PG CA S2-01 (DSC)

TWO YEAR MASTER'S DEGREE COURSE

(CBCS in Accordance with NEP 2020)

SEMESTER - II

COMPUTER ORGANIZATION AND ARCHITECTURE

MSc IT

Hiranya Chandra Bhuyan School of Science and Technology

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakoti Mahanta, Gauhati University
Prof. Hiren Kumar Deva Sarma, Gauhati University
Dr. Lipi B Mahanta, IASST, Guwahati
Dr. Sanjib Kr. Kalita, Gauhati University

Course Co-ordinator : Dr. Nabankur Pathak, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
1	Dr. Chakradhar Das, Bongaigaon Polytecnique College
2, 3, 10	Dr. Kshirod Sarmah, Pandit Deendayal Upadhyaya Adarsha Mahavidyalaya
4	Mr. Kashyap Mahanta, KKHSOU
5, 6 7, 11	Ms. Jonalee Barman Kakati, NERIM
8, 9	Dr. Nabankur Pathak, KKHSOU
12, 15	Dr. Pranab Das, Assam Don Bosco University
13, 14	Ms. Shobhanjana Kalita, Tezpur University

Editorial Team

Content	:	Dr. Jyotiprokash Goswami, Assam Engineering College
Language	:	
Structure, Format & Graphics	:	Dr. Tapashi Kashyap Das, KKHSOU

April, 2024

ISBN :

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

Headquarters: Patgaon, Rani Gate, Guwahati-781017
City Office: Resham Nagar, Khanapara, Guwahati-781022; Web: www.kkhsou.in

The University acknowledges with thanks the financial support provided by the Distance Education Bureau, UGC for preparation of this material.

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
Computer Organization and Architecture
CONTENTS

		Pages
UNIT 1:	**Digital Components** Half Adder, Full Adder, Half Subtractor, Full Subtractor, Coder-Decoder, Multiplexer, De-multiplexer, Magnitude Comparator, Flip-Flops, Counter, Register.	**7-44**
UNIT 2:	**Data Representation** Data Representation, Computer Arithmetic and their Implementation; Control and Data Path, Data Path Components, Design of ALU and Data Path, Control Unit Design	**45-66**
UNIT 3:	**Computer Arithmetic** Integer Representation, Sign Magnitude Representation, Twos Complement Representation, Integer Arithmetic.	**67-79**
UNIT 4:	**Basic Organization of Computer and Design** Basic Organization of the Computer and Block Level Description of the Functional Units from Program Execution Point of View; Fetch, Decode and Execute Cycle.	**80-97**
UNIT 5:	**Instruction Sets** Elements of a Machine Instruction, Instruction Representation, Simple Instruction Format, Instruction Types, Number of Addresses, Types of Operands, Types of Operations. Different Instruction Formats, Instruction Types, Instruction Execution, Assembly Language Notation.	**98-115**
UNIT 6:	**Addressing Modes** Addressing: Immediate, Direct, Indirect, Registrar, Registrar indirect, Relative Index.	**116-126**
UNIT 7:	**Input-Output Organization** Different I/O techniques (Programmed I/O, Interrupt-Driven I/O), DMA (Direct Memory Access), I/O Processors.	**127-138**

UNIT 8: Introduction to Cache and Virtual Memory **139-160**

Memory Hierarchy, Semiconductor Memories, Internal Organization of Typical RAM and ROM Memory, Switches, Cache Memory, Cache Memory Access Techniques; Mapping Functions, Virtual Memory, Locality of Reference,Paging, Cache Coherence Problem.

UNIT 9: Memory and I/O Access **161-171**

Memory Read Write Operations, Concept of Handshaking, Polling Techniques(Serial and Hub Polling) and Interrupt Driven I/O, Priority and Daisy Chaining Technique, Introduction to Memory Mapping.

UNIT 10: Memory and I/O Interfacing **172-186**

I/O Processor, Priority Encoder, Device Scheduler, Interfacing with the I/O Devices, Keyboard, Printer and Display Interfaces.

UNIT 11: External Memory **187-209**

Magnetic Disk, Magnetic Read Write Operation, Sector, Track, Inter Track and Inter Sector Gap, Cylinder, Fixed Head Disk, Seek Time, Access Time,Transfer Time, Rotational Delay, RAID, Optical Memory, Magnetic Tap.

UNIT 12: Processor Structure and Function **210-225**

Processor Organization: Fetch Instruction, Interpret Instruction, Fetch Data,Process Data, Write Data, Registrar Organization, Control and Status Registrar, Instruction Cycle, Instruction Pipelining (Definition only).

UNIT 13: Introduction to Parallel Processing **226-236**

Introduction to Pipelining and Basics of Parallel Processing, Scalable Architecture.

UNIT 14: CISC and RISC Architecture **237-244**

CISC (Complex Instruction Set Computers), RISC (Reduced Instruction Set Computers), Examples of CICS and RISC.

UNIT 15: Parallel Architectures **245-264**

SISD, SIMD, MISD, MIMD, Scalar, Flynn's Classification of Computer Architecture, Vector, Superscalar and Pipelined Processor, Pipelining,Vector Computing, Arithmetic Pipeline, Vector and Scalar Register, Chaining, Scatter Gather Operations, Vector-Register Processor, Memory-Memory Vector Processor. Array Processor.

Reference **265**

COURSE INTRODUCTION

Welcome to the exciting world of Computer Organization and Architecture! In today's digital age, understanding the inner workings of computers and their architectural foundations is essential for anyone interested in computer science and technology. This course is designed to take you on a journey through the fundamental components and principles that make up modern computing systems.

Unit 1: Digital Components

We'll begin our exploration by delving into the digital components that form the building blocks of computers. You'll learn about critical elements like Half Adders, Full Adders, Multiplexers, Flip-Flops, and more, which are the fundamental units responsible for processing and storing data.

Unit 2: Data Representation

Understanding how data is represented within a computer is paramount. In this unit, we'll dive into data representation, computer arithmetic, and the implementation of these concepts. You'll discover the inner workings of control and data paths, as well as the design of Arithmetic Logic Units (ALUs) that perform mathematical operations.

Unit 3: Computer Arithmetic

The world of numbers within computers is vast. Here, you'll learn about various integer representations, including Sign Magnitude and Twos Complement, as well as how computers perform arithmetic operations with these representations.

Unit 4: Basic Organization of Computer and Design

Uncover the basic organization of a computer system and explore its functional units from a program execution perspective. We'll walk you through the critical fetch, decode, and execute cycle that forms the foundation of computing.

Unit 5: Instruction Sets

Delve into the language computers speak with their machine instructions. You'll gain insights into different instruction formats, types, and execution, and become proficient in assembly language notation.

Unit 6: Addressing Modes

Learn how computers locate and access data through various addressing modes, including Immediate, Direct, and Indirect addressing. These modes are the keys to efficient memory management.

Unit 7: Input-Output Organization

Discover different input and output techniques, from Programmed I/O to Interrupt-Driven I/O,

and explore concepts like Direct Memory Access (DMA) and I/O Processors that enable efficient data exchange between a computer and its peripherals.

Unit 8: Introduction to Cache and Virtual Memory

Explore memory hierarchy, cache memory, and virtual memory systems, which enhance a computer's performance and storage capabilities. Dive into concepts like paging and cache coherence.

Unit 9: Memory and I/O Access

Learn the intricacies of memory read/write operations, handshaking protocols, and polling techniques. Understand how interrupts are used to manage I/O operations.

Unit 10: Memory and I/O Interfacing

Discover how computers interface with I/O devices, including keyboards, printers, and displays. Gain insight into device scheduling and priority encoding.

Unit 11: External Memory

Explore external storage devices like magnetic disks, optical memory, and RAID systems, and understand their role in data storage and retrieval.

Unit 12: Processor Structure and Function

Delve into the heart of a computer—the processor. Learn about processor organization, instruction cycles, and the concept of instruction pipelining.

Unit 13: Introduction to Parallel Processing

Get introduced to pipelining and the basics of parallel processing, a critical concept in modern computer architecture.

Unit 14: CISC and RISC Architecture

Compare and contrast Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC), examining examples of both.

Unit 15: Parallel Architectures

Explore different types of parallel architectures, including SISD, SIMD, MISD, MIMD, and vector processing. Gain an understanding of the concepts and technologies behind high-performance computing.

By the end of this course, you'll have a deep understanding of the inner workings of computers, their architectural principles, and the technologies that power modern computing systems. Whether you're aspiring to be a computer scientist, engineer, or simply curious about how the digital world operates, this course will equip you with the knowledge you need to succeed in today's tech-driven landscape. Get ready for an exciting journey into the world of Computer Organization and Architecture.

UNIT 1: DIGITAL COMPONENTS

UNIT STRUCTURE

1.1 Learning Objectives

1.2 Introduction

1.3 Combinational Circuits

- 1.3.1 Half-Adder
- 1.3.2 Full-Adder
- 1.3.3 Half-Subtractor
- 1.3.4 Full-Subtractor
- 1.3.5 Multiplexer
- 1.3.6 Demultiplexer
- 1.3.7 Encoder
- 1.3.8 Decoder
- 1.3.9 Magnitude Comparator

1.4 Sequential Circuits

1.5 Flip-Flops

- 1.5.1 RS Flip-Flop
- 1.5.2 D Flip-Flop
- 1.5.3 JK Flip-Flop
- 1.5.4 MS Flip-Flop

1.6 Counters

- 1.6.1 Asynchronous Counter
- 1.6.2 Synchronous Counter

1.7 Register

- 1.7.1 Serial In – Serial Out Register
- 1.7.2 Serial In – Parallel Out Register
- 1.7.3 Parallel In – Serial Out Register
- 1.7.4 Parallel In – Parallel Out Register

1.8 Let Us Sum Up

1.9 Further Reading

1.10 Answers to Check Your Progress

1.11 Model Questions

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define Combinational circuit and Sequential circuit
- describe the working principle of Half-adder, Full-adder
- describe the working principle of Half-subtractor, Full-subtractor
- describe the working principle of Multiplexer, Demultiplexer
- describe the working principle of Encoder, Decoder
- describe the working principle of Flip-Flop, Register, Counter.

1.2 INTRODUCTION

This is the first unit of this block. In this unit, we will discuss the various combinational and sequential circuits to build your understanding on different digital components that are used in digital applications. A sound knowledge on these components will make you feel confident to understand the working principle of many digital devices in general and the computer system in particular. Combinational circuits, sequential circuits along with counters and registers are covered in detail in this unit. In the next unit, we will explore different data representation systems.

1.3 COMBINATIONAL CIRCUITS

To obtain a desired output according to a Boolean equation various logic gates are often interconnected. These circuits are called combinational circuits. These are the combinations of different fundamental logic gates and hence the name Combinational Circuit. Its main characteristic is that the output is solely determined by the present inputs.

Inputs to a combinational circuit are given as or combinations of 0's and 1's and outputs are also available as 0 or 1 or combinations of 0's and 1's. In the process of designing a combinational circuit, first, a truth table is formed for the Boolean expression that describes the combinational circuit. Next, the function is simplified and then the simplified function is implemented with gates to obtain the logic diagram of the combinational circuit. We may

also obtain the Boolean function/expression from this logic diagram after determining the truth table.

1.3.1 Half-Adder

Half-adder is a circuit that can add two binary bits. Its outputs are SUM and CARRY. The following truth table shows the various combinations of inputs and their corresponding outputs of a half-adder. X and Y denote inputs and C and S denote the two outputs CARRY and SUM.

Table 2.1: Truth Table for a Half-Adder

X	Y	CARRY (C)	SUM (S)
0	0	0	0
0	1	0	1 (Y)
1	0	0	1 (X)

The minterms for SUM and CARRY for which SUM or CARRY are 1 are shown in the bracket. The *Sum-of-Product* (SOP) equation for SUM is:

$S = \bar{X}Y + X\bar{Y}$ … … …(1)

$= X \oplus Y$

Similarly, the SOP equation for the CARRY is:

$C = XY$ … … …(2)

Combining the logic circuits for equation (1) & (2) we get the circuit for Half-Adder as:



Figure 2.1: Half-Adder Circuit and Symbol

1.3.2 Full-Adder

Full-Adder is a logic circuit to add three binary bits. Its outputs are SUM and CARRY. In the following truth table X, Y, Z are inputs and C/ and S/ are the outputs to mean CARRY and SUM respectively.

Table 1.2: Truth Table for Full-Adder

X	Y	Z	CARRY (C')	SUM (S')
0	0	0	0	0
0	0	1	0	1 (Z)
0	1	0	0	1 (Y)
0	1	1	1 (YZ)	0
1	0	0	0	1 (X)
1	0	1	1 (X Z)	0
1	1	0	1 (XY)	0
1	1	1	1 (XYZ)	1 (XYZ)

The minterms are written in the brackets for each 1 output in the truth table. From these the SOP equation for full adder can be written as:

$S' = \quad Z + \quad Y \quad + XY \quad + XYZ$

$= \quad (\quad Z + Y \quad) + X(\quad \quad + YZ\ )$

$= \quad S + X \quad$

$= X \oplus S$ … … …(3)

Here, S is SUM of Half-Adder. Again, SOP equation for Full-Adder CARRY is:

$C' = \quad YZ + X \quad Z + XY \quad + XYZ$

$= \quad YZ + XYZ + X \quad Z + XY \quad$

$= (\quad + X)YZ + X(\quad Z + Y \quad)$

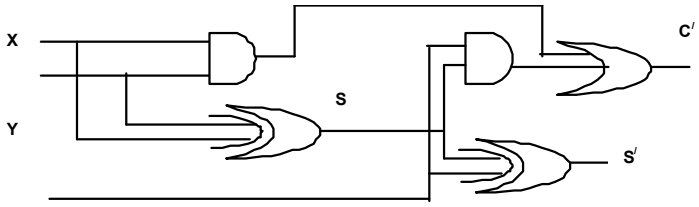
$= YZ + XS$

$= C + XS$ … … …(4)

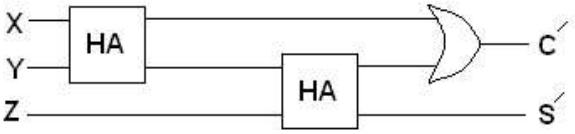
Here also C means CARRY of half-adder and S means SUM of half-adder.

Note: We are using C' and S' to mean CARRY and SUM of full-adder just only to differentiate them from the CARRY and SUM of the half adder.

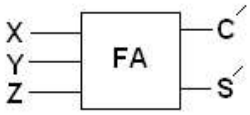
Now, using two half-adder circuits and one OR gate we can implement equation (3) and (4) to obtain a full-adder circuit as follows:



(a) Logic Diagram



(b) Block Diagram



(c) Symbol

Figure 1.2: Full-Adder Circuit

1.3.3 Half-Subtractor

A half-subtractor subtracts one bit from another bit. It has two outputs. Those are: DIFFERENCE (D) and BORROW (B).

Table 1.3: Truth Table for Half-Subtractor

X	Y	BORROW (B)	DIFFERENCE (D)
0	0	0	0
0	1	1 (Y)	1 (Y)
1	0	0	1 (X)
1	1	0	0

The minterms are written within parenthesis for output 1 in each column. The SOP equations are:

D = Y + X

= X ⊕ Y

= S (SUM of half-adder) … … …(5)

B = Y … … …(6)



Figure 2.3: The Half-Subtractor Circuit and the Symbol

1.3.4 Full-Subtractor

A full-subtractor circuit can find Difference and Borrow arises on the subtraction operation involving three binary bits.

Table 1.4: Truth Table of Full-Subtractor

X	Y	Z	BORROW (B')	DIFFERENCE (D')
0	0	0	0	0
0	0	1	1 (Z)	1 (Z)
0	1	0	1 (Y)	1 (Y)
0	1	1	1 (YZ)	0
1	0	0	0	1 (X)
1	0	1	0	0
1	1	0	0	0
1	1	1	1 (XYZ)	1 (XYZ)

The SOP equation for the DIFFERENCE is:

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

$= Y + X + XYZ + Z$

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

$= D + Z$ … … …(7)

And SOP equation for BORROW is:

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

$= Z + XYZ + Y + YZ$

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

$= Z + Y$ … … …(8)

In equation (7) and (8), D stands for DIFFERENCE of half-subtractor. Now, from the equations (7) and (8) we can construct a full-subtractor using two half-subtractor and an OR gate.

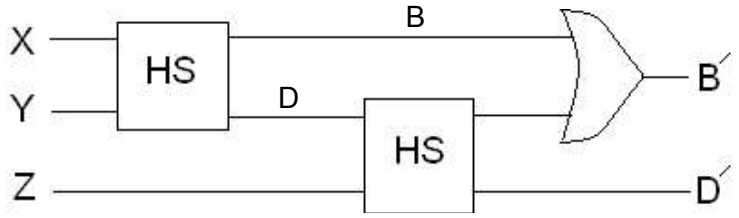
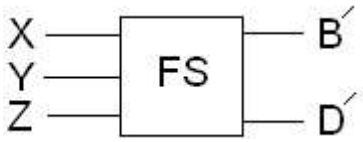


Figure 2.4: Full-Subtractor Circuit

The symbol of full-subtractor is:





CHECK YOUR PROGRESS

Q.1: Choose the appropriate option:

i) A half-adder can add–

a) Two binary number of 4 bit each

b) Two binary bits

c) Add half of a binary number

d) None of these

ii) A full-adder is a logic circuit that has two outputs namely–

a) product & Sum b) sum & borrow

c) sum & carry d) carry & borrow

iii) A half-subtractor can perform–

a) subtraction of two binary bits

b) product of two binary bits

c) complement of half binary bits

d) none of these

iv) A full-subtractor has the ability to do:

a) subtraction of two binary numbers

b) subtraction of three binary bits

c) product of three binary bits

d) division three binary bits

1.3.5 Multiplexer

Multiplexer is a circuit which has many inputs and only one output. Multiplexer can select any one of its many inputs by applying a control signal and steer the selected input to the output. Generalized block diagram of a multiplexer is shown in figure 2.5.

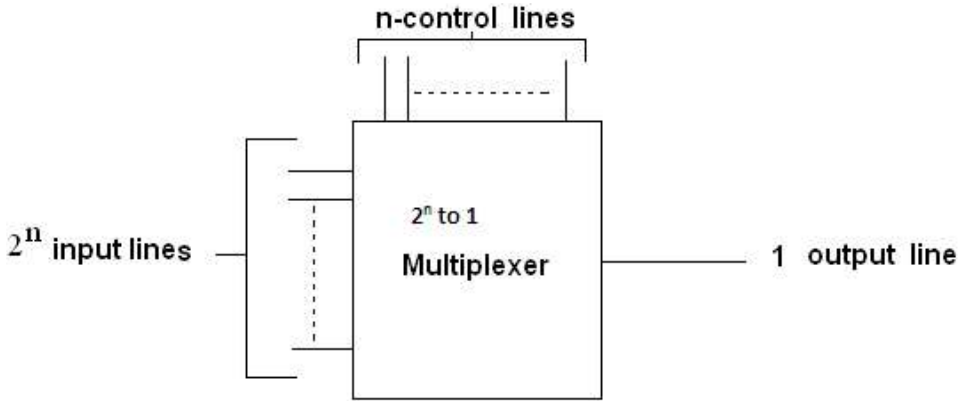


Figure 1.5: Block Diagram of Multiplexer

It has 2^n inputs, n numbers of control or selection lines and only one output. A multiplexer is also called a many-to-one data selector.

8-to-1 MULTIPLEXER: Figure 1.6 shows a 8-to-1 multiplexer, where there are 8 inputs, 3 control/selection lines and 1 output. The eight inputs are labeled as I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7 and the control/selection lines are as A, B, C, which input is steered to the output depends on the value of ABC. As for example, if,

ABC = 000

then the upper AND gate is enabled and all other AND gates are disabled. As a result, the input I_0 alone is steered to the output. Similarly if,

ABC = 110

then the AND gate connected to the data line I_6 is enabled while all the other AND gates are disabled. Therefore, the input I_6 appears at the output. Hence when, ABC = 110, the output is Y = I_6.

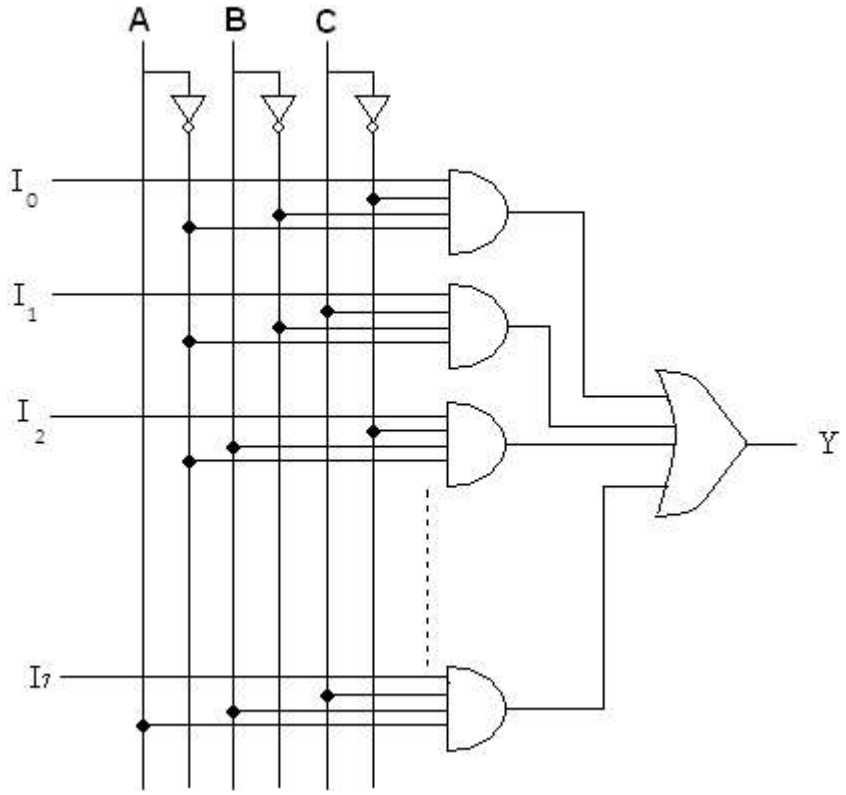


Figure 1.6: 8-to-1 Multiplexer

16-to-1 MULTIPLEXER: Figure 1.7 shows a 16-to-1 multiplexer. In this circuit, there are 16 data input lines, 4 control lines and 1 output. The input lines are denoted by I_0, I_1, I_2, I_3, ... I_{15} and the control lines are by ABCD. The output is denoted by Y.

Out of 16 input lines, only one is transmitted to the output depending upon the value of ABCD. If,

ABCD = 0000

then, I_0 is steered to the output since the upper AND gate is enabled alone and all others are disabled. Similarly if,

ABCD = 0010

then, I_2 appears at the output. If,

ABCD = 1111

then the last AND gate is only enabled and therefore I_{15} appears at the output.

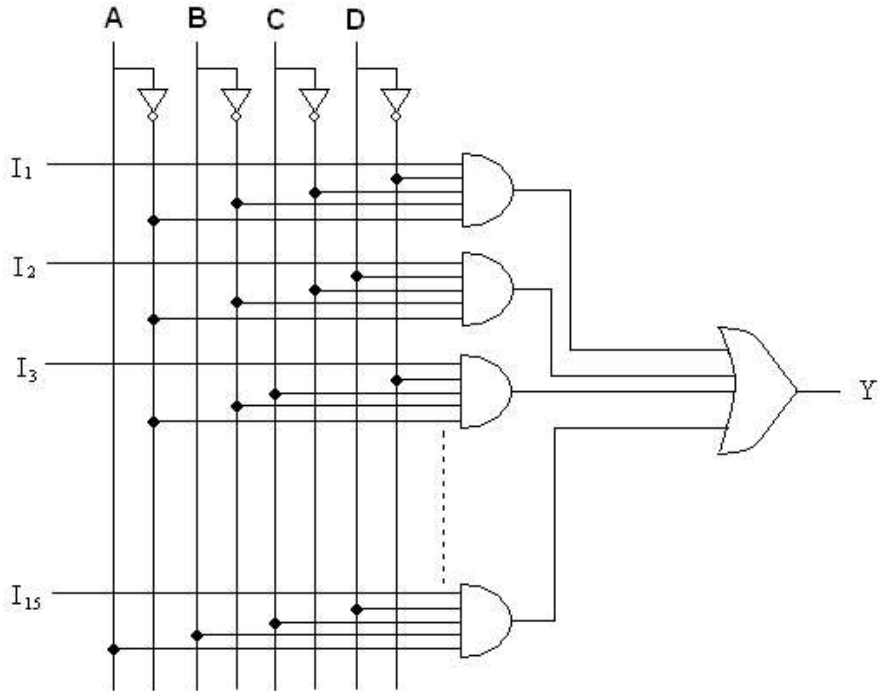


Figure 1.7: 16-to-1Multiplexer

1.3.6 De-Multiplexer

It is opposite to the multiplexer. De-multiplexer means One-to-Many. It has 1 input and many outputs. With the application of appropriate control signal, the common input data can be steered to one of the output lines. Figure 2.8 shows a generalized block diagram of a de-multiplexer.

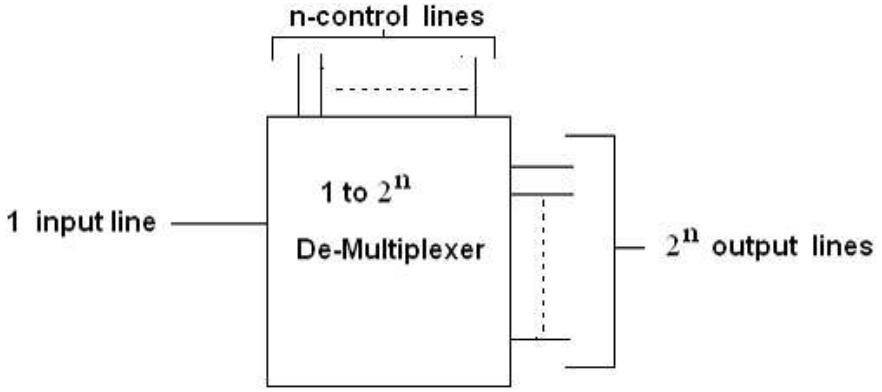


Figure 1.8: Block Diagram of a De-Multiplexer

To have 2^n output lines, there must be n control lines in a de-multiplexer.

1-to-16 DE-MULTIPLEXER: In figure 2.9 we have shown a 1-to-16 de-multiplexer.

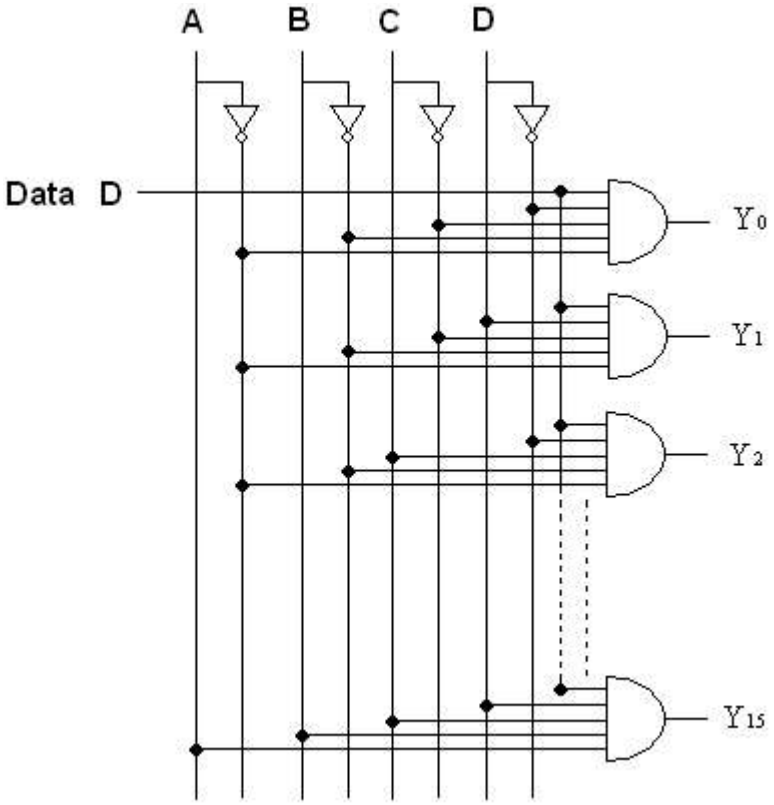


Figure 1.9: 1-to-16 De-Multiplexer

Here the data input line is denoted by D. This input line is connected to all the AND gates through which output appears. Depending upon the control signal, only one AND gate becomes enabled and the data input D appears through that AND gate. So, when ABCD = 0000, the upper AND gate is enabled and data input D appears at Y_0 as output.

When ABCD = 1111, the bottom AND gate becomes enabled and D appears at Y_{15} as output. For other combination, D appears at other output terminal.



CHECK YOUR PROGRESS

Q.2: Choose the appropriate option:

i) Multiplexer means–

a) multiple to many b) one to many

c) many to one d) one to one

ii) A 16-to-1 multiplexer has–

a) 1 control lines b) 2 control lines

c) 3 control lines d) 4 control lines

iii) De-multiplexer means–

a) deduct multiple bits b) one-to-many

c) multiple-to-multiple d) one-to-one

1.3.7 Encoder

An encoder converts a digital signal into a coded signal. A generalized view of an encoder is shown in Figure 2.10.

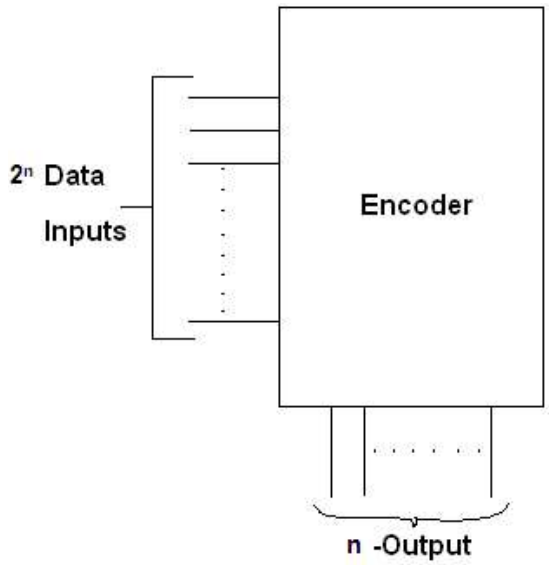


Figure 1.10: Block Diagram of an Encoder

In the figure we can see that there are 2^n input lines and n numbers of output lines. Out of inputs, only one input line is active at a time. Encoder generates a coded output which is unique for each of the active input.

DECIMAL-TO-BCD ENCODER:

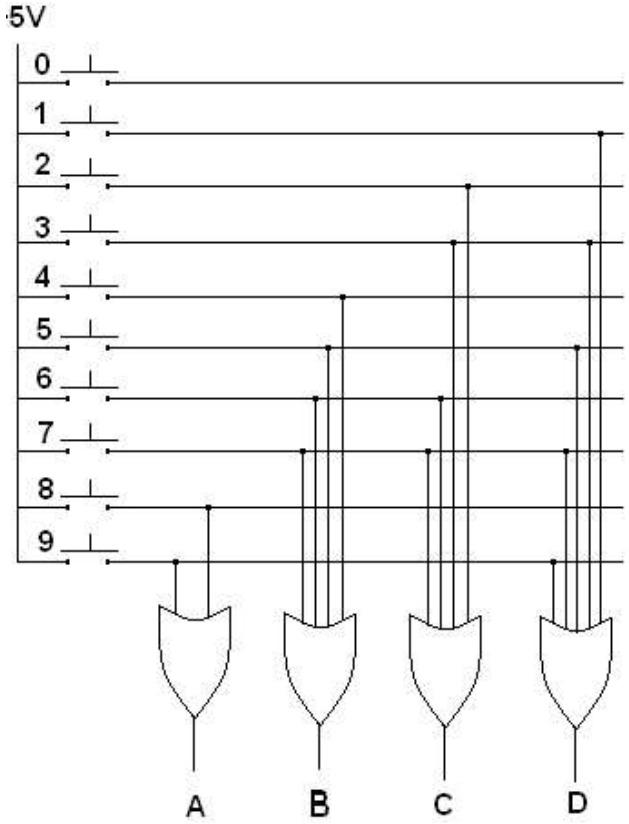


Figure 1.11: Decimal-to-BCD Encoder

A decimal to BCD encoder is shown in Figure 1.11. This circuit generates BCD output when any one of the push button switches is pressed. As for example, if button 6 is pressed, the B and C OR gates have high inputs and the corresponding output becomes–

ABCD = 0110

If button 8 is pressed, the OR gate A receives a high input and therefore the output becomes–

ABCD = 1000

1.3.8 Decoder

A decoder is a digital circuit which has n-input lines and 2^n output lines. A decoder and a de-multiplexer has similarity. In a de-multiplexer, there is a single input line connected to every output 'AND' gate whereas in a decoder the input line is absent.

Let us draw a 3–to–8 decoder as shown in Figure 1.12. Its truth table is shown below:

Table 1.5: Truth table for a 3-to-8 decoder

Inputs			Outputs							
A	**B**	**C**	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

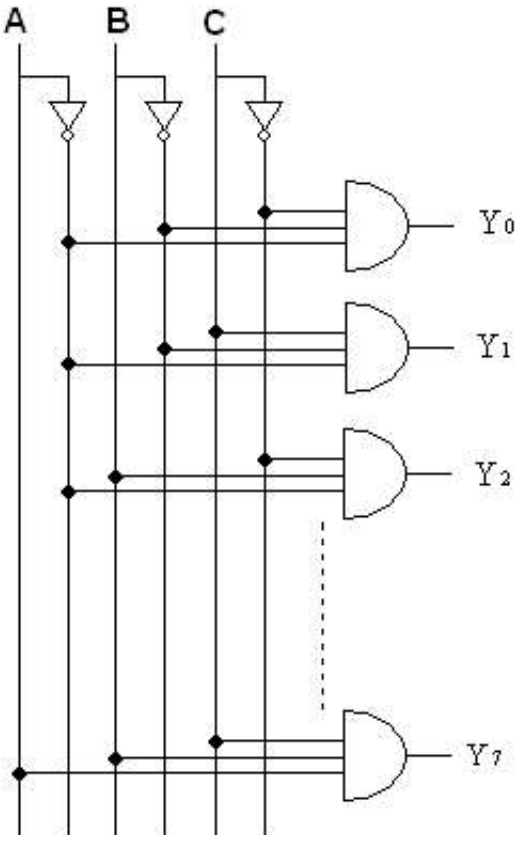


Figure 1.12: 3-to-8 Decoder

1.3.9 Magnitude Comparator

A magnitude comparator circuit can compare two binary numbers to determine which one is greater than the other or their equality. Such a magnitude comparator has three output lines for A > B, A = B, A < B where, A & B are two n-bits binary numbers. Every bit of one number is compared with the corresponding bit of the other number by XOR gate.

A 4-bit magnitude comparator, SN 7485 is available in chip form the block diagram of which is shown in Figure 1.13. It compares two 4-bit binary numbers $A_3A_2A_2A_1A_0$ and $B_3B_2B_1B_0$. Three output terminals are available for A < B, A = B and A > B.

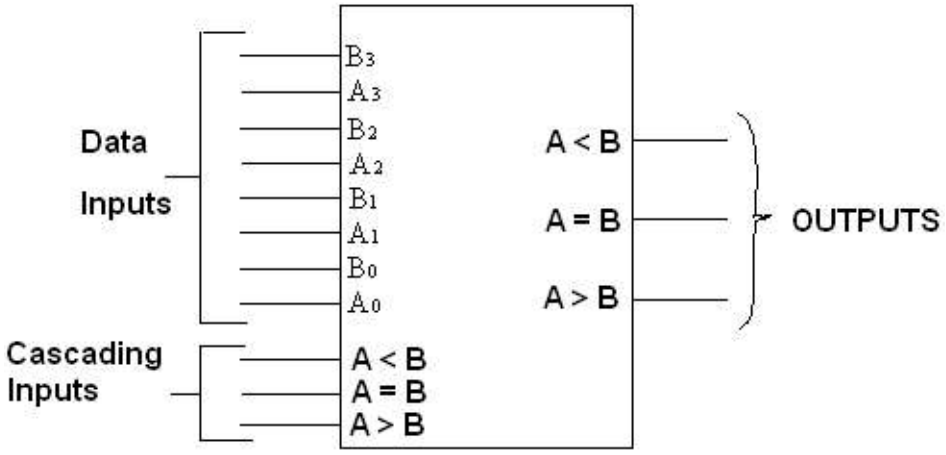


Figure 1.13: Two 4-bit Words Magnitude Comparator SN 7485

To compare any number having more than 4-bits, two or more such chip can be cascaded. The A > B, A < B and A = B outputs of a stage that handles less significant bits are connected to the corresponding cascading inputs of the next stage that handles the more significant bits.

Table 1.6: Function Table SN 7485

Comparing Inputs				Cascading Input			Outputs		
A_3B_3	A_2B_2	A_1B_1	A_0B_0	A > B	A < B	A = B	A > B	A < B	A = B
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H



CHECK YOUR PROGRESS

Q.3: Choose the correct option:

i) An encoder–

a) converts a digital input to another form of digital output.

b) converts analog input to digital output

c) selects one out of many inputs.

d) none of these.

ii) Decoder has n inputs line and–

a) n output lines b) 2^n output lines.

c) n^2 output lines d) no output lines.

iii) Magnitude comparator–

a) compares two multi bit binary number.

b) magnify any digital signal.

c) compress binary numbers.

d) check error in a binary number.

1.4 SEQUENTIAL CIRCUITS

If the output of a circuit depends on its present inputs and the immediate past output, then the circuit is called **sequential circuit**. To build a sequential circuit, we need memory circuits and combinational circuits. Flip-Flop is used as memory circuit the application of which we would see in counter, register etc.

1.5 FLIP-FLOPS

A digital circuit that can produce two states of output, either high or low is called a multivibrator. Three types of multivibrators are: monostable, bi-stable and a stable.

A Flip-Flop is a bi-stable multivibrator and therefore it has two stable states of output-either high or low. Depending on its inputs and previous output, its new output is either high (or 1) or low (or 0). Once the output is fixed, the inputs can be removed and then also the already fixed output will be retained by the flip-flop. Hence a flip-flop can be used as basic circuit of memory for storing for one bit of data. To store multiple bits we can use multiple numbers of flip-flop. Flip-flops are also used to build counter, register etc.

There are many types of Flip-Flops. These are:

- RS Flip-flop
- D Flip-Flop
- JK Flip-Flop
- MS Flip-Flop

1.5.1 RS Flip-Flop

RS Flip-Flop is also called Set-Reset Flip-Flop. An RS flip-flop can be built using many different circuits. Here we have shown two RS flip-flop circuits using NOR and NAND gate. To study its working principle, we can use any one of them. In the following section, let us consider the RS flip-flop using NOR gate.

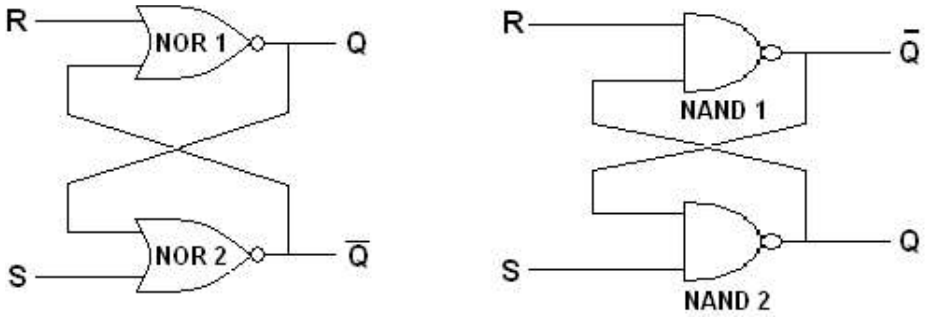


Figure 1.14: Basic Circuit of RS Flip-Flop using NOR and NAND Gate

RS flip-flop has two inputs– Set (s) and Reset (R). It has two outputs, Q and . It should be noted is always the complement of Q. Various combinations of inputs and their corresponding outputs are listed in the truth table below:

Table 1.7: Truth Table of RS Flip-Flop with NOR Gates

R	S	Q	Action
0	0	Last value	No change
0	1	1	Set
1	0	0	Reset
1	1	?	Forbidden

The first input condition in the table is R = 0, S = 0. Since a = 0 input has no effect on its output, the Flip-Flop retains its previous state. Hence Q remains unchanged.

The second input condition R = 0, S = 1 forces the output of NOR gate 2 low. This low output will reach NOR gate1 and when both inputs of NOR gates1 are low, its inputs Q will be high. Thus a 1 at the S input will SET the flip-flop and Q will be equal to 1.

The third input condition R = 1, S = 0 will force the output of NOR gate1 to low. This low will reach NOR gate2 and forces its outputs to high. Hence, when R = 1, S = 0, then Q = 0, = 1. Thus the flip-flop is RESET.

The last input condition in the table R = 1, S = 1 is forbidden since it forces both the NOR gates to the low state. It means both Q = 0, and = 0 at the same time, which violates the basic definition

of flip-flop that requires Q to be the complement of . Hence this input condition is forbidden and its output is unpredictable.

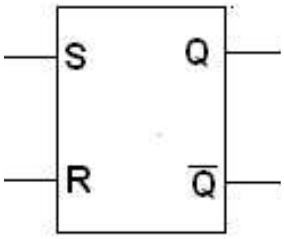
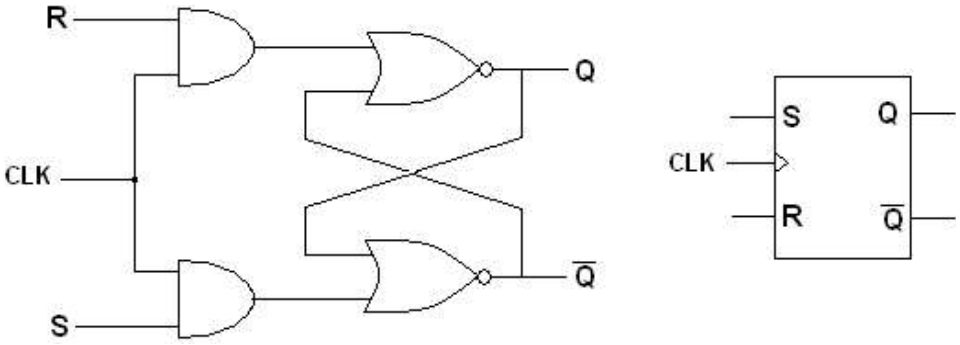


Figure 1.15: Symbol of RS Flip-Flop

CLOCK INPUT: For synchronization of operation of multiple flip-flop, an additional signal is added to all types of flip-flop. It is called clock signal, generally abbreviated as CLK. Addition of CLK signal ensures that whatever may be the input to the flip-flop, it affects the output only when CLK signal is given. Figure 1.16 shows a clocked RS flip-flop.



(a) (b)

Figure 1.16: (a) Clocked RS Flip-Flop
(b) Symbol of Clocked RS Flip-Flop

1.5.2 D Flip-Flop

D flip-flop is a modification of RS flip-flop. In RS flip-flop, when both the inputs are high, R = 1, S = 1 and the output becomes unpredictable and this input combination is termed as forbidden. To avoid this situation, the RS flip-flop is modified so that both the inputs cannot be same at a time. The modified flip-flop is called D flip-flop. Figure 1.17 shows a clocked D flip-flop.

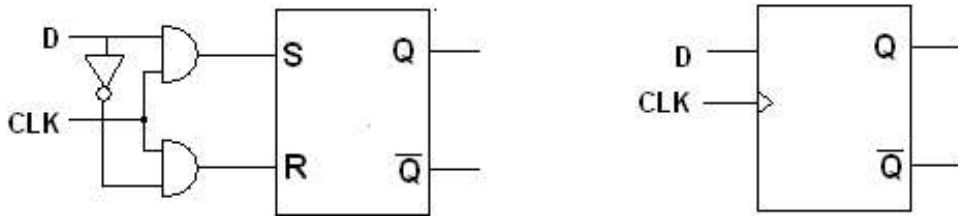


Figure 1.17: (a) Clocked D Flip-Flop; (b) Symbol of D Flip-Flop

In D flip-flop both inputs of RS flip-flip are combined together to make it one by a NOT gate so that inputs can not be same at a time. Hence, in D flip-flop there is only one input. The truth table is:

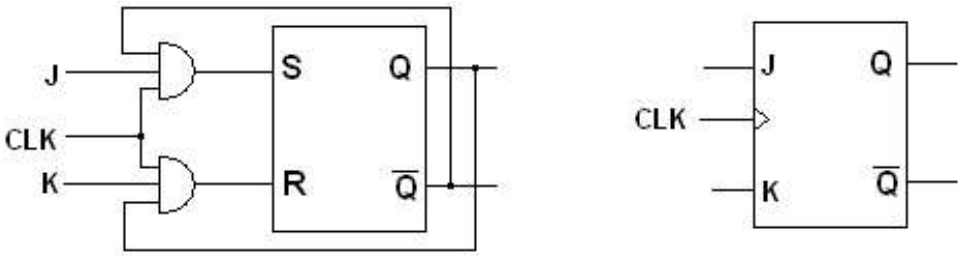
Table 1.8: Truth Table of D Flip-Flop

CLK	D	Q
0	X	Last state
1	0	0
1	1	1

In a clocked D flip-flop the value of D cannot reach the output Q when the clock pulse is low. During a low clock, both AND gates are disabled, therefore, D can change value without affecting the value of Q and this fact is expressed by putting an “X” as D input in the truth table 1.8. On the other hand, when the clock is high, both AND gates are enabled. In this situation, Q is forced to be equal to the value of D. In another way, we can say that in the D flip-flop above, Q follows the value of D while the clock is high. This kind of D flip-flop is often called a D latch.

1.5.3 JK Flip-Flop

In RS flip-flop, the input R = S = 1 is called forbidden as it causes an unpredictable output. In JK flip-flop this condition is used by changing the RS flip-flop in some way. In JK flip-flop both input can be high simultaneously and the corresponding toggle output makes the JK flip-flop a good choice to build counter- a circuit that counts the number of +ve or –ve clock edges. Figure 2.18 shows one way to build a JK flip-flop.



(a) (b)

Figure 1.18: (a) JK Flip-Flop, (b) Symbol of JK Flip-Flop

Table 1.9: Truth Table for JK Flip-Flop

CLK	J	K	Q
X	0	0	Last state
1	0	1	0
1	1	0	1
1	1	1	Toggle

The inputs J and K are called control inputs because their combinations decide what will be the output of JK flip-flop when a +ve clock pulse arrives. When J and K are both low, both the AND gates are disabled. Therefore, the CLK pulse has no effect. The first input combination of the truth table shows this and under this case the output Q retains its last state i.e. no change of state.

When J is low, K is high, the upper AND gate is disabled while the lower AND gate is enabled. Hence the flip-flop cannot be set; instead it is reset, i.e. Q = 0. This is shown by the second entry in the truth table.

When J is high, K is low the upper AND gate is enabled while the lower one is disabled. So the flip-flop is set there by making Q = 1.

When J and K are both high, then the flip-flop is set or reset depending on the previous value of Q. If Q is high previously, the lower AND gate sends a RESET trigger to the flip-flop on the next clock pulse. Then Q becomes equal to 0. On the other hand if Q is low previously, the upper AND gate sends a SET trigger on the flip-flop making Q = 1.

So, when J = K = 1, Q changes its value from 0 to 1 or 1 to 0 on the positive clock pulse. This changing of Q to or to Q is called toggle. Toggle means to change to the opposite state.

Any flip-flop may be driven by +ve as well as –ve clock. As such JK flip-flop can also be driven by positive clock as well as negative clock. Figure 1.19 shows symbol of positive clocked and negative clocked JK flip-flop.

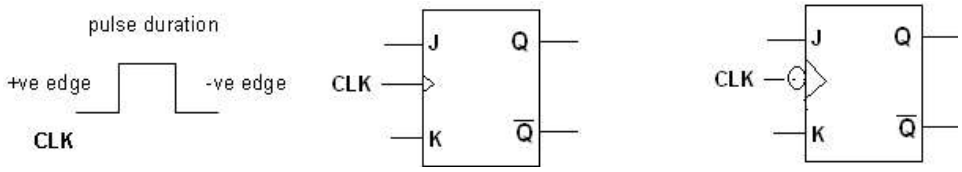


Figure 1.19: Positive and Negative clocked JK Flip-Flop

If a JK flip flop is driven by clock pulse duration, due to the feedback connection as shown in figure 2.18(a), the state of the flip flop i.e., value of Q, may be repeatedly change many times in a single clock pulse. So, it is highly desirable to make the flip flop sensitive to/ driven by pulse transition not pulse duration.

RC Differentiator Circuit: The clock pulse applied to a flip-flop is a square wave signal. Clock pulse is used to achieve synchronization of flip-flop operation. To make the synchronization more precise, the square wave pulse is further modified to make it a narrow spike by using a RC differentiator circuit as shown in Figure 1.20

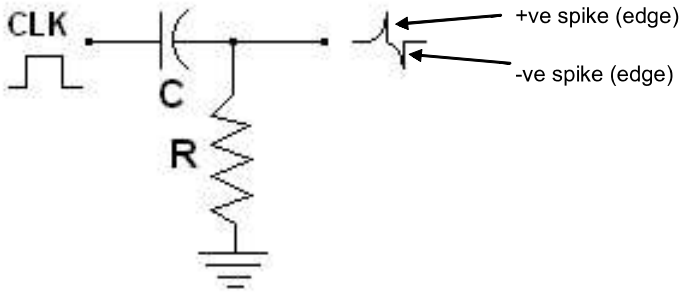


Figure 1.20: RC Differentiator Circuit

The upper tip of the differentiated pulse is called positive edge and the lower tip is called is negative edge. When a flip-flop is triggered by this type of narrow spike, it is called edge triggered flip-

flop. If the flip-flop is driven by +ve edge, it is called +ve edge triggered flip-flop. If it is a driven by negative edge, it is called negative edge triggered flip-flop.

Racing: In a flip-flop if the output toggles more than once during a clock edge then it is called racing. All flip-flop has a propagation delay, means the output changes its state after a certain time period from applying the input and the clock pulse. So, when a flip-flop is edge triggered, then due to propagation delay the output cannot affect the input again because, by that time, the edge of clock pulse has already passed away. If the propagation delay of a flip-flop is 20 ns and the width of the spike is less than 20 ns, then the returning Q and arrive too late to cause false triggering.

1.5.4 MS Flip-Flop

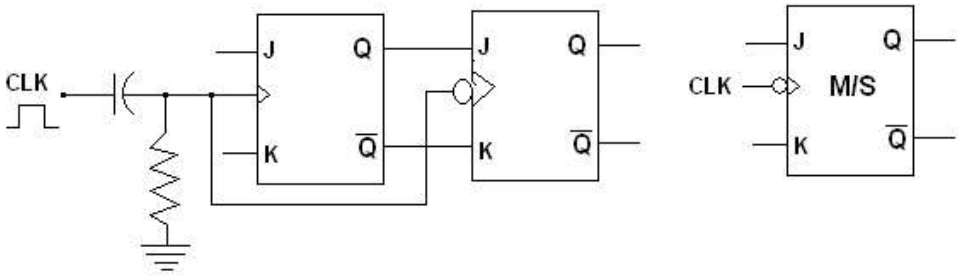
MS flip-flop is another way to avoid racing. Figure 1.21 shows one way to build MS flip-flop using two JK flip-flop, one of which is positive edge triggered and the other is negative edge triggered. The first JK flip-flop is master and the later is slave. The master responds to its J and K inputs at the positive edge. J = 1, K = 0, the master sets on the positive clock edge. The high Q output of the master drive the j input of the slave. So, at the negative clock edge, the slave also sets, copying the action of the master.

When j = 0, K = 1, master resets at +ve clock edge and the slave resets of the –ve clock edge.

When j = K = 1 master toggles at +ve clock edge, and the slave toggles at the –ve clock edge.

Hence whatever master does, the slave copies it.

MS flip-flop is a very popular flip-flop in industry due to its inherent resistance to racing. Hence to build counter it is extensively used.



(a) (b)

Figure 1.21: (a) Edge triggered JK MS Flip-Flop
(b) Symbol of JK MS Flip-Flop



CHECK YOUR PROGRESS

Q.4: Choose the correct option:

i) A flip-flop is basically a–

a) mono-stable multi-vibrator b) a stable multi-vibrator
c) bi-stable multi-vibrator d) none of these

ii) JK flip-flop has the specialty in–

a) fast response time b) toggle property
c) spike shaped clock input d) preset input

iii) In MS flip-flop the master changes state–

a) after the slave b) with the slave at the same time
c) before the slave d) never

2.6 COUNTER

A counter is one of the most useful sequential circuits in a digital system. A counter driven by a clock can be used to count a prescribed sequence of states upon the application of input pulses. This input pulses, which is also called count pulses, may be clock pulses or may originate from an external source and may occur at prescribed intervals of time or at random. The sequence of states that a counter goes may follow a binary sequence or may any other sequence of states. Counters are used for counting the number of occurrences of an event and are useful for generating timing sequences of an event and are useful for generating timing sequences to control operations in a digital system.

There are basically two types of counter— *Synchronous counter* and *asynchronous counter*.

Counters are constructed by using flip-flops and other logic gates. If the flip-flops are connected serially then the output of one flip-flop is applied as input to the next flip-flop. Thus, the clock pulse is applied only to the first flip-flop for triggering it and the other flip-flops are triggered by the output of the previous flip-flop. Therefore, this type of counter has a cumulative setting time due to propagation delay. Counter of this type are called serial or asynchronous counter. These counters have speed limitation.

Speed can be increased by using parallel or synchronous counter. Here, flip-flops are triggered by a clock at a time and thus the setting time is equal to the propagation delay of a single flip-flop. But this type of synchronous counters require more hardware and hence it is costly.

Combination of serial or parallel counter is also done to get an optimize solution of speed and hardware/cost. If each clock pulse advances the contents of the counter by one, it is called up counter. If the content of the counter goes down at each clock pulse, it is called down counter.

Before operation, some time it is required to reset all the flip-flops to zero. It is called “Clear”. Some time, it is required to set the flip-flops. It is called preset. To do these, two extra inputs are there in every flip-flop called CLR and PR.

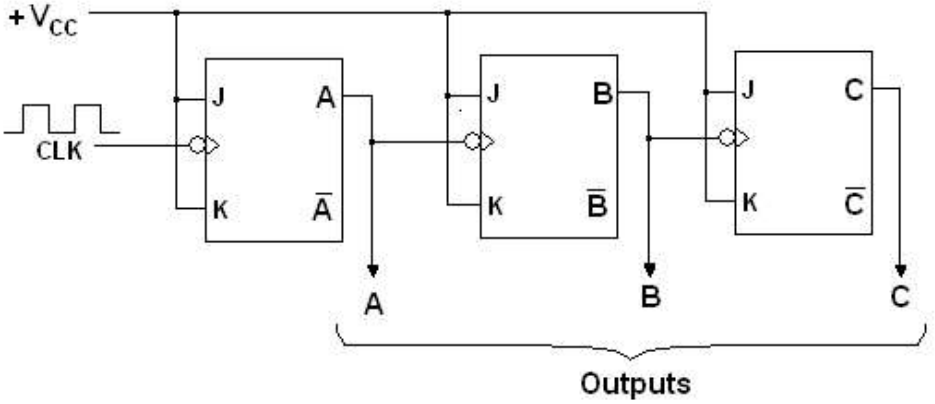
1.6.1 Asynchronous Counter

When the output of a flip-flop is used as the clock input for the next flip-flops it is called asynchronous counter.

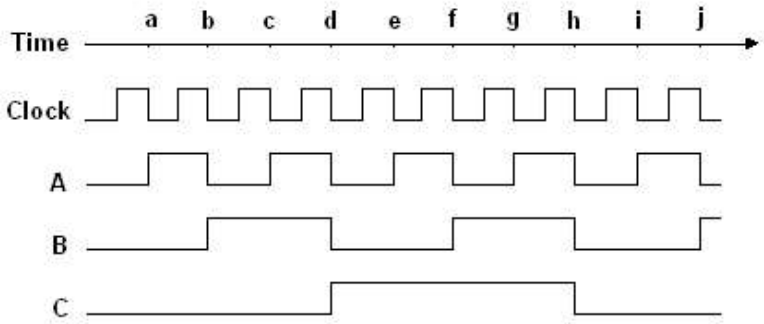
Asynchronous counters are also called ripple counter because flip-flop transitions ripple through from one flip-flop to the next in sequence until all flip-flop reach a new state.

A binary ripple counter can be constructed by using clocked JK flip-flop. Figure 1.22(a) shows three MS JK flip-flops connected in series. The clock drives flip-flop A. The output of A drives B and the output of B drives C. J and K inputs of all the flip-flops are connected to positive to make them equal to 1. under this condition each flip-flop will change state (toggle) with a negative transition at its clock point.

In the counter shown in Figure 1.22(a) the flip-flop A changes its state at the negative edges of the clock pulses. Its output is applied to the B flip-flop as its clock input.



(a) Three-bit Binary Counter



(b) Output Waveforms

Clock transitions	C	B	A
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
0	0	0	0

(c) Truth Table

Figure 1.22: Three Bit Binary Counter

The output of B flip-flop toggles at the negative edges of the output of A flip-flop. Similarly, output of B flip-flop is used as clock input of the C flip-flop and therefore C toggles at the negative edges of the output of B flip-flop. We can see that triggering pulses move through the flip-flops like a ripple in water.

The wave form of the ripple counter is shown in Figure 1.22(b). It shows the action of the counter as the clock runs. To understand the wave form let us assume that the counter is cleared before the operation. The A output is assumed as the list significant bit (LSB) and C is the most-significant-bit (MSB). Hence, at very beginning the contents of the counter is CBA = 000.

Flip-flop A changes its state to 1 after the negative pulse transition. Thus, at point a on the time line, A goes high. At point b, to it goes low, at c it goes back to high and so on.

Now, output of A acts as clock input of B. So, each time the output of A goes low, flip-flop B will toggle. Thus, at point b on the time line, B goes high, at point d it goes low, and toggles back high again at point f and so on.

Since B acts as the clock input for C, each time the output of B goes low, the C flip-flop toggles. Thus, C goes high at point d on the time line, it goes back to low again at point h.

We can see that the output wave form of A has half the frequency of the clock input wave. B has half the frequency than that of A and C has half the frequency than that of B.

We can further see that since the counter has 3 flip-flops cascaded together, it progresses through 000 – 001 – 010 – 011 – 100 – 101 – 110 – 111 as its CBA output. After CBA = 111, it starts the cycle again from CBA = 000. One cycle from 000 – 111 takes 8 clock pulses, as it is evident from the wave form as well as from the truth table.

1.6.2 Synchronous Counter

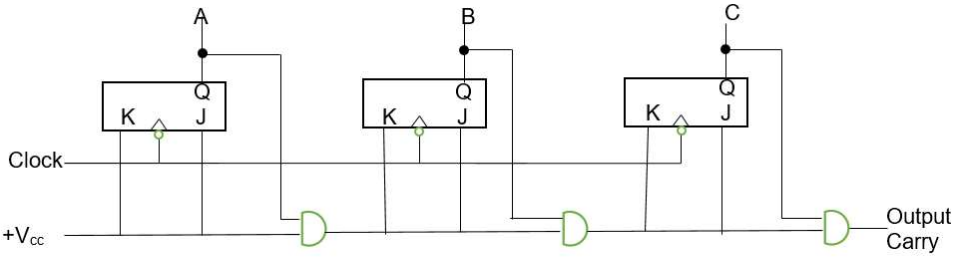
An asynchronous counter or ripple counter has limitation in its operating frequency. Each flip-flop has a delay time which is additive in asynchronous counter. In synchronous counter the delay of asynchronous counter is overcome by the use of simultaneous applications of clock pulse to all the flip-flops. Hence, in synchronous counter, the common clock pulse triggers all the flip-flops simultaneously and, therefore, the individual delay of flip-flop does not add together. This feature increases the speed of synchronous counter. The clock pulse applied can be counted by the output of the counter.

To build a synchronous counter, flip-flops and some additional logic gates are required. Figure 1.23 shows a three stage synchronous

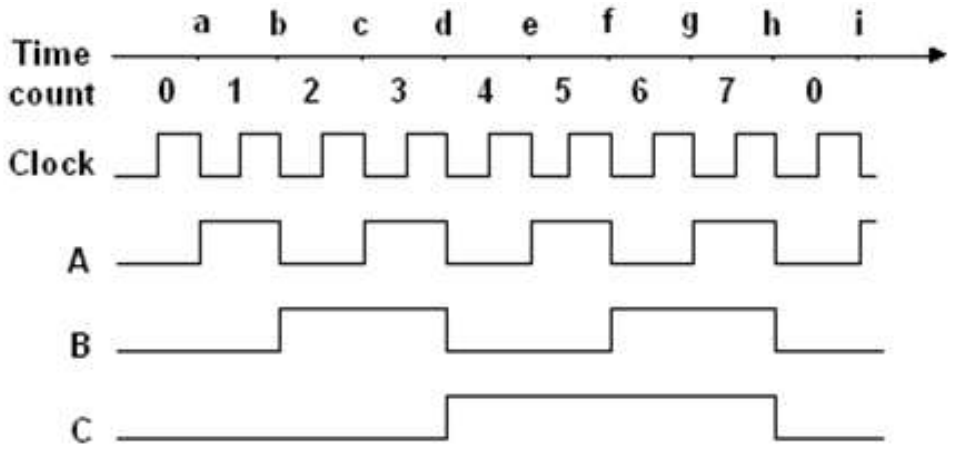
or parallel binary counter along with its output wave forms and truth table. Here the J and K inputs of each flip-flop is kept high and, therefore, the flip-flops toggle at the negative clock transition at its clock input. From figure 1.23, we can see that the output of A is AND operated with CLK to drive the 2nd flip-flop and the outputs of A, B are AND operated with CLK to drive the third flip-flop. This logic configuration is often referred to as "steering logic" since the clock pulses are steered to each individual flip-flop.

In the figure, the clock pulse is directly applied to all the first flip-flop. J and K inputs of the 1st flip-flop are both high, so the first flip-flop toggles state at the negative transition of the input clock pulses. This can be seen at points a, b, c, d, e, f, g, h, i on the time line.

The AND gate X is enabled when A is high, and it allows a clock pulse to reach the 2nd flip-flop. So, the 2nd flip-flop toggles with every other negative clock transition at points b, d, f and h on the time line.



a) 3-bit synchronous binary counter



b) Output waveform

Count	C	B	A
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
0	0	0	0

c) Truth table

Figure 1.23: Synchronous Counter

The AND gate Y is enabled only when both A and B are high and it transmits the clock pulses to the clock input of the 3rd flip-flop. The 3rd flip-flop toggles state with every fourth negative clock transition at d and h on the time line.

The wave form and the truth table show that the synchronous counter progresses upward in a natural binary sequence from 000 to 111. The total count from 000 to 111 is 8 and hence this counter can also be called MOD-8 counter, in count up mode.



CHECK YOUR PROGRESS

Q.5: State True or False:

a) Counters are non sequential digital circuits.

b) Asynchronous counters are faster in operation than synchronous counters.

c) The natural progression of a counter is called MODE.

d) Counters can be used to build digital clock.

1.7 REGISTER

A number of flip-flop connected to store binary number is called a Register. The number to be stored is entered or shifted into the register and also taken out or shifted out as per necessity. Hence, registers are also known as shift register.

Registers are used to store data temporarily. Registers can be used to perform some important arithmetic operations like complementation, multiplication, division etc. It can be connected to form counters, to convert serial data to parallel and parallel to serial data.

Types of Registers: According to shifting of binary number different types of registers are:

- Serial In– Serial Out (SISO)
- Serial In– Parallel Out (SIPO)
- Parallel In– Serial Out (PISO)
- Parallel In– Parallel Out (PIPO)

1.7.1 Serial In– Serial Out Register (SISO)

Figure 1.24 shows a typical 4-bit SISO register using flip-flops. Here the content of the register is named as QRST. Let us consider that all flip-flops are initially reset. Hence at the beginning QRST = 0000. Suppose a binary number 1011 which we want to store in the SISO register.

At time A: A 1 is applied at the D input at the first flip-flop. At the negative edge of the CLK pulse, this 1 is shifted into Q. The 0 of Q is shifted into R, 0 of R is shifted into S and 0 of S is shifted into T. The output of flip-flop just after time A is QRST = 1000.

At time B: Another 1 is applied in the data input of the first flip-flop. So at the negative CLK edge, this 1 is shifted to Q. The 1 of Q is shifted in R., 0 of R shifted in S, 0 of S is shifted into T. so, at the end of time B the output of all the flip-flops is QRST = 1100.

At time C: A 0(zero) is applied in the D input of the 1st first flip-flop. At the negative CLK edge, this 0 shifts to Q. The 1 of Q shifts into

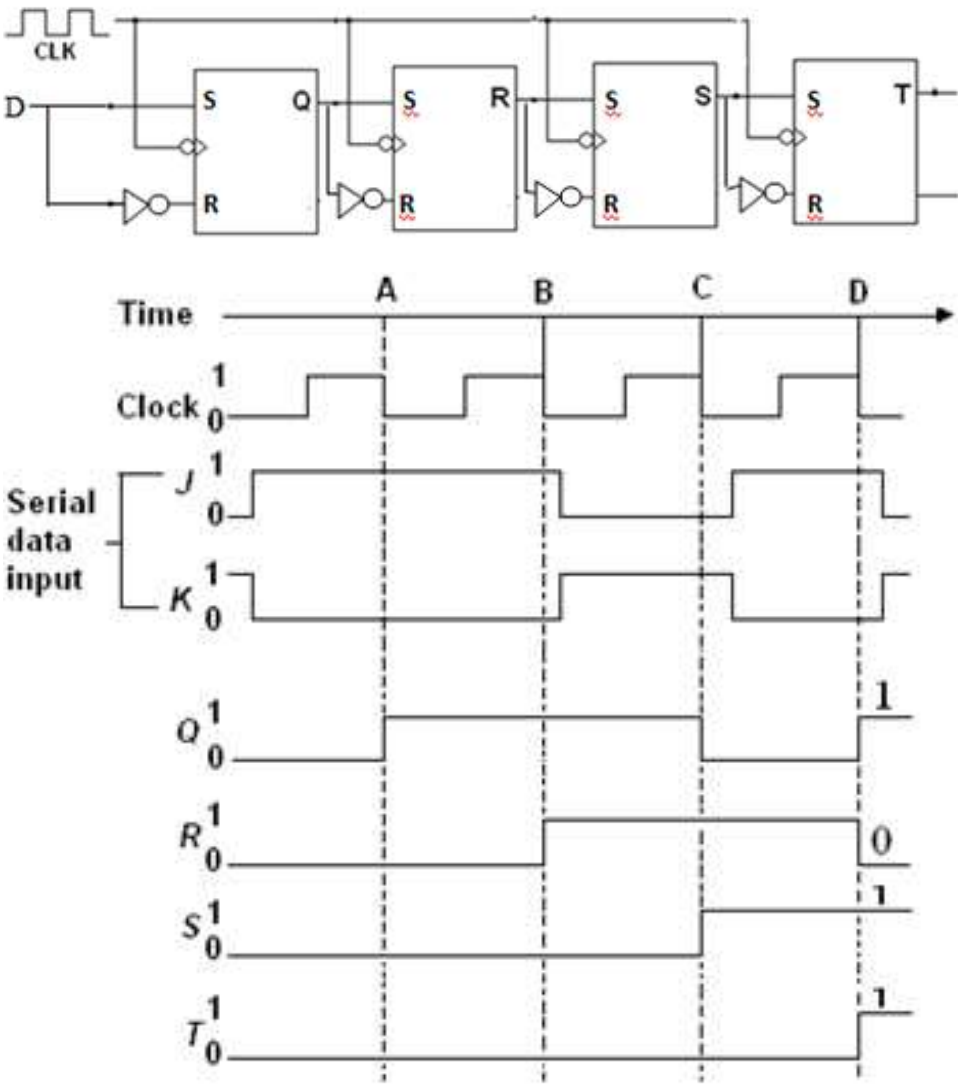


Figure 1.24: 4-Bit Serial In-Serial Out Shift Register

At time D: 1 is applied to D input of the first flip-flop. So this 1 shifts into Q at the negative transmission of CLK. The previous 0 of Q shifts into R, the 1 of R shifts into S, the 1 of S shifts into T. Hence at the end of time D, the registers contains QRST = 1011.

In the above steps, using 4 CLK pulses, we have shifted a 4 bit binary number 1011 in the register in a serial fashion.

To take out this binary number serially, we need another 4 CLK pulses and four 0 inputs into D pin of the first flip-flop. The binary number leave the register serially through the T pin of the last flip-flop.

2.7.2 Serial In– Parallel Out Register (SIPO)

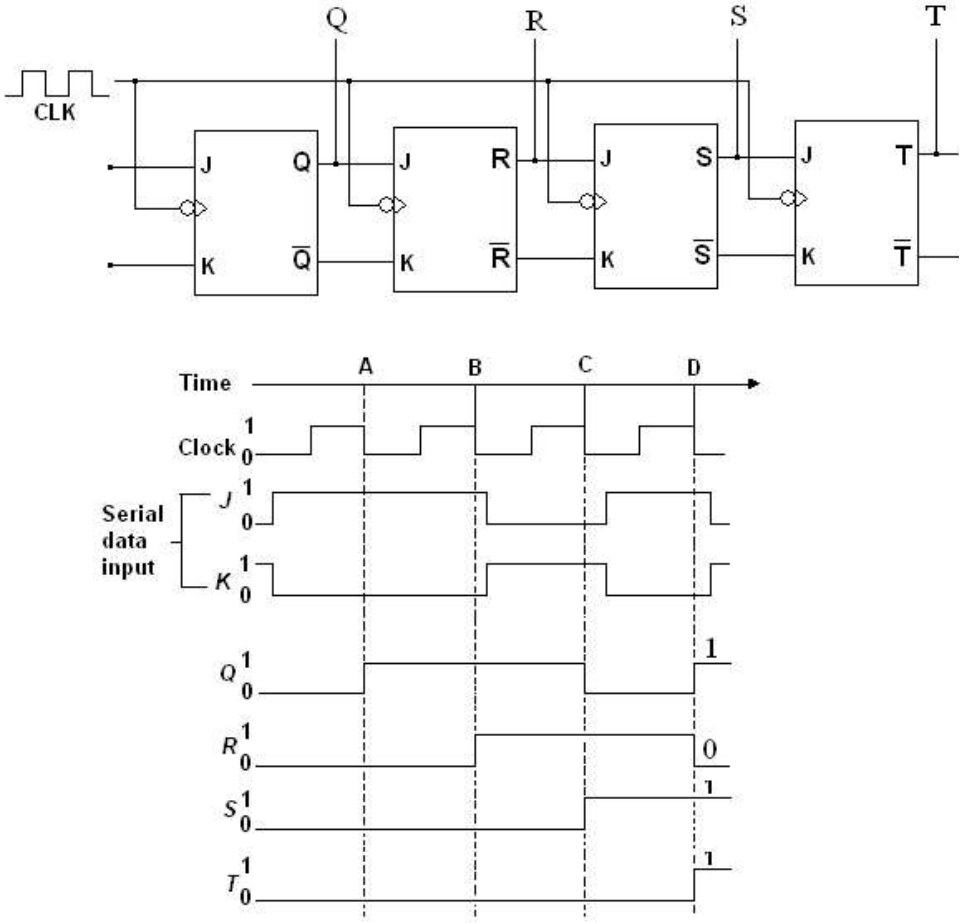


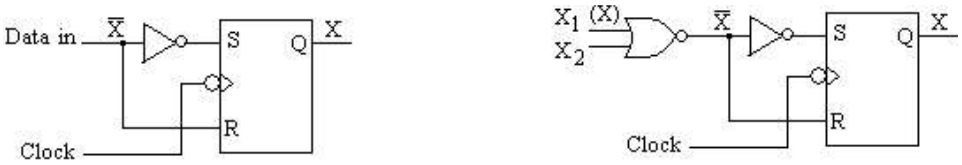
Figure 1.25: 4 Bit Serial In– Parallel Out Shift Register

In this type of shifts register, data are entered serially into the register and once data entry is completed it can be taken out parallely. To take the data parallely, it is simply required to have the output of each flip-flop to an output pin. All other constructional features are same as Serial In– Serial Out (SISO) register.

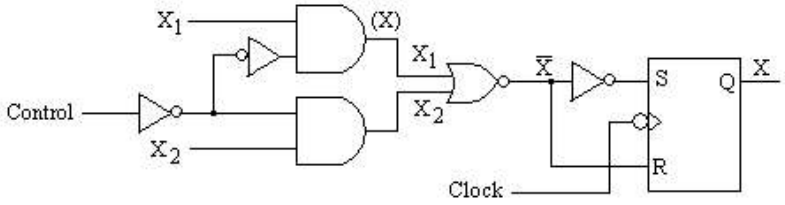
The shifting of data into SIPO is same as SISO registers. In the SIPO of Figure 1.25, a binary number, say, 1011 would be shifted just like the manner as described in the previous section. It would take 4 CLK pulses to complete the shifting. As soon as shifting is completed, the stored binary number becomes available in the output pins QRST. SIPO register is useful to convert serial data into parallel data.

1.7.3 Parallel In– Serial Out Register (PISO)

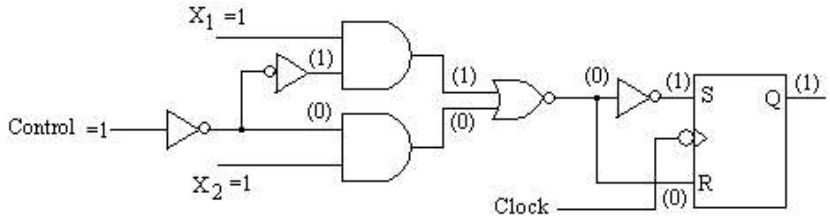
PISO register takes data parallel i.e., data are stored at a time and shifts data serially. Commercially available TTL IC for PISO is 54/74166. To understand the functional block diagram of 54/74166 we should first understand the following–



(a) D-Type Flip-Flop (b) D-Type Flip-Flop with NOR Gate



(c) D-Type Flip-Flop with Control Logic



(d) Example of Data Transmision

Figure 1.26: Building Block of Parallel In-Serial Out Register

Figure 1.26(a) is a clocked RS flip-flop, which is converted to D flip-flop by a NOT gate. The output of the flip-flop is 1 if Data IN ( ) is 0. Next add a NOR gate as in Figure 1.26(b). Here, if X_2 is at ground level, X_1 will be inverted by the NOR gate. As for example, if $X_1 = 1$, then output of the NOR gate will be = 0, thereby a 1 will be applied as S input into the flip-flop. This NOR gate allows entering data from two sources, either from X_1 or X_2. To shift X_1 into the flip-flop, X_2 is kept at ground level and to shift X_2 into the flip-flop, X_1 is kept at ground level. Here, ground level connection implies a 0(zero) input.

Now in Figure 1.26(c) two AND gates and two NOT gates are added. These will allow the selection of data X_1 or data X_2. If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus, the data X_1 will enter at the upper leg of the NOR gate and at the same time the lower leg of the NOR gate is kept at ground. Opposite to this, if the control is low, the upper AND gate is disabled and the lower AND gate is enabled. So X_2 will appear at the lower leg of the NOR gate and during this time the upper leg of the NOR leg gate is kept at ground level.

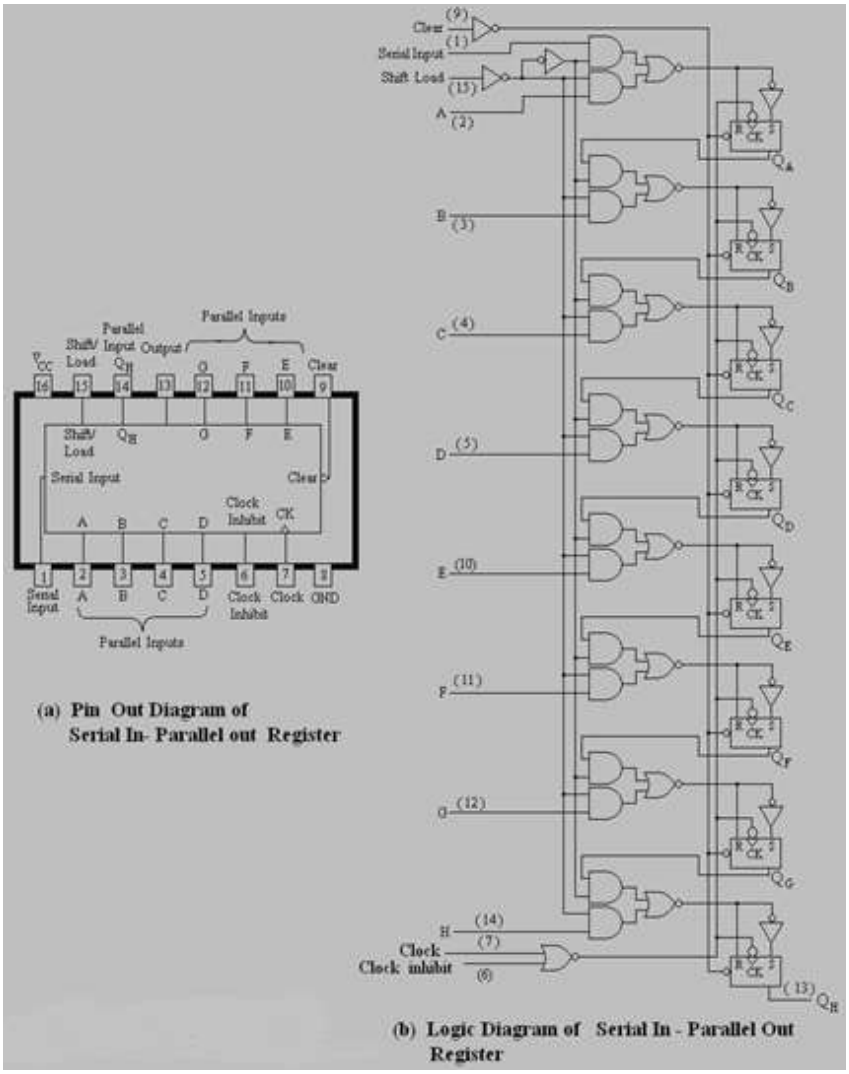


Figure 1.27: Circuit of 54/74166

(a) Pin Out Diagram of Serial In-Parallel Out Register

(b) Logic Diagram of Serial In-Parallel Out Register

If we study the Figure 1.27 of PISO we see that circuit of Figure 1.26(a) is repeated 8 times to form the 54/74/66 shift register. These 8 circuits are connected in such a style that it allows two operations: (1) The parallel data entry and (2) shifting of data serially through the flip-flop from Q_A toward Q_B.

If Figure 1.27 the X_2 input of Figure 1.26(c) is taken out from each flip-flop to form 8 inputs named as ABCDEFGH to enter 8 bit data parallely to the register. The control is named here as SHIFT/ LOAD which is kept low to load 8 bit data into the flip-flops with a single clock pulse parallely. If the SHIFT/LOAD is kept high it will enable the upper AND gate for each flip-flop. If any input is given to this upper AND gate then a clock pulse will shift a data bit from one flip-flop to the next flip-flop. That means data will be shifted serially.

1.7.4 Parallel In– Parallel Out Register (PIPO)

The register of Figure 1.27 can be converted to PIPO register simply by adding an output line from each flip-flop.

The 54/74198 is an 8 bits such PIPO and 54/7459A is a 4 bit PIPO register. Here the basic circuit is same as Figure 1.26(c). The parallel data outputs are simply taken out from the Q sides of each flip-flop. In Figure 1.28 the internal structure of 54/7459A is shown.

When the MODE CONTROL line is high, the data bits ABCD will be loaded into the register parallely at the negative clock pulse. At the same time the output is available $Q_AQ_BQ_CQ_D$. When the MODE CONTROL is low, then the left AND gate of the NOR gate is enabled. Under this situation, data can be entered to the register serially through SERIAL INPUT. In each negative transition, a data bit shifted serially from Q_A to Q_B, from Q_B to Q_C and so on. This operation is called right-shift operation.

With a little modification of the connection, the same circuit can be used for shift-left operation. To operate in shift-left mode, the input data is to be entered through D input pin. It is also necessary to connect Q_D to C, Q_C to B, Q_B to A as shown in Figure 1.28. MODE

CONTROL line is high to enter data through the D input pin and each stored data bits of flip-flops will be shifted to left flip-flop on each negative clock transition. This is serial data and left shift operation.

To clock inputs– clock1 and clock2 is used here to perform shift right and shift left operation..

Hence 54/7495A can be used as Parallel In– Parallel Out shift register as well as shift right and shift left register.

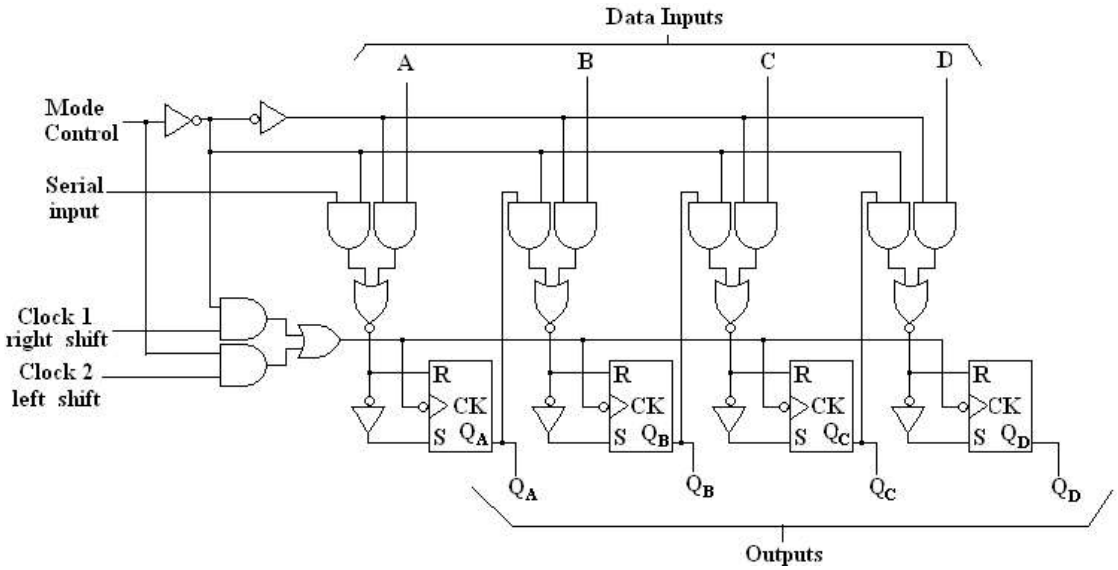


Figure 1.28: Parallel In– Parallel Out Shift Register



CHECK YOUR PROGRESS

Q.6: Choose the appropriate option:

i) Shift registers are–

a) basically a synchronous circuit

b) an asynchronous circuit

c) permanent memory

d) none of these

ii) In SIPO–

a) data enters parallely and leaves serially

b) data enters serially and leaves serially

c) data enters serially and leaves parallely



1.8 LET US SUM UP

- Digital circuits are of two categories– combinational and sequential.
- A combinational circuit is some combinations of logic gates as per specific relationship between inputs and outputs.
- Adder and subtractor circuits can perform binary addition and subtraction.
- A multiplexer is a combinational circuit which selects one of many inputs.
- Demultiplexer is opposite to a multiplexer.
- An encoder generates a binary code for 2^n input variables.
- A decoder decodes an information received from n-input lines and transmits the decoded information to maximum outputs.
- A sequential circuit's output depends on past output and present inputs.
- A flip-flop is basically a single cell of memory which can store either 1 or 0.
- Sequential circuits use flip-flop as their building block.
- There are many types of flip-flop viz RS, D, JK, MS flip-flop.
- A counter is a sequential circuit that goes through a prescribed sequence of states upon the application of input pulses. The Input pulse, which is called count pulses or may originate from an external source. There are two types of counters– asynchronous and synchronous counter.
- Shift registers are also sequential circuit which are used to store binary bits. They are of four different types– Serial In– Serial Out, Serial In– Parallel Out, Parallel In– Parallel Out and Parallel In– Serial Out register.



1.9 FURTHER READING

1) Leach, D. P. & Malvino, A. P. (1994). *Digital Principles and Applications*. Glencoe/McGraw-Hill.
2) Lee, S. C. (1976). Digital Circuits and Logic Design.
3) Mano, M. M. (2006). *Computer Systems Architecture*.
4) Mano, M. (1979). *Digital Logic, Computer Design*. Englewood Cliffs Prentice-Hall.

5) Talukdar, P. H. *Digital Techniques*. N. L. Publications.



1.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (b), ii) (c), iii) (a), iv) (b)

Ans. to Q. No. 2: i) (c), ii) (d), iii) (b)

Ans. to Q. No. 3: i) (a), ii) (b), iii) (a)

Ans. to Q. No. 4: i) (c), ii) (b), iii) (c)

Ans. to Q. No. 5: a) False, b) False, c) True, d) True

Ans. to Q. No. 6: i) (a), ii) (c)



1.11 MODEL QUESTIONS

Q.1: Distinguish between combinational circuit and sequential circuit.

Q.2: With truth table and logic diagram explain the working of a full-adder circuit.

Q.3: With truth table and logic diagram explain the working of a full-subtractor circuit.

Q.4: What is a multiplexer? With diagram explain the working of a 8-to-1 multiplexer.

Q.5: Explain the principle of an encoder. Draw a decimal-to-BCD encoder.

Q.6: What are the differences between asynchronous and synchronous counter? Draw a MODE-8 counter and explain its working principle.

Q.7: Draw logic diagram with output wave form of a 4-bit Serial In-Parallel Out shift register for an input of 1101. Explain its operation.

Q.8: Why is square wave clock pulse converted to a narrow spike to be used for flip-flops? Draw a RC differentiator circuit to convert a square wave into a narrow spike.

Q.9: What is racing? To get rid of racing what techniques are used?

Q.10: What is a magnitude comparator? Draw a block diagram and the function table of the magnitude comparator SN 7485.

*** ***** ***

UNIT 2: DATA REPRESENTATION

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 Representation of Characters in Computer

2.4 Positional Numbering Systems

- 2.4.1 Binary Number System
- 2.4.2 Decimal Number System
- 2.4.3 Octal Number System
- 2.4.4 Hexadecimal Number System

2.5 Computer Arithmetic

2.6 Binary Arithmetic

- 2.6.1 Binary Addition
- 2.6.2 Binary Subtraction
- 2.6.3 Binary Multiplication
- 2.6.4 Binary Division

2.7 Control and Data Path

- 2.7.1 Interaction between Control and Data Path

2.8 Central Processing Unit

2.9 Design of Arithmetic-Logic Unit

2.10 Control Unit

- 2.10.1 Hardware Control Unit
- 2.10.2 Microprogrammed Control Unit

2.11 Let Us Sum Up

2.12 Further Reading

2.13 Answers to Check Your Progress

2.14 Model Questions

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn to represent data in the computer system
- explain how different types of data stored and processed by digital computer system

- learn about different number system that are used in digital computer system
- learn computer arithmetic
- learn about control and data path, central processing unit, design of arithmetic and logic unit
- describe different categories of control unit.

2.2 INTRODUCTION

The basic functional units of digital computers are made of electronic circuits and it works on electrical signals namely digital and analog signal. We know analog signal is continuous in nature and digital is discrete in nature. The organization of any computer system depends considerably on how it represents numbers, characters, and control flow of information. Digital computer is an electronic device which can manipulate numbers and symbols under control of a set of instructions that is known as computer program. We should first understand the form in which data is stored in computer's memory before knowing how data is processed by a computer. This unit describes the various ways in which computers can store and manipulate numbers and characters as well as symbols. There are generally five types of data stored and processed by the digital computer system, which are respectively (i) Text, that consists of string of characters, (ii) Numbers or numerals, (iii) Audio data namely speech as well as music, (iv) Pictures or images may be monochrome or colour and (v) Video or moving picture data which is a sequence of pictures as well as audio data.

The basic unit of information in a digital computer is called a ***bit***, which is a contraction of binary digit. Generally, a bit is nothing but a "on" or "off" state (or "high" or "yes" **represents +5 volt** and "low" or "no" **represents 0 volt**) within a computer circuit. The binary number system as it has only two digits viz. 0 and 1 is also called a base two system. This binary number system is used in digital technology because all electrical and electronic circuits can be made to respond to the two-state concept. For example, a switch can be either opened or closed, only two possible states.

In 1964, the designers of the IBM System/360 mainframe computer established a convention of using groups of 8 bits as the basic unit for addressing computer memory. They called this collection of 8 bits a ***byte***. Computer ***words*** consist of two or more adjacent bytes that are sometimes addressed and almost always are manipulated collectively. The word size represents the data size that is handled most efficiently at a time by a particular architecture of the digital computer. Words can be 16 bits, 32 bits, 64 bits, or any other size that makes sense within the context of a computer's organization (including sizes that are not multiples of eight). Eight-bit bytes can be divided into two 4-bit halves called ***nibbles***. Because each bit of a byte has a value within a positional numbering system, the nibble containing the least-valued binary digit is called the low-order nibble, and the other half the high-order nibble.

2.3 REPRESENTATION OF CHARACTERS IN COMPUTER

We know that physical devices used to store and process data in computers are two-state devices, it may be either ON or OFF. The two states normally used are 1 and 0 also known as binary bits. Information processing using computers requires processing not only 26 capital English letters but also the 26 small English letters, 10 digits and around 32 other characters like punctuators, arithmetic operators, and parentheses as well as special characters. Coding of characters has been standardized to facilitate exchange of recorded data among computers. The most popular standard of coding scheme is known as ASCII (American Standard Code for Information Interchange), which uses 7 bits to code each of character. In this standard, codes are defined to convey information such as end of line, end of page, etc. to the computer system.

For example the ASCII code for A is 1000001 which is 65_{10} decimal equivalents.

In addition to **ASCII**, another important coding scheme known as **ISCII** (Indian Standard Code for Information Interchange) has been standardized by the Indian Standards Organization, which has 8 bits code

which allows English and Indian script alphabets to be used simultaneously. Recently, a new coding scheme for characters called **Unicode** has been standardized specifically to accommodate a large number of special symbols like Greek characters, mathematical symbols as well as non-English language characters. The first 128 codes of Unicode correspond to ASCII codes.

2.4 POSITIONAL NUMBERING SYSTEM

At some point during the middle of the sixteenth century, Europe embraced the decimal (or base 10) numbering system that the Arabs and Hindus had been using for nearly a millennium. Today, we take for granted that the number 243 means two hundreds, plus four tens, plus three units. Notwithstanding the fact that zero means “nothing,” virtually everyone knows that there is a substantial difference between having 1 of something and having 10 of something.

The general idea behind positional numbering systems is that a numeric value is represented through increasing powers of a **radix** (or **base**). This is often referred to as a weighted numbering system because each position is weighted by a power of the radix. The set of valid numerals for a positional numbering system is equal in size to the radix of that system. For example, there are 10 digits in the decimal system, 0 through 9, and 3 digits for the ternary (base 3) system, 0, 1, and 2. The largest valid number in a radix system is one smaller than the radix, so 8 is not a valid numeral in any radix system smaller than 9. To distinguish among numbers in different radices, we use the radix as a subscript, such as in 33_{10} to represent the decimal number 33. Any decimal integer can be expressed exactly in any other integral base system. The two most important radices in computer science are binary (base two), and hexadecimal (base 16). Another radix of interest is octal (base 8). The binary system uses only the digits 0 and 1; the octal system, 0 through 7. The hexadecimal system allows the digits 0 through 9 with A, B, C, D, E, and F being used to represent the numbers 10 through 15.

We can generalize the most common representation of any number system in a ûxed **base** using positional notation is as follows :

Choose an integral base b > 1. A positive integer A is represented by the length n and the digits a_i of its base b expansion can be expressed as

$$A = a_{n-1}b^{n-1} + \cdots + a_1 b + a_0,$$

where $0 \leq a_i \leq b - 1$, and a_{n-1} is sometimes assumed to be non-zero.

Since the base b is usually fixed in a given program, only the length n and the integers (a_i) $0 \leq i < n$ need to be stored. Some common choices for b are 2^{32} on a 32-bit computer, or 2^{64} on a 64-bit machine.

2.4.1 Binary Number System

A number system that uses only two digits namely 0 and 1 is called the binary number system. The base or radix of binary number is 2 (two). The two numbers 0 and 1 are also known as bits or binary digits. Any information in computer, we have to take help of these two digits or signals. These two digits or signals corresponds to two levels of electrical signals, and symbolically we represent them as 0 and 1.

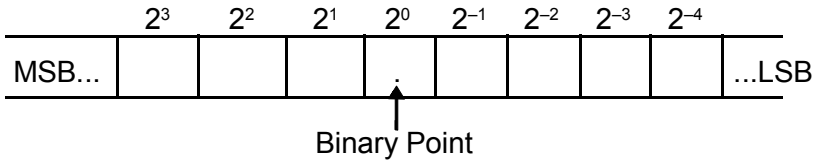


Figure 2.1: Binary Positional Values as a Power of its Base

From the above diagram it is clear that binary number system can be expressed as groups of number by two and its power of two. Here, the MSB is the **most significant bit,** the most left position and the LSB is the **least significant bit,** the most right position of the stream of binary bits.

For example, the binary number with four bits 1010.110_2 actually stands for the following representation:

$$1010.110 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}$$
$$= 8 + 0 + 2 + 0 + 0.5 + 0.25 + 0$$
$$= \mathbf{10.75}_{10}$$

In each binary digit, the value increases in powers of two starting with 0 to the MSB (left of the binary point) and decreases to the LSB (right of the binary point starting with power –1 of two (2).

2.4.2 Decimal Number System

In our day to day activities for arithmetic, we use the Decimal Number System. The decimal number system is said to be of base, or radix **10**, because it uses ten digits and the coefficients are multiplied by power of 10. It consists of ten numbers starting from 0, 1, 2, 3, 4, 5, 6, 7, 8 to 9. And we can represent any decimal number as follows:

$$2345.452_{10} = 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2} + 2 \times 10^{-3}$$

$$= 2000 + 300 + 40 + 5 + 0.4 + 0.05 + 0.002$$

$$= 2345.452$$

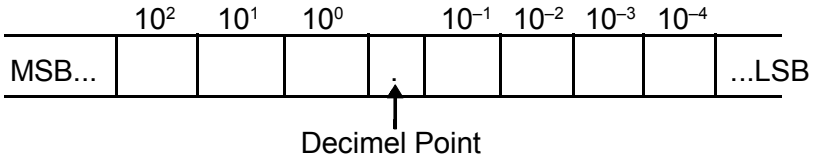


Figure 2.2: Positional Values in Decimal Numbers

2.4.3 Octal Number System

The octal number system is said to be of base, or radix 8, because it uses 8 digits and the coefficients are multiplied by power of 8. Eight digits used in octal system are: 0, 1, 2, 3, 4, 5, 6 and 7. The octal numbers are used extensively by early minicomputers. By grouping of 3 bits binary numbers can be represented by octal numbers and this can conveniently be used for entire data in the digital computer system.

In this system each number has a weight corresponding to its position as shown below:

$$a_{n-1}8^{n-1} + a_{n-2}8^{n-2} \cdots + a_1 8 + a_0 8^0 + a_{-1}8^{-1} + \cdots + a_{-n-1}8^{-n-1}$$

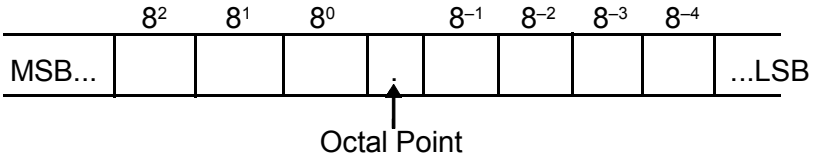


Figure 2.3: Positional Values in Octal Numbers

2.4.4 Hexadecimal Number System

The hexadecimal number system is said to be of base, or radix 16, because it uses 16 symbols and the coefficients are multiplied by power of 16. Sixteen digits used in hexadecimal system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Hexadecimal numbers are extensively used in microprocessor works. The hexadecimal number system is formed a binary number by grouping bits in group of 4 bits each, starting at the binary point.

In this system each number has a weight corresponding to its position as shown below:

$$a_{n-1}16^{n-1} + a_{n-2}16^{n-2} \cdots + a_1 16 + a_0 16^0 + a_{-1}16^{-1} + \cdots + a_{-n-1}16^{-n-1}$$

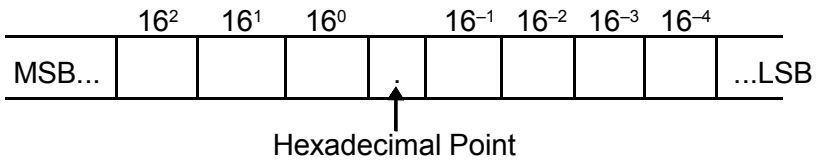


Figure 2.4: Positional Values in Hexadecimal Numbers

Following Table 2.1 describe the relation among all four number systems.

Hexadecimal	Decimal	Octal	Binary
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

2.5 COMPUTER ARITHMETIC

Data is manipulated by using the arithmetic instructions in digital computer systems. Data is also manipulated to produce required results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we have algorithms. In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations to get final resultant values as output from the computer system. And these instructions perform a great activity of computations in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

A processor has an arithmetic processor that executes arithmetic operations. The data, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point– binary numbers (i) signed magnitude, (ii) signed 1's complement or (iii) signed 2's complement. Most computers use the signed magnitude representation for the mantissa.

2.6 BINARY ARITHMETIC

In this section we will discuss how to add, subtract, multiply and divide binary numbers.

2.6.1 Binary Addition

When we add any two binary numbers, we write the numbers one below the others with their least significant bits (LSB) aligned. If the numbers have fractional parts, then the binary points must be aligned. In addition we start with the LSB. Depending upon the values of 'X' and 'Y' of the bits to be added, we will have a sum and may have a carry to the next stage of addition. There may be four possible combinations in which 'X' and 'Y' occur that can be cleared from the following table:

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2.6.2 Binary Subtraction

In order to subtract we use following subtraction table which is similar to addition:

X	Y	Difference (X-Y)	Carry
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Here we observe that when we have to subtract a '1' from a '0', we have to borrow a '1' from the adjacent bit position. Thus, when we are subtracting of bits belonging to the n^{th} stage, then we have to know whether a '1' has been borrowed from that stage by the $(n-1)^{th}$ stage.

2.6.3 Binary Multiplication

Multiplication is nothing but successive addition. That means multiplication of 4 by 3, is achieved by adding 4 to it 3 times which is the basic idea to implement binary multiplication.

Example:	Multiply 1101 by 1011
Multiplicand	1101
Multiplier	1011
Partial Product	1101
	+ 1101
	+ 0000
	+ 1101
Product	10001111

2.6.4 Binary Division

Division of two fixed-point binary numbers in signed magnitude representation is performed with paper and pencil by a process of successive compare, shift and subtract operations. Binary division is much simpler than decimal division because here the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor.

Now, we shift the divisor once to the right and subtract it from the dividend. The difference is known as a partial remainder because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial remainder. Comparing a partial remainder with the divisor continues the process. If the partial remainder is greater than or equal to the divisor, the quotient bit is equal to 1. The divisor is then shifted right and subtracted from the partial remainder. If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Obviously the result gives both a quotient and a remainder.



CHECK YOUR PROGRESS

Q.1: State whether the following statements are true (T) or false (F):

i) The most basic unit of information in a digital computer is called a byte.

ii) Analog signal is not continuous in nature.
iii) 1 nibble = 4 bits
iv) ISCII is a 7 bits coding system.
v) The radix (base) of Binary and Octal numbers are 10, 2.
vi) ASCII stands for American Standard Code for Information Interchange.
vii) Arithmetic processor executes arithmetic operations.

2.7 CONTROL AND DATAPATH

The major components of a computer system includes: (i) The CPU (ii) Memory (iii) Peripherals or I/O units that make the computer useful, such as the disk, printer, and mouse. As we know that all data to be processed by a computer has to be stored in memory, the characteristics of the interconnection paths are determined by the memory structure. In order to get good speed, all bits in a word are transmitted simultaneously.

A set of wires carries a group or stream of bits in parallel or serial has an associated control scheme is known as a *bus* or *path*. A bus carries a word to or from memory is known as a data bus. Its width will be equal to the word length in bits of the memory. In order to retrieve a word from memory, it is needed to specify its address. The address is carried by a Memory Address Bus (MAB) whose width equals the number of bits in the MAR of the memory. Thus if a computer's memory has 64 K, 32 bit words , then the data bus will be 32 bits wide and the address bus 16 bits wide. Moreover buses to carry address and data, it needs control signals between the various units of a computer. For example, control signals issued by the processor to memory to READ and WRITE and to I/O units to START or STOP a unit are carried by a control bus (CB).

A system bus will therefore consists of three type of buses or lines:

i) Data lines/ data bus,
ii) Address lines/ Address bus and
iii) Control lines/ control bus.

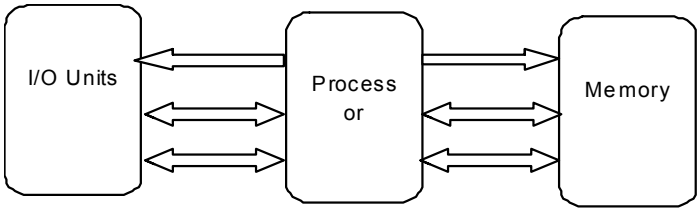


Figure 2.5: Interconnection of Computer units via two System Buses

Here in the above diagram, we are connecting I/O units to the computer is to connect them to the processor via a bus which is known as device address bus (DAB) that specifies the address of the I/O units to be accessed by the processor. A data bus (DB) carrying a word from the I/O unit to the processor or carrying a word from the processor to the I/O units as needed. Again a control bus (CB) carries commands such READ, WRITE, START, STOP etc. from the processor to I/O units. Also CB carries the I/O unit's status information to the processor. The I/O to processor system bus will thus encompass I/O device address lines, data lines and control lines. Above interconnection of I/O units, processor and memory using two independent system buses is known as a two bus interconnection structure. This interconnection will require the processor to supervise and participate in the transfer of data from and to I/O units. In general, all data is first taken to a processor register from the input unit and from there to the memory. Such data transfer is also known as program controlled transfer.

2.7.1 Interaction between Data Path and Control Unit

In this case control unit selects the microoperation to be performed. Also, CU determines the sequences of microinstructions based on control signals and status signals. On the other hand, Datapath are registers, ALU, Shifters,

Combinational circuits, different categories of buses to implement microoperations under control of the control unit (CU).

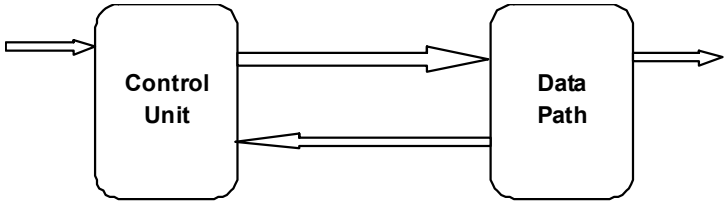


Figure 2.6: Interaction between Control Unit and Data Path

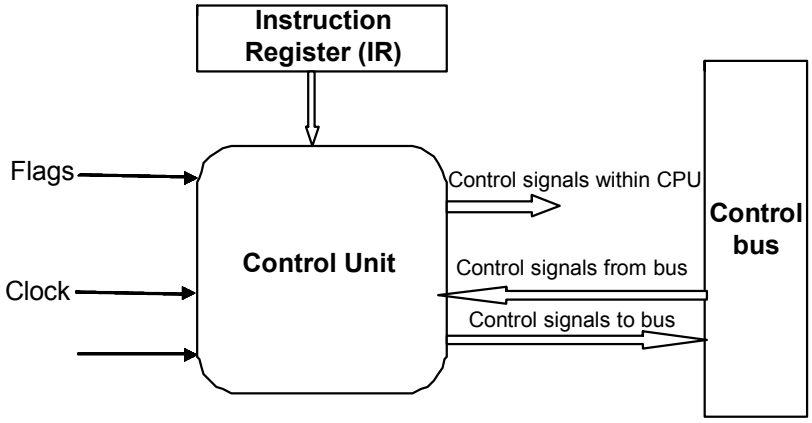


Figure 2.7: A Block Diagram of Control Unit

2.8 CENTRAL PROCESSING UNIT (CPU)

The CPU consists of a control unit (CU) and the arithmetic/logic unit (ALU). The major components of the CPU are: (i) A unit to load instructions from memory to the CPU, the arithmetic-logic unit (ALU) to process data. (ii) Registers to hold arguments and results of ALU processing, (iii) A control unit (CU) to decode instructions and initiate ALU action.

CPU is the main unit of computer, which is responsible to perform all the operations. The CPU of the IAS computer consists of a data processing unit and a program control unit. The data processing unit contains a high speed registers intended for temporary storage of instructions, memory addresses and data. The main action specified by instructions are performed by the arithmetic-logic circuits of the data processing unit. The control circuits in the program control unit are responsible for fetching instructions, decoding opcodes, controlling the information movements correctly through the system, and providing proper control signals for all CPU actions.

The operation or task that must perform by CPU at the time of execution of an instruction can be summarized as below:

- **Fetch Instruction:** The CPU reads an instruction from memory .
- **Interprets Instruction:** The instruction is decoded to determine what action is required.
- **Fetch Data:** The execution of an instruction may require reading data from memory or I/O module.

- **Process Data:** The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write Data:** The result of an execution may require writing data to memory or an I/O module.

To do these tasks, it should be clear that the CPU needs to store some data temporarily. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is being executed. In other words, the CPU needs a small internal memory. These storage locations are generally referred as registers.

The major components of the CPU are an arithmetic and logic unit (ALU) and a control unit (CU). The ALU does the actual computation or processing of data. The CU controls the movement of data and instruction into and out of the CPU and controls the operation of the ALU.

The ALU is connected to storage elements (registers for holding data to be processed and the results of the processing) by data buses. The ALU processor is normally composed of single element (one-bit) processors (primarily adders), which are assembled together to form a 32-bit processor, in the case of the MIPS R2000. It also includes some other processing elements (such as a shifter).

As mentioned earlier, the ALU needs other components as given below:

Registers to store arguments and results. *Buses* to carry data from registers to the ALU and results back to the register unit. Two memory *access units*, associated with buses. An *instruction fetch unit* to get instructions from computer memory as needed. This includes a *program counter*, which always points to the address of the next instruction to be accessed. A second path to memory to obtain data to be used in the program and to store data back into memory as required. A *control unit* that tells the ALU what to do.

For the ALU, we need some more components: (1) a register block (group of registers) in which to store operands and results, and (2) the ALU itself, which is primarily combinational logic, as mentioned earlier. Since, the ALU is largely an adder and other logic, we use the adder symbol.

In the MIPS CPU, all arguments used in processing are stored in registers. This requires two buses to route data to the ALU. All registers are tied to both output buses. A single input bus carries results back to all 32 registers.

2.9 DESIGN OF ALU (ARITHMETIC AND LOGIC UNIT)

ALU is responsible to perform the operation in the computer. The basic operations are implemented in hardware level. ALU is having collection of two types of operations: (i) Arithmetic operations. (ii) Logical Operations.

Consider an ALU having 4 arithmetic operations and 4 logical operations: To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups- arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.

Let us consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT and EX-OR.

We need three control lines to identify any one of these operations. The input combination of these control lines are shown below:

C_2 Control line is used to identify the group: logical or arithmetic, i.e., For arithmetic operation $C_2 = 0$ and for logical operation $C_2 = 1$

Control C_0 and C_1 lines and are used to identify any one of the four operations in a group. One possible combination is given below:

C_0	C_1	Arithmetic ($C_2 = 0$)	Logical ($C_2 = 1$)
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	EX-OR

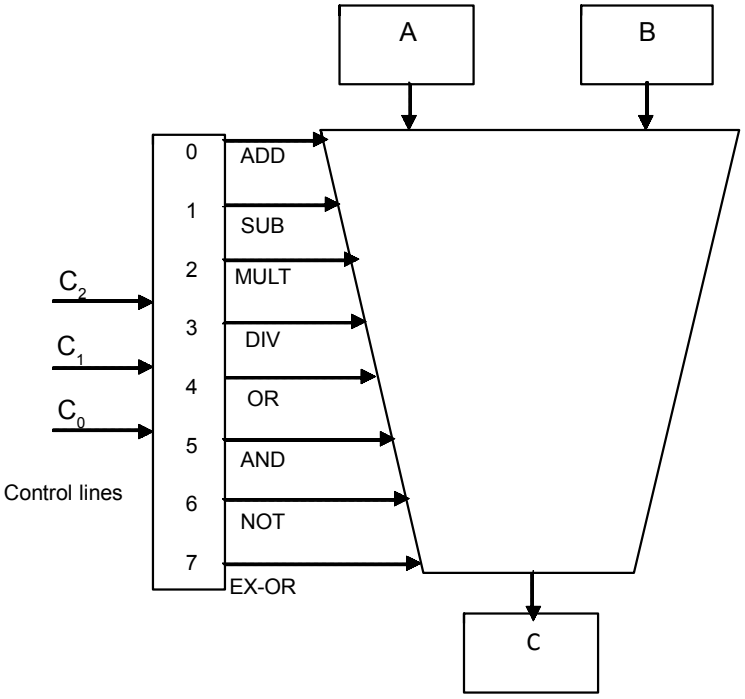


Figure 2.8: Block Diagram of the ALU

Here in the block diagram of the ALU has got two input registers named as A and B and one output storage register, named as C. It performs the operation as: C = A op B

The input data are stored in A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

For example, if the contents of controls lines are, 000, then the decoder enables the addition operation and it activates the adder circuit and the addition operation is performed on the data that are available in storage register A and B. After the completion of the operation, the result is stored in register C.

2.10 CONTROL UNIT DESIGN

Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory. It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.

Control unit can be designed by two techniques which are:

i) Hardwired Control Unit

ii) Microprogrammed Control Unit.

2.10.1 Hardwired Control Unit

It is implemented with the help of gates, flip flops, decoders etc. in the hardware. The inputs to control unit are the instruction register, flags, timing signals etc. This organization can be very complicated if we have to make the control unit large. Understand the design of hardwired control unit to generate all sequences of the control signals.

It is important to learn here, how an encoder outputs the control signals at each step and generates sequence by hardware from the start of count 0 to the end state and also encoder circuit outputs the control signals for given inputs from IR(Instruction Register), status flags and sequence counter in the hardwired control.

If the design has to be modified or changed, all the combinational circuits have to be modified which is a very difficult task.

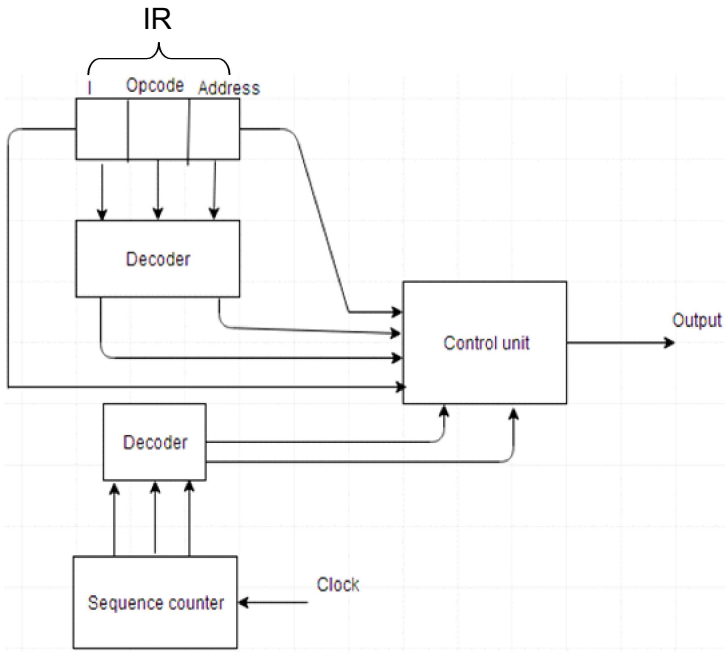


Figure 2.9: Simple Design of Hardwired Control Unit

2.10.2 Microprogrammed Control Unit

It is implemented by using programming approach. A sequence of micro operations is carried out by executing a micro program consisting of micro-instructions. In this organization any modifications or changes can be done by updating the micro program in the control memory by the programmer. In this method, it is important to understand the design of microprogrammed control unit to generate all sequences of the control signals using a control memory. Here important issues are we should learn how the control memory stores at successive addresses the data for generating the control signal values 0 or 1 stored at each address and generates the sequences by address changes from the start address to end address and also how the start address for generating the control signals depend on the instruction at IR (Instruction Register). Each instruction specifies a sequence of microoperations that are used to implement the instruction. Each instruction translated into a short program of microinstructions by the hardware; similar to the way a compiler translates each instruction in a high-level language program into a sequence of assembly-language instructions.

Difference between Hardwired Control and Microprogrammed Control Unit:

Hardwired Control Unit	Microprogrammed Control Unit
Here technology is circuit (hardware) based.	Here technology is software (programs) based.
It is implemented through flip-flops, gates, decoders etc.	In this case, microinstructions generate signals to control the execution of instructions.
It is used fixed instruction format.	Here variable instruction format (16-64 bits per instruction) are used.
Here instructions are register based.	Here instructions are not register based.
ROM is not used in this unit	ROM is used in this unit

It is used in RISC processor	It is used in CISC processor
Faster decoding.	Slower decoding.
It is very difficult to modify.	It can be easily modified.
Chip area is less.	Chip area is large.



CHECK YOUR PROGRESS

Q.2: Fill up the blanks:

i) A set of wires carries a group or stream of bits is called .....

ii) A data bus (DB) carrying ..... from the I/O unit to the processor.

iii) CU determines the sequences of .....

iv) The major components of the CPU are .....

v) I/O units to the computer is to connect them to the processor via a bus which is known as .....

vi) Control unit generates ..... and ..... for the operations of the computer.

vii) Hardwired CU is used in ..... processor.

viii) Microprogrammed CU technology is ..... based.



2.11 LET US SUM UP

- Digital computer is an electronic device which can manipulate numbers and symbols under control of a set of instructions that is known as computer program.
- There are generally five types of data stored and processed by the digital computer system, which are respectively (i) Text that consists of string of characters, (ii) Numbers or numerals, (iii) Audio data namely speech as well as music, (iv) Pictures or images may be monochrome or colour and (v) Video or moving picture data.

- The most basic unit of information in a digital computer is called a bit, which is a contraction of binary digit. In the short cut, a bit is nothing more than a state of “on” or “off” (or “high” or “Yes” **represents 5 volt** and “low” or “no” **represents 0 volt**) within a computer circuit.
- The general idea behind positional numbering systems is that a numeric value is represented through increasing powers of a **radix** (or **base**). This is often referred to as a weighted numbering system because each position is weighted by a power of the radix.
- A number system that uses only two digits namely 0 and 1 is called the binary number system. The base or radix of binary number is 2.
- The decimal number system is said to be of base, or radix **10**, because it uses ten digits and the coefficients are multiplied by power of 10.
- The octal number system is said to be of base, or radix 8, because it uses 8 digits and the coefficients are multiplied by power of 8. Eight digits used in octal system are: 0, 1, 2, 3, 4, 5, 6 and 7.
- Sixteen digits used in hexadecimal system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- The major components of a computer system includes: (i) The CPU (ii) Memory (iii) Peripherals or I/O units that make the computer useful, such as the disk, printer, and mouse.
- The CPU consists of a control unit (CU) and the arithmetic/logic unit (ALU). The major components of the CPU are: (i) A unit to load instructions from memory to the CPU, The arithmetic-logic unit (ALU) or datapath, to process data. (ii) Registers to hold arguments and results of ALU processing, (iii) A control unit (CU) to decode instructions and initiate ALU action.
- Datapath are registers, ALU, Shifters, Combinational circuits, different categories of buses to implement microoperations under control of the control unit (CU).
- A control bus (CB) carries commands such READ, WRITE, START, STOP etc. from the processor to I/O units. Also CB carries the I/O unit’s status information to the processor.

- ALU is responsible to perform the operation in the computer. The basic operations are implemented in hardware level. ALU is having collection of two types of operations: (i) Arithmetic operations. (ii) Logical Operations.
- Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory.
- Control unit can be designed by two techniques which are: (i) Hardwired Control Unit, (ii) Microprogrammed Control Unit.
- Hardwired CU is implemented with the help of gates, flip flops, decoders etc. in the hardware. The inputs to control unit are the instruction register, flags, timing signals etc.
- It important to understand the design of microprogrammed control unit to generate all sequences of the control signals using a control memory.



2.12 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI.
2) Hammacher, Carl. (Fifth Edition) (International Edition, 2002). *Computer Organization*. McGrawHill.
3) Rajaraman, V. (Fouth Edition, 2008). *Fundamentals of Computers*.
4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.



2.13 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) False, ii) False, iii) True, iv) False, v) False, vi) True, vii) True

Ans. to Q. No. 2: i) Bus, ii) words, iii) microinstructions, iv) CPU, Memory and I/O units, v) Device Address Bus (DAB), vi) Timing and control signals, vii) RISC and viii) Software



2.14 MODEL QUESTIONS

Q.1: Explain, how is a character represent in digital computer system?

Q.2: What are different types of data stored and processed by the digital computer system?

Q.3: What is positional number system in computer system? Explain its different categories of number systems.

Q.4: Explain the mechanism of Control unit and data path.

Q.5: What is a bus? Explain how various buses interconnect different component of a computer system.

Q.6: Explain different functions of CPU in digital computer system.

Q.7: What is ALU? How it works in computer system?

Q.8: What do you mean by control unit design? Explain its different categories.

*** ***** ***

UNIT 3: COMPUTER ARITHMETIC

UNIT STRUCTURE

3.1 Learning Objectives

3.2 Introