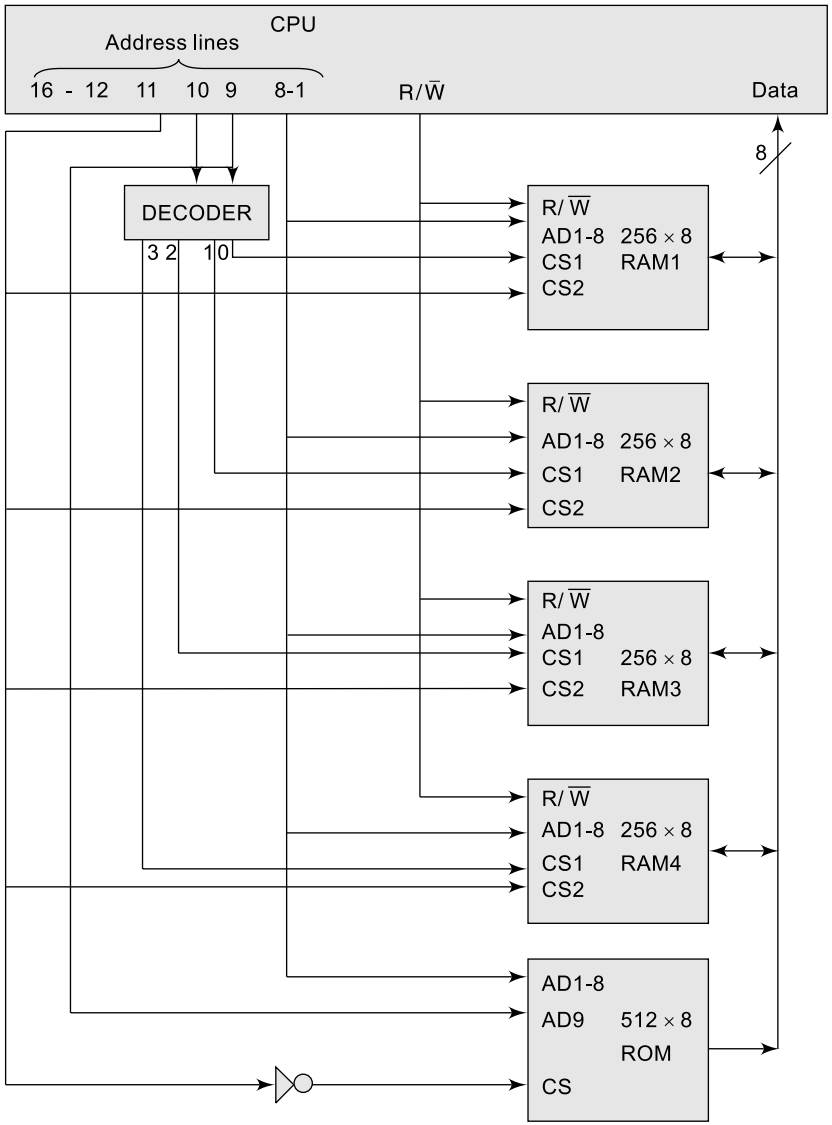


Figure 8 *Memory connection with 16-bit CPU*

Answer

Static MOS Cell

One static MOS cell is shown in Fig. 9. Four transistors (T_3, T_4, T_5 and T_6) are cross connected in such a way that they produce a stable state. In the state 1, the voltage at point A is maintained high and voltage at point at B is low by keeping transistors T_3 and T_6 on (i.e. closed), while T_4 and T_5 off (i.e. open). Similarly, in state 0, the voltage at A is low and at point B is high by keeping transistors T_3 and T_6 off, while T_4 and T_5 on. Both these states are stable as long as the power is applied on it. Thus, for state 1, if T_1 and T_2 are turned on (closed), bit lines b and b' will have high and low signals respectively.

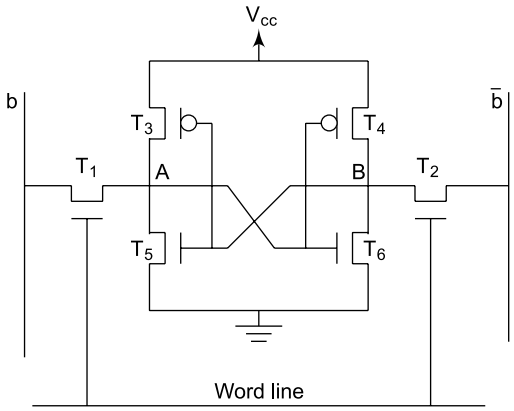


Figure 9 *A static MOS cell*

Read Operation: For the read operation, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T_1 and T_2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

Write Operation: Similarly, for the write operation, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored into the cell.

The main advantage of using static MOS memory is the low power consumption. Since, when the cell is being accessed the current flows in the cell only. Otherwise, T_1, T_2 and one transistor in each inverter are turned off, ensuring that there is no active path between V_{cc} and ground.

Dynamic MOS Cell:

Though static MOS memory is very fast, but it is expensive because of its each cell requires several transistors. Relatively less expensive RAM is dynamic MOS memory, due to the use of one transistor and one capacitor in each cell, as shown in Fig. 10, where C is the capacitor and T is the transistor. Information is stored in a DRAM cell in the form of a charge on a capacitor and this charge needs to be periodically recharged.

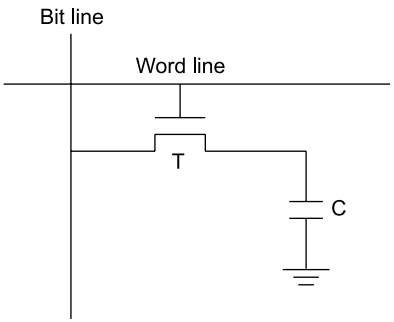


Figure 10 *A dynamic MOS cell*

For storing information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitor drops below some threshold value.

To implement interrupts, the CPU uses a signal, known as an *interrupt request (INTR)* signal to the interrupt controller hardware, which is connected to each I/O device that can issue an interrupt to it. Here, interrupt controller makes liaison with the CPU on behalf of I/O devices. Typically, interrupt controller is also assigned an *interrupt acknowledge (INTA)* line that the CPU uses to signal the controller that it has received and begun to process the interrupt request by employing an ISR (interrupt service routine). Devices are connected in daisy chain fashion, as shown in Fig. 11, to set up priority interrupt system.

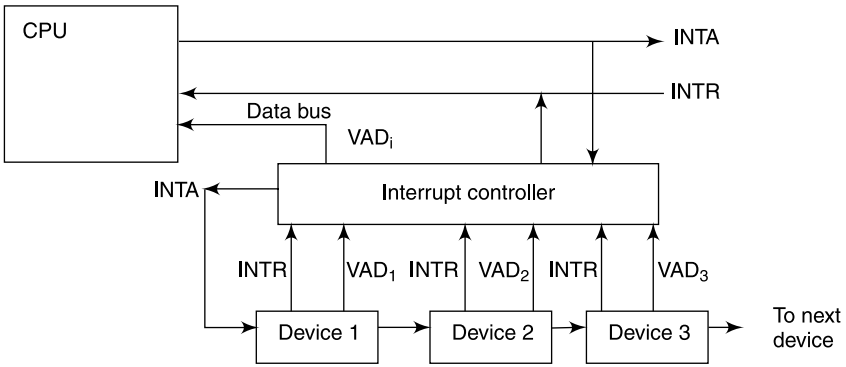


Figure 11 *Daisy chaining priority interrupt system*

The devices are placed in a chain-fashion with highest priority device in the first place (device 1), followed by lower priority devices. When one or more devices send interrupt signal through the interrupt controller to the CPU, the CPU then set interrupt acknowledge (INTA) to the controller, which in turns sends it to the highest priority device. If this device has generated the interrupt INTR, it will accept the INTA; otherwise it will pass the INTA signal to the next device until the INTA is accepted by one requestor device. When the INTA is accepted by a device, device puts its own interrupt vector address (VAD) to the data bus using interrupt controller.

To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access* (*DMA*).

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a *DMA controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (Hold Acknowledge). Figure 12 shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a

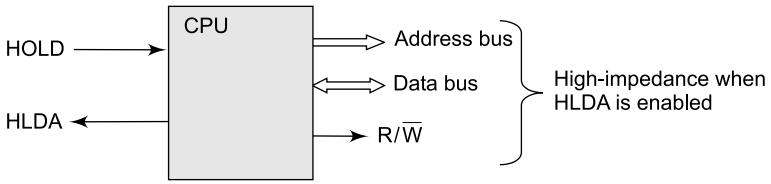


Figure 12 *CPU bus signals for DMA transfer*

high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.

10. Discuss various addressing modes with examples. Write a program that can evaluate the expression X = A – B + C – D in a single accumulator processor. Assume that the processor has load, store, sub and add instructions. What is the difference between zero-address and one-address instructions. Write a short note on overflow detection with examples. What are status flags ? 5 + 3 + 2 + 5

Answer

To obtain the addresses of operands, the address mode is needed. A processor can support various addressing modes in order to give flexibility to the users. The addressing mode gives the way addresses of operands is determined.

The various addressing modes are discussed next.

1. **Implied (or Inherent) Mode:** In this mode the operands are indicated implicitly by the instruction. The accumulator register is generally used to hold the operand and after the instruction execution the result is stored in the same register. For example,

   RAL; Rotates the content of the accumulator left through carry.
   CMA; Takes complement of the content of the accumulator.

2. **Immediate Mode:** In this mode the operand is mentioned explicitly in the instruction. In other words, an immediate-mode instruction contains an operand value rather than an address of it in the address field. To initialize registers to a constant value, this mode of instructions is useful. For example:

   MVI A, 06; Loads equivalent binary value of 06 to the accumulator.
   ADI 05; Adds the equivalent binary value of 05 to the content of AC.

3. **Stack addressing Mode:** Stack-organized computers use stack addressed instructions. In this addressing mode, all the operands for an instruction are taken from the top of the stack. The instruction does not have any operand field. For example, the instruction

   SUB

uses only one op-code (SUB) field, no address field. Both the operands are in the topmost two positions in the stack, in consecutive locations. When the SUB instruction is executed, two operands are popped out automatically from the stack one-by-one. After subtraction, the result is pushed onto the stack. Since no address field is used, the instruction is short.

4. **Register (Direct) Mode:** In this mode the processor registers hold the operands. In other words, the address field is now register field, which contains the operands required for the instruction. For example:
   ADD R1, R2; Adds contents of registers R1 and R2 and stores the result in R1.
5. **Register Indirect Mode:** In this mode the instruction specifies an address of CPU register that holds the address of the operand in memory. In other words, address field is a register which contains the memory address of operand.
6. **Auto-increment or Auto-decrement Mode:** This is similar to the register indirect mode except that after or before register's content is used to access memory it is incremented or decremented. It is necessary to increment or decrement the register after every access to an array of data in memory, if the address stored in the register refers to the array. This can be easily achieved by this mode.
7. **Direct (or Absolute) Address Mode:** In this mode the instruction contains the memory address of the operand explicitly. Thus, the address part of the instruction is the effective address. Examples of direct addressing are:

   ```
   STA 2500H    ; Stores the content of the accumulator in the memory location
                  2500H.
   LDA 2500H    ; Loads the accumulator with the content of the memory location
                  2500H.
   ```

8. **Indirect Address Mode:** In this mode the instruction gives a memory address in its address field which holds the address of the operand. Thus, the address field of the instruction gives the address where the effective address is stored in memory. The following example illustrates the indirect addressing mode:

   ```
   MOV (X), R1  ; Content of the location whose address is given in X is loaded
                  into register R1.
   ```

9. **Relative Address Mode or PC-relative Address Mode:** In this mode the effective address is obtained by adding the content of program counter (PC) register with address part of the instruction. The instruction specifies the memory address of operand as the relative position of the current instruction address. Generally, this mode is used to specify the branch address in the branch instruction, provided the branch address is nearer to the instruction address.
10. **Indexed Address Mode:** In this mode the effective address is determined by adding the content of index register (XR) with the address part of the instruction. This mode is useful in accessing operand array. The address part of the instruction gives the starting address of an operand array in memory. The index register is a special CPU register that contains an index value for the operand. The index value for operand is the distance between the starting address and the address of the operand. For example, an

operand array starts at memory address 1000 and assume that the index register XR contains the value 0002. Now consider load instruction

LDA 1000

The effective address of the operand is calculated as:

Effective address = 1000 + content of XR
= 1002.

11. **Base Register Address Mode:** This mode is used for relocation of the programs in the memory. *Relocation* is a technique of moving program or data segments from one part of memory to another part of memory. Relocation is an important feature of multiprogramming systems. In this mode the content of the base register (BR) is added to the address part of the instruction to obtain the effective address.

We have to evaluate the expression X = A – B + C – D in a single accumulator processor. Single accumulator based machine is a simple CPU in which the accumulator register (AC) is used implicitly for processing all instructions of a program and intermediate results are stored into this register. The instruction format in this computer uses one address field. For this the CPU is known as *one address machine*.

For this, LOAD symbolic op-code is used for transferring data to register from memory. STORE symbolic op-code is used for transferring data to memory from register. The symbolic op-codes ADD and SUB are used for the arithmetic operations of addition and subtraction respectively. Assume that the respective operands are in memory addresses A, B, C and D and the result must be stored in the memory at address X.

The single accumulator based computers use one-address instructions. Here, all instructions use an implied accumulator (AC) register. The program to evaluate X = (A – B) + (C – D) using one-address instructions is as follows:

```
LOAD C          ; AC ← M[C]
SUB D           ; AC ← AC – M[D]
STORE T         ; T ← AC
LOAD A          ; AC ← M[A]
SUB B           ; AC ← AC – M[B]
ADD T           ; AC ← AC + M[T]
STORE X         ; X ← AC
```

T is the temporary memory location required for storing the intermediate result.

One-address instructions are followed by the single accumulator based computers. Here, all instructions use an implied accumulator (AC) register. Zero-address instructions are used by stack-organized computers, which do not use any address field for the operation-type instructions. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions. Generally, one address instructions are executed quickly compared to the zero address instructions, because the accumulator register is used all the times for one address instructions execution while memory is generally accessed most of the time for zero address instructions execution.

An *overflow* is a problem in digital computers because the numbers are stored in registers, which are finite in length. If two numbers of *n* digits each are added and the

sum occupies $n + 1$ digits, an overflow will occur. This holds good irrespective of the numbers' type. Since a register of n-bit can not accommodate the result of $n + 1$ bits, an overflow results. If it occurs, a corresponding flip-flop in CPU is set, which is then verified by the user or program.

If one number is positive and the other is negative, after an addition overflow cannot occur, since addition of a positive number to a negative number produces a number that is always smaller than the larger of the two original numbers. However, an overflow may occur if the two numbers added are of same sign i.e., both are positive or both are negative. Let's consider following examples.

Carries:	01		Carries:	10
+69	0 1000101		–69	1 0111011
+78	0 1001110		–78	1 0110010
+147	1 0010011		–147	0 1101101

Observe that the 8-bit result that should have been positive (in first example) has a negative sign bit and the 8-bit result that should have been negative (in second example) has a positive sign bit. However, if the carry out from the sign bit position is treated as the sign of the result, the 9-bit answer thus obtained will be correct answer. Since the 9-bit answer cannot be accommodated with 8-bit register, we say that an overflow results.

To detect an overflow condition the carry into the sign bit position and the carry out from the sign bit position are examined. If these two carries are both 0s or both are 1s, there is no overflow. If these two carries are not equal (i.e., if one is 0 and other is 1), an overflow condition exists. This is illustrated in the examples where the two carries are explicitly shown. Using an XOR gate, whose two inputs are these carries, an overflow can be detected when the output of the gate is 1.

The processor uses one special register called *status register* (SR) to hold the latest program status. It holds 1-bit flags to indicate certain conditions that produced during arithmetic and logic operations. The bits are set or reset depending on the outcome of most recent arithmetic and logic operation. The register generally contains following four flags:

Carry (*C*)*:* It indicates whether there is any end-carry from the most significant bit position.

Zero (*Z*)*:* It indicates whether the result is zero or non-zero.

Sign (*S*): It indicates whether the result is positive or negative.

Overflow (*V*): It indicates whether the operation produces any overflow or not.

There may be other flags such as parity and auxiliary carry.

11. Write short notes on any *three* of the following: 3×5
    (a) Carry look ahead adder
    (b) Design of a 4-bit adder-subtractor circuit
    (c) Tri-state buffer
    (d) Booth's algorithm for multiplication
    (e) Cache memory.

Answer

(a) Carry look ahead adder: A Carry Look-ahead Adder (CLA) is a high-speed adder, which adds two numbers without waiting for the carries from the previous stages. In the CLA, carry-inputs of all stages are generated simultaneously, without using carries from the previous stages.

In the full adder, the carry output C_{i+1} is related to its carry input C_i as follows:

$$C_{i+1} = A_iB_i + (A_i + B_i)C_i$$

This result can be rewritten as:

$$C_{i+1} = G_i + P_iC_i \quad (1)$$

where $G_i = A_iB_i$ and $P_i = A_i + B_i$

The function G_i is called the *carry-generate* function, since a carry C_{i+1} is generated when both A_i and B_i are 1s. The function P_i is called as *carry-propagate* function, since if A_i or B_i is a 1, then the input carry C_i is propagated to the next stage. The basic adder (BA) for generating the sum S_i, carry propagate P_i and carry generate G_i bits, is shown in Fig. 13. The sum bit S_i is = $A_i \oplus B_i \oplus C_i$. For the implementation of one basic adder, two XOR gates, one AND gate and one OR gate are required.

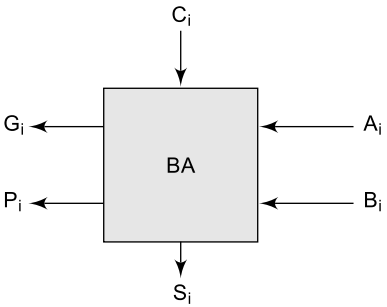


Figure 13 *Basic adder*

Now, we want to design a 4-bit CLA, for which four carries C_1, C_2, C_3 and C_4 are to be generated. Using equation number (1); C_1, C_2, C_3 and C_4 can be expressed as follows:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2$$
$$C_4 = G_3 + P_3C_3$$

These equations are recursive and the recursion can be removed as below.

$$C_1 = G_0 + P_0C_0 \quad (2)$$

$$\begin{aligned} C_2 &= G_1 + P_1C_1 \\ &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P_1P_0C_0 \end{aligned} \quad (3)$$

$$\begin{aligned} C_3 &= G_2 + P_2C_2 \\ &= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\ &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \end{aligned} \quad (4)$$

$$\begin{aligned} C_4 &= G_3 + P_3C_3 \\ &= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) \\ &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \end{aligned} \quad (5)$$

The equations (2), (3), (4) and (5) suggest that C_1, C_2, C_3 and C_4 can be generated directly from C_0. In other words, these four carries depend only on the initial carry C_0. For this reason, these equations are called *carry look-ahead* equations. A 4-bit carry look-ahead adder (CLA) is shown in Figure 14.

The maximum delay of the CLA is $6 \times \Delta$ (for G_i and P_i generation, delay = Δ, for C_i generation, delay = 2Δ and lastly another 3Δ for sum bit S_i) where Δ is the average gate delay. The same holds good for any number of bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

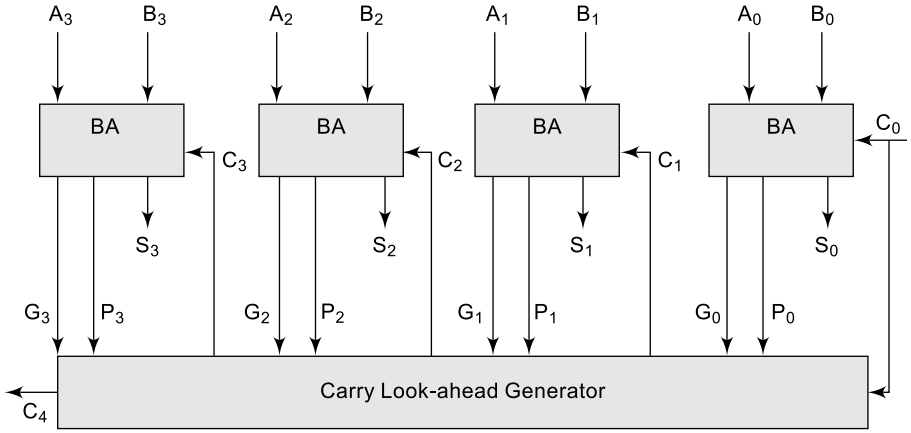


Figure 14 *4-bit Carry Look-ahead Adder (CLA)*

(b) Design of a 4-bit adder-subtractor circuit: Recall that the subtraction A – B is equivalent to A+ 2's complement of B (i.e. 1's complement of B +1). The addition and subtraction can be combined to a single circuit by using exclusive-OR (XOR) gate with each full adder. The circuit is shown in Fig. 15.

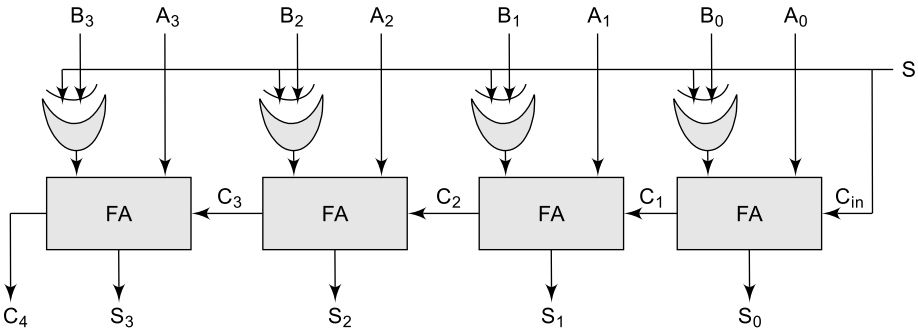


Figure 15 *4-bit Binary Adder-Subtractor*

The selection input S determines the operation. When S = 0, this circuit performs the addition operation and when S = 1, this circuit performs subtraction operation. The inputs to each XOR gate are S-input and B-input. When S = 0, we have $0 \oplus B = B$ (It can be verified from the truth table of XOR gate). This means that the direct B-value is given as input into a full adder (FA) and the carry-input into first full adder is 0. Thus, the circuit performs addition. When S = 1, we have $1 \oplus B = \overline{B}$ (It can be verified from the truth table of XOR gate) and carry-input is 1. This means that the circuit performs the addition of A with 2's

complement of B. For unsigned numbers, A – B if A ≥ B or the 2's complement of (B – A) if A < B. For signed numbers, the result is A – B provided that there is no overflow.

(c) Tri-state buffer: A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that no output is produced though there is an input signal and does not have logic significance. The gate is controlled by one separate control input C. If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low the gate does not produce any output irrespective of the input values. The graphic symbol of a tri-state buffer gate is shown in Figure 16.

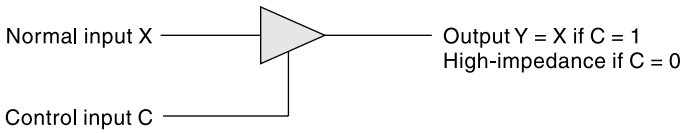


Figure 16 *Graphic symbol for a tri-state buffer gate*

A common bus system with tri-state buffers is described in Fig. 17. The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus. Note that only one buffer may be in the active state at any given time. Because the selection lines S_0, S_1 of the decoder activate one of its output lines at a time and the output lines of the decoder act as the control lines to the buffers. For example, if select combination S_1S_0 is equal to 00, then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A_0, 0^{th} bit of A register.

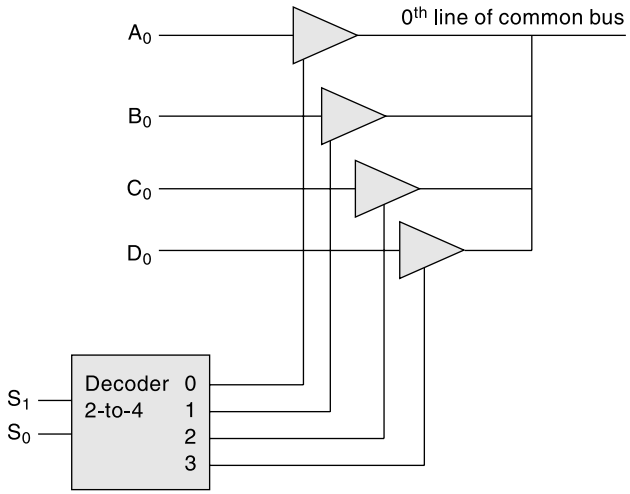


Figure 17 *A single line of a bus system with tri-state buffers*

(d) Booth's algorithm for multiplication: The algorithm inspects two lower-order multiplier bits at a time to take the next step of action. The algorithm is described by the flowchart in Figure 18. A flip-flop (a fictitious bit position) is used to the right of lsb of the multiplier and it is initialized to 0. Subsequently, it receives the lsb of the multiplier when the multiplier is shifted right.

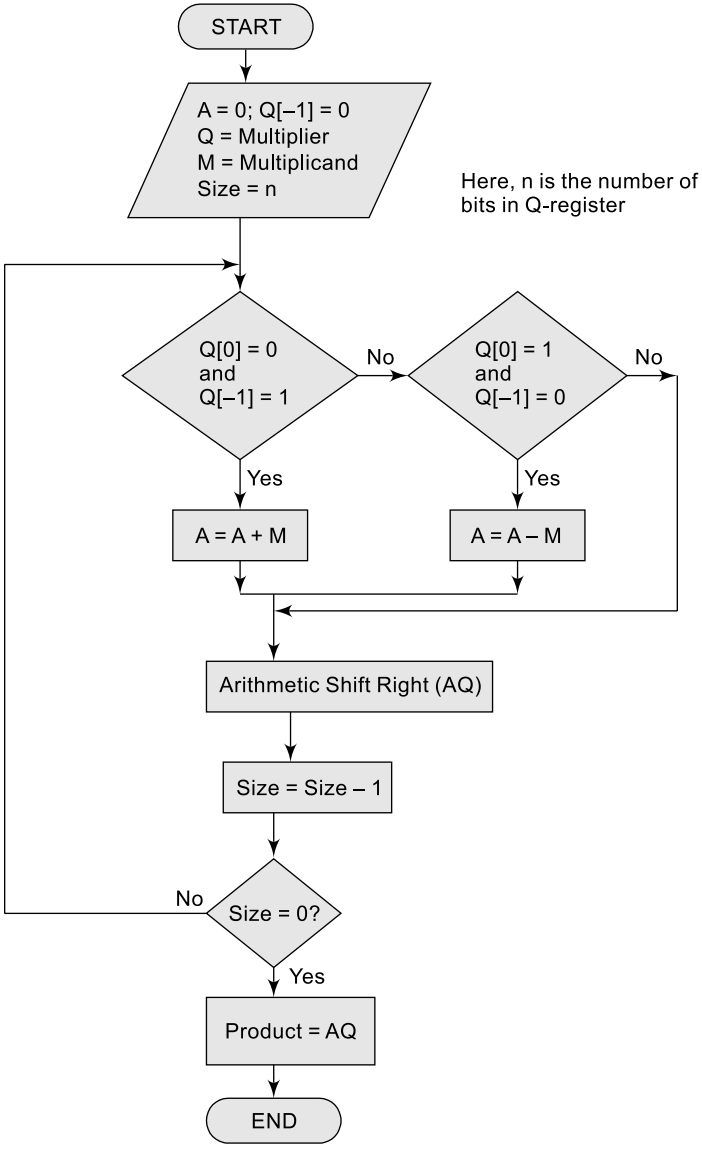


Figure 18 *Booth's multiplication algorithm*

Once all bits of the multiplier are inspected, the accumulator and multiplier registers together contains the product. Ignore the right end flip-flop used for holding an initial 0, as it is a fictitious bit and subsequent lsbs from multiplier. The circuit block diagram of the Booth's multiplication algorithm is shown Fig. 19.

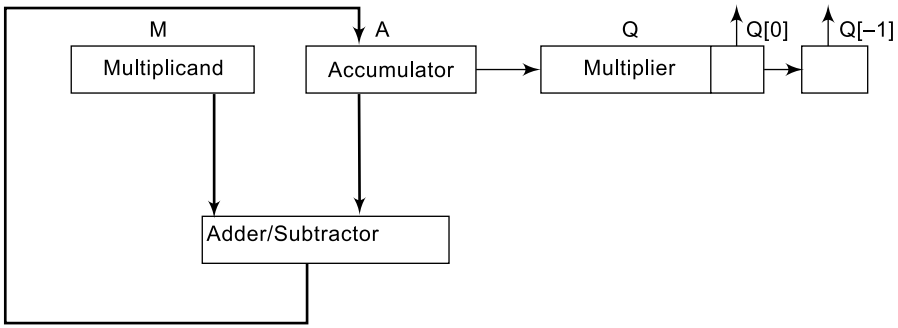


Figure 19 *Block diagram of Booth's multiplication algorithm*

Example:
To see how this procedure works, the following example is considered. M = –6 = 1010 and Q = 7 = 0111.

	M	A	Q	Size
Initial Configuration	1010	0000	0111 0	4
Step-1				
As Q[0] = 1 and Q[–1] = 0				
A = A – M	1010	0110	0111 0	–
And ARS (AQ)	1010	0011	0011 1	3
Step-2				
As Q[0] = 1 and Q[–1] = 1				
ARS(AQ)	1010	0001	1001 1	2
Step-3				
As Q[0] = 1 and Q[–1] = 1				
ARS (AQ)	1010	0000	1100 1	1
Step-4				
As Q[0] = 0 and Q[–1] = 1				
A = A + M	1010	1010	1100 1	–
ARS (AQ)	1010	1101	0110 0	0

Since the size register becomes 0, the algorithm is terminated and the product is = AQ = 1101 0110, which shows that the product is a negative number. To get the number in familiar form, take the 2's complement of the magnitude. The result is – 42.

Advantages of the Booth's multiplication method:

(i) Pre-processing steps are unnecessary, so the Booth's algorithm treats signed numbers in a uniform way with unsigned numbers.

(ii) Less number of additions and subtractions are required, compared to the sequential multiplication method.

(e) Cache memory: Cache memory is small and fast memory used to increase the instruction-processing rate. Its operation is based on the property called "locality of reference" inherent in programs. Analysis of a large number of typical programs shows that the CPU references to main memory during some time period tend to be confined within a few localized areas in memory. In other words, few instructions in the localized areas in memory are executed repeatedly for some time duration and other instructions are accessed infrequently. This phenomenon is known as the property of *locality of reference*.

If active segments of a program are placed in a fast small cache memory, the average memory access time can be reduced, thus reducing the total execution time of the program. This memory is logically placed between the CPU and main memory. Because we know that the cache memory's speed is almost same as that of CPU. The main idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will be almost same as access time of cache.

The main characteristic of cache memory is its fast access time. Therefore, the waiting time for the CPU is very small or nil when searching for words in the cache. The transfer of data as a block from main memory to cache memory is referred to as a *mapping* process. Three types of cache mapping have been used.

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

Cache memory is one high-speed main memory (SRAM). The cache memory can be placed in more than one level. Most of the recent microprocessors—starting from Intel 80486—have on-chip (memory chip is placed inside the CPU chip) cache memory also known as internal cache. High performance microprocessors such as Pentium pro and later have two levels of cache memory on-chip. These are known as level 1 (L1) and level 2 (L2) caches. An on-chip cache is slightly faster than an off-chip cache of same technology.

Group-A
(Multiple-Choice-Type Questions)

natives from the following:

ber of directly addressable locations in
ntrol bus, 20-bit address bus, and 8-bit

(b) 2K (c) 1 M

k connected tri-state buffers in paralle
k connected buffers

k connected tri-state buffers in paralle
re kept in the
(b) control memory (c) cache me

;X ← AC

tructions, the program code in assemb

;TOS ← A [TOS means top o

;TOS ← B

;TOS ← (A + B)

;TOS ← C

;TOS ← (A + B) * (C)

;X ← TOS

00 rpm
/3600 s = 1/60 s.
rack = 20 * 4000 bytes = 80000 bytes
e = 80000 / (1/ 60) = 4800000 bytes/s

rchitecture? What is Von Neumann bo

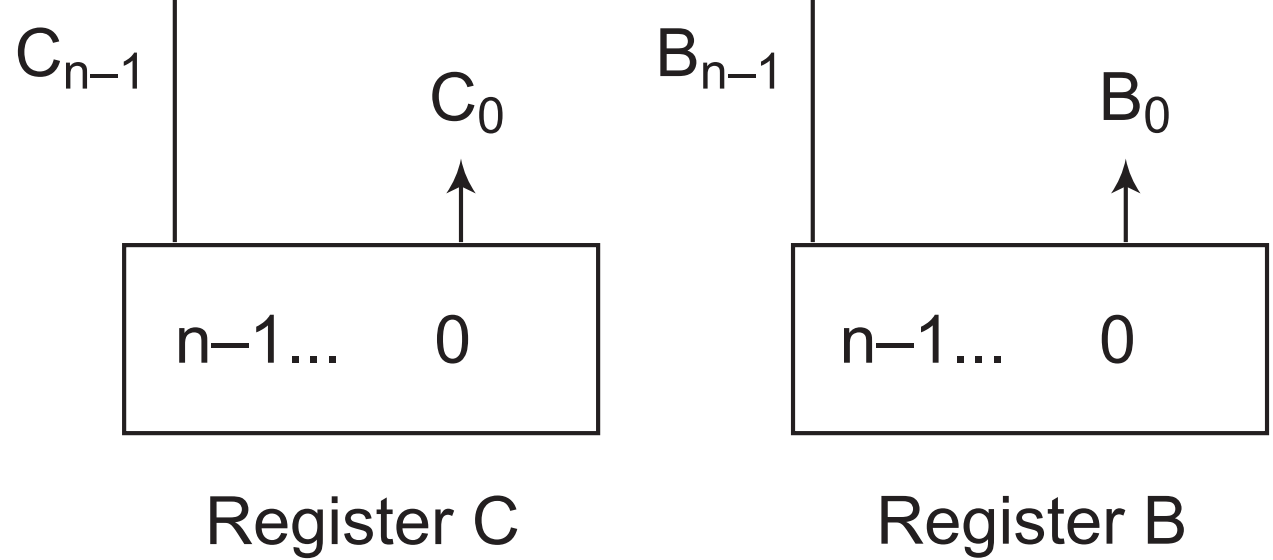
d his colleagues began the design of a

The POP operation results in deleting
stack pointer register.
ows a stack of four data-words in the r

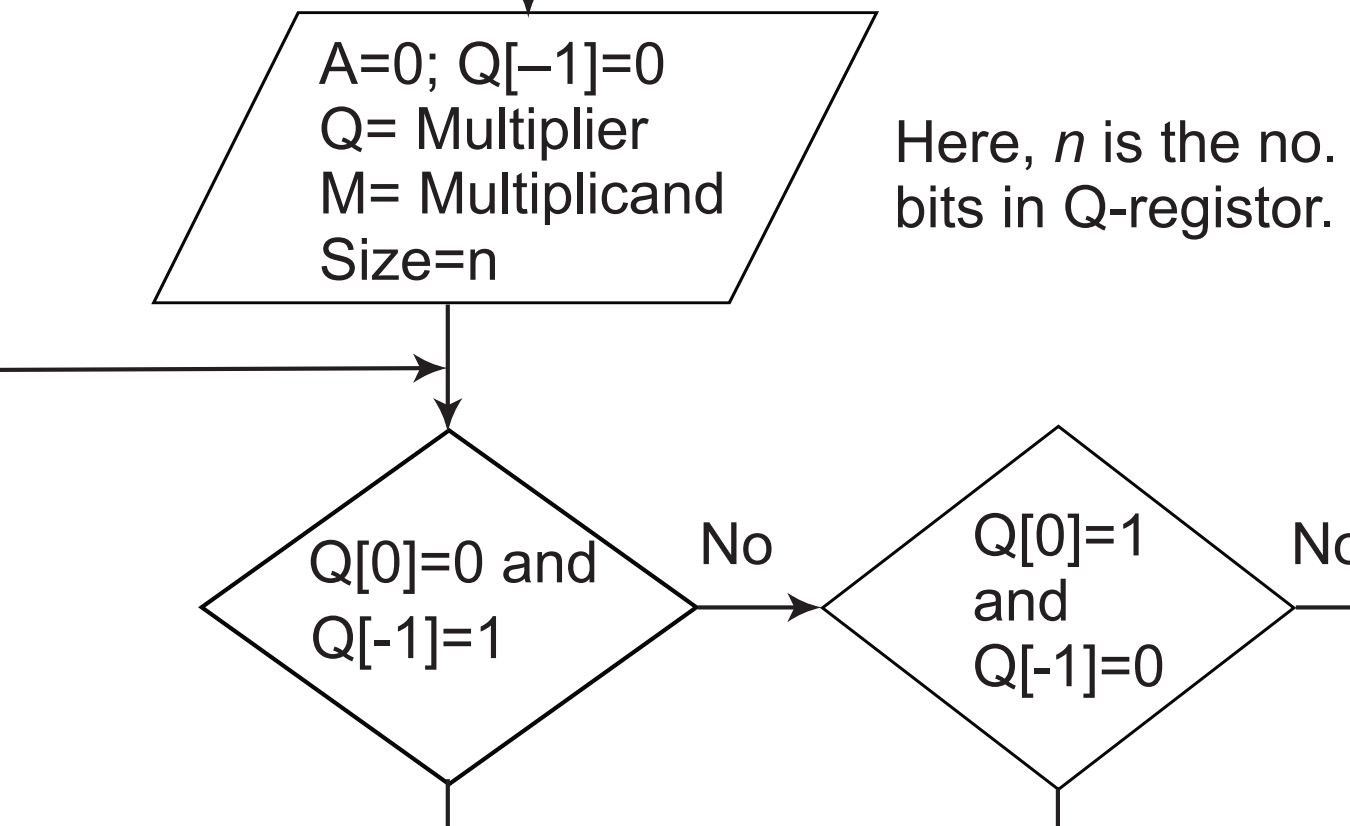
tes the data word at the top of the stac
utomatically in either case. The PUSH

	;decrement the
ddress>	;store the conte
	into SP, i.e., a
e implemented as	
← SP	;transfer the co

data processing tasks as specified by
he main constituent parts of the Central
e CPU is the register unit—collection o
ction temporarily.
his is the unit that supervises the flow
ol unit retrieves the instructions using
is stored in the memory. The instructio
it itself and then the decoded instructio



e 4: *Bus system for four registers using*


A=0; Q[–1]=0
Q= Multiplier
M= Multiplicand
Size=n
Here, n is the no.
bits in Q-registor.
Q[0]=0 and
Q[-1]=1
No
Q[0]=1
and
Q[-1]=0
No

00110 11111 01010 1

00110 00101 01010 1

ks in a cache set.

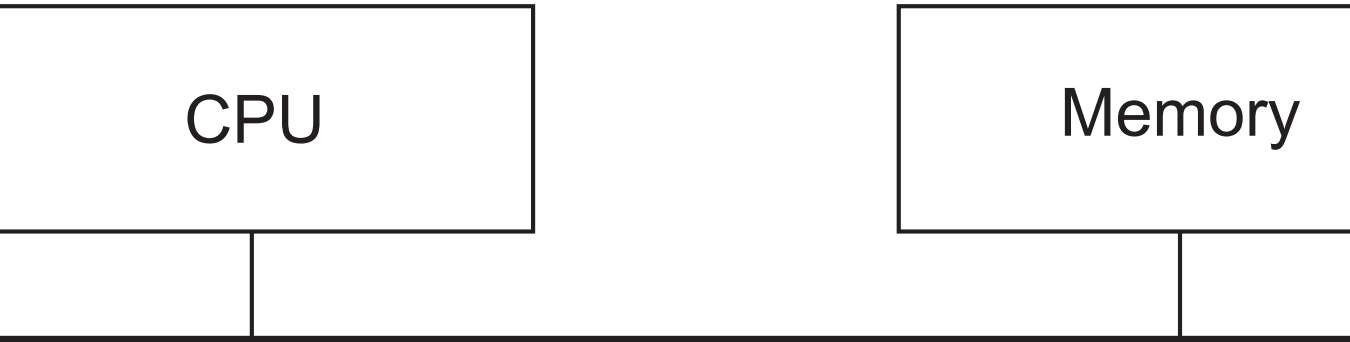
system having cache access time of 5 n
of 0.96 and main memory hit ratio of
o achieve an overall access time of 16

speed main memory, sometimes used t
rent programs and data available to the
main memory, thus the resulting proce

the cache memory contains = 1 MB/1
bits required to specify one block in th
s = 28 – (13 + 7) = 8-bit.
ddress format is

Tag	Block	Word
8	13	7

care of supporting one of the I/O contr
ers is kept in each DMA controller. Su
isk Controller (FDC) and Hard-Disk C
the software.

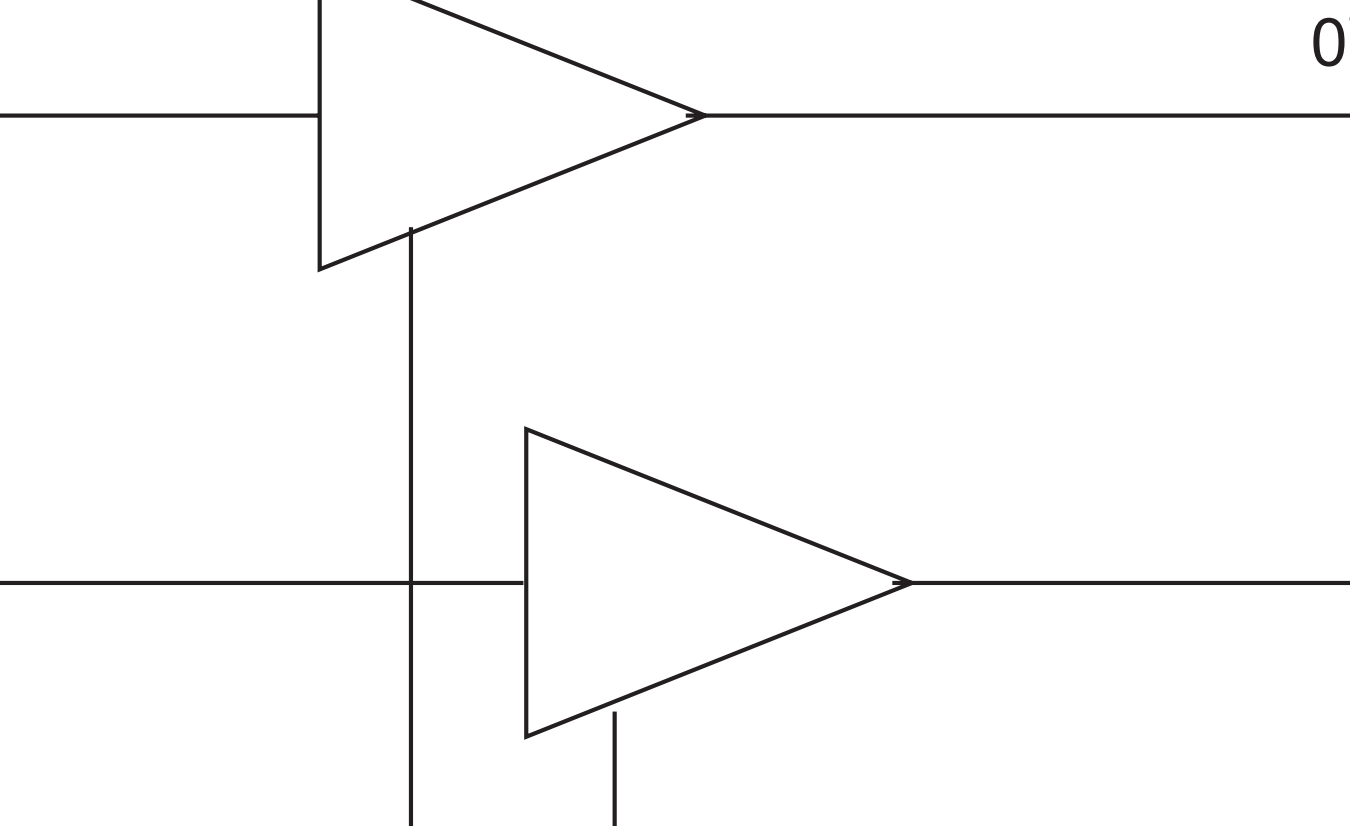


: *DMA controller having multiple DMA*

ed for a single instruction execution is
o go through an instruction cycle that

tion from memory.
uction.
d(s) from memory or register.

rent) Mode: In this mode, the operand
accumulator register is generally used t
tion the result is stored in the same re
s the content of the accumulator left t
: In this mode, the operand is mentio
an immediate-mode instruction contai
n the address field. To initialize registe



the page number and the other is frame
page would be mapped to which fram
ng the page tables; most allocate a pag
d by the CPU (i.e., virtual address) is di
isplacement (d). The page number *p* is
word number within the page *p*. The st

ctions have the following general attri

s a high degree of parallelism
ing of the control information
ruction, a single field can produce an e
in this organization. The vertical micr
e control signals due to the decoding ti

Group-A
(Multiple-Choice-Type Questions)

natives for the following:

mance
ance

(b) increases
(d) none of t

(b) 0000011 (c) 1100010

the ALU is

ational (b) combinat

ial (d) none of t

ansfer but is not involved in checking
erefore, the execution time of the CPU
rograms, when no data transfer is requ
4-bit IEEE floating point representatio
gh' and 'write-back' policies in cache m

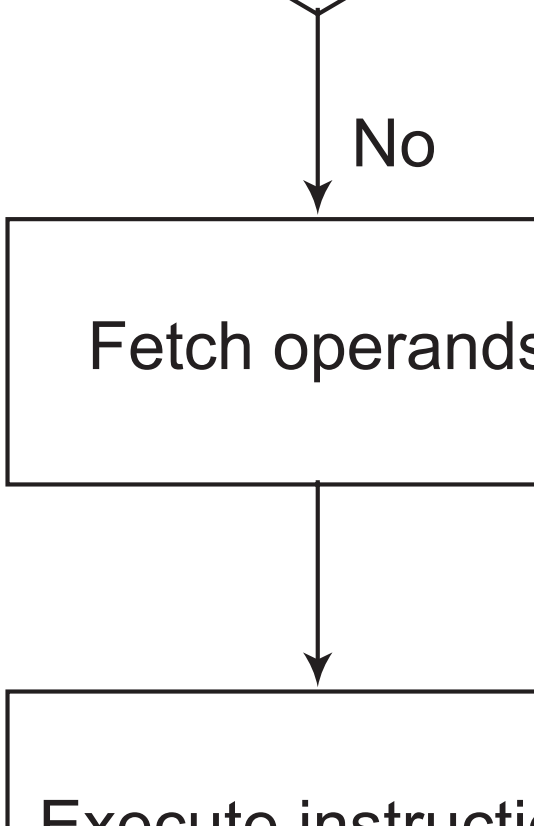
$-9.5 = -1001.1$ in binary $= -1.0011 \times 2$

$M = 0.00110\ 00000\ 00000\ 00000\ 000$

computed by adding the offset value
JR 20) requires 2 bytes: one for the o
der that the instruction is stored in the

es a variety of addressing modes; some
rent) Mode: In this mode, the operand
accumulator register is generally used
xecution, the result is stored in the sam
tes the content of the accumulator left
e: In this mode, the operand is mention
an immediate-mode instruction contai
n the address field. To initialize registe

tion is a technique of moving program
o another part. Relocation is an importa
mode, the content of the Base Register
n to obtain the effective address. This
e, but the exception is in the way they
ess of a memory array of operands and
placement or offset relative to this sta
e has the advantage over index addre


No
Fetch operands

low by keeping transistors T_3 and T_6 c
rly, in the state 0, the voltage at A is lo
and T_6 off, while T_4 and T_5 on. Both t
n it. Thus, for the state 1, if T_1 and T_2
h and low signals respectively. The sta

f using CMOS SRAMs is the low powe
the current flows in the cell only. Othe

memories, which collectively give a m
system is broadly divided into foll

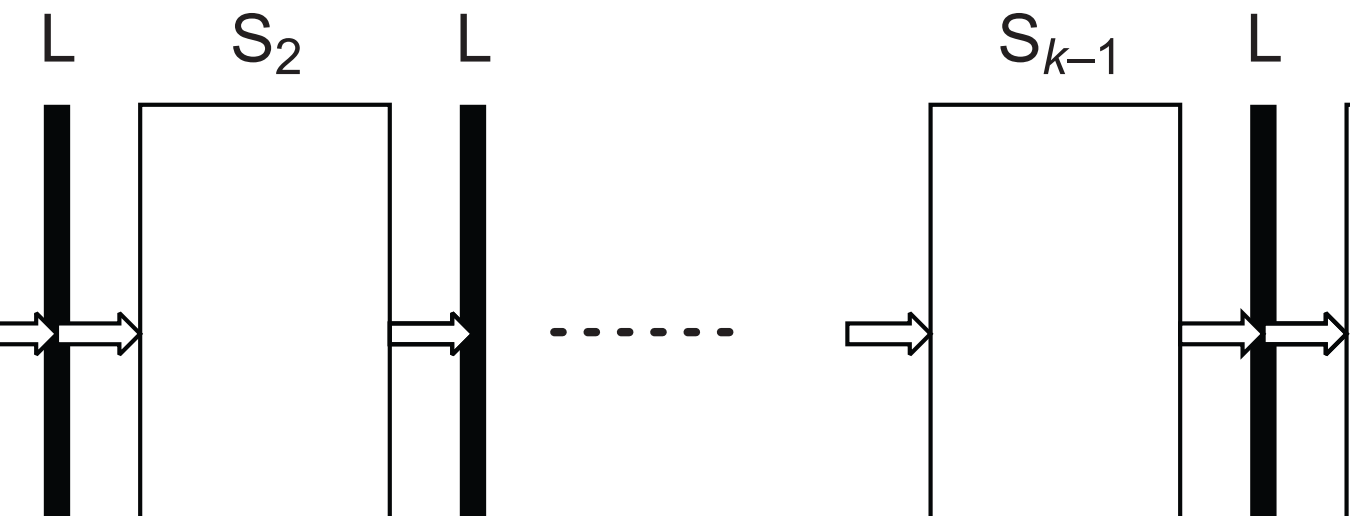
liary) memory
memory

y

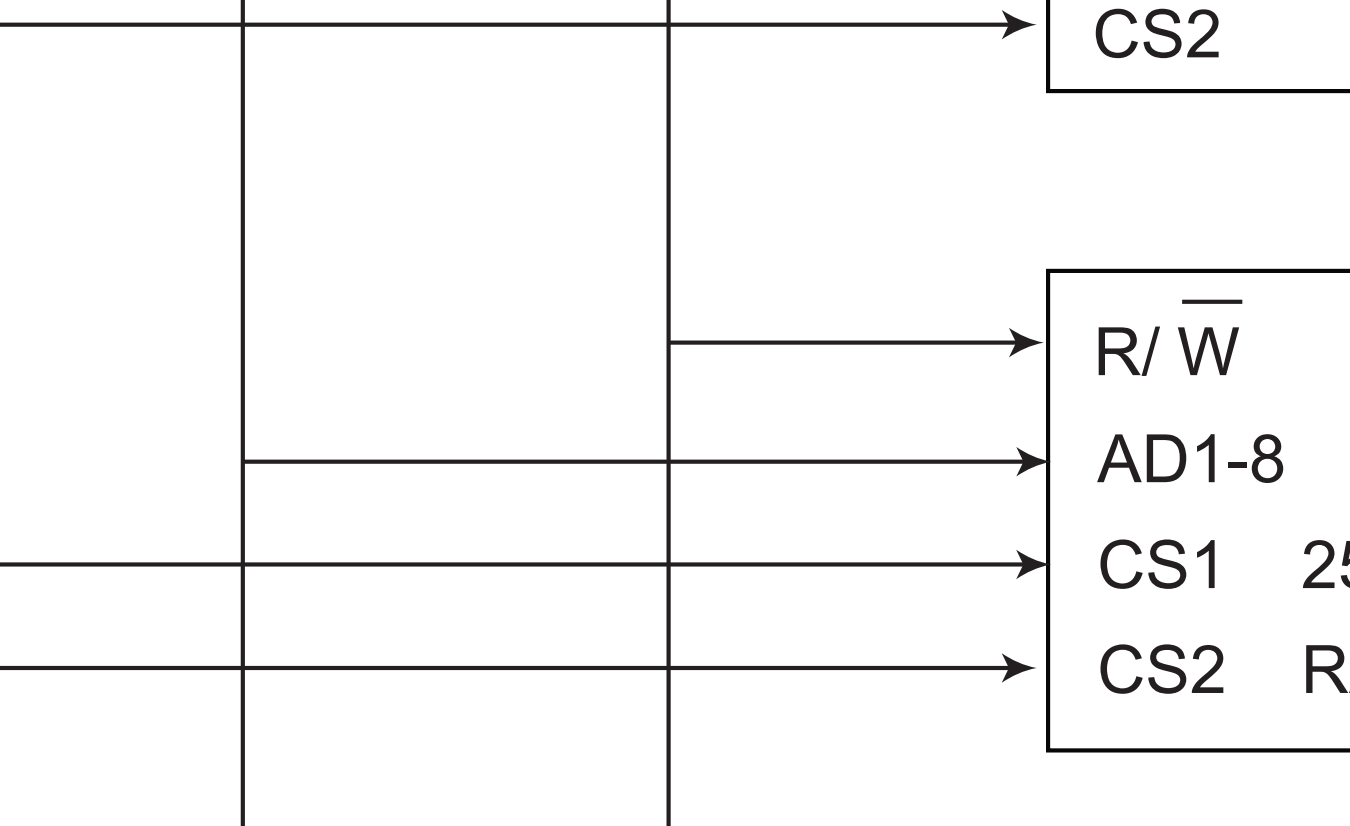
sters and the CPU because they are fa
egisters are very expensive, only a fe

n memory of 64K × 16 and a cache m
ith a block size of 4 words.
are there in tag, index, block, and wor
of one cache word?
ks can be accommodated in the cache?

rforms overlapped computations to expl
ecture uses one parallel processing con



mory can be designed by means of a ta
he memory address space assigned to e
ction to the CPU shown in Figure 6 is c
ress for memory read or write operatio
mory and address lines 9 are used in du
RAMs, the line 9 and line 10 are used
d to the ROM as address line along wit
OM, since the ROM has 512 locations. T


CS2
R/ W
AD1-8
CS1
CS2

he interrupt controller makes liaison
interrupt controller is also assigned a
to signal the controller that it has re
nploying an ISR (Interrupt Service Ro
shown in Figure 7, to set up a priorit
in a chain-fashion with highest priority d
rity devices. When one or more devic
to the CPU, the CPU then sets interr

Group-A
(Multiple-Choice-Type Questions)

.

ottleneck is a problem, which occurs
memory
between CPU and main memory

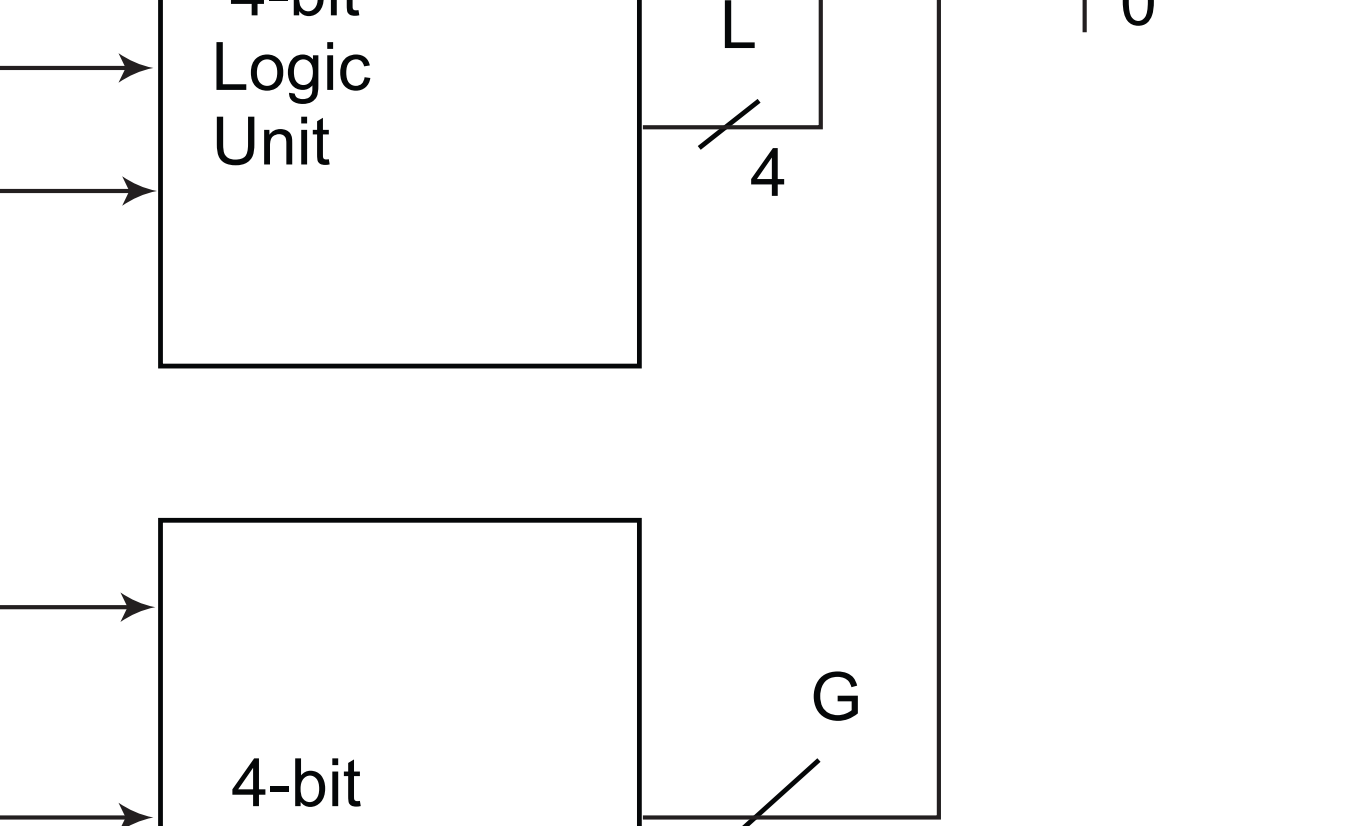
be of one-byte length
RISC processor is
(b) 2 (c) 3

ycle stealing means
pportunity to transfer only one word i

5_{10} = –1010.1 in binary = –1.0101 × 2
0.010100 000000 000000 00000
E + 127 = +3 + 127 = 130 = 1000 00
ive, the sign bit S = 1
e-precision (32-bit) representation is

010100 000000 000000 00000

lower access time, which means it is
ires constant power supply, which mean
hereas the DRAM offers reduced pow
tion is stored in the capacitor.
ively small internal circuitry in the or
capacity in a single DRAM memory c
ize SRAM memory chip. In other wor
ed to the SRAM.
er than DRAM


4-bit
Logic
Unit
L
4
4-bit
G
0

ue of decomposing a sequential task i
cial dedicated stage (segment) that ope
orms partial processing dictated by the
stage is transferred to the next stage in
ruction has passed through all the stag
Stages are pure combinational circuits
ta stream flowing through the pipe. Th
(i.e., collection of registers). Figure 2

= 30 cm

(30 – 20)/2 cm = 5 cm

mm

face = (5*10)/0.5 = 100

nference = 20 * Π cm

ge density = 2000 bits/ cm, which will

city/track = 20 * Π*2000 bits = 122.77

n

e distributed system is faster and in th
pacity, 128-byte lines and is 4-way set-
ssing data in the memory.
and sets does the cache have?
es are required for tag?
of tags are required in each entry in the
hain memory subsystem has the follow
ccess time 80 ns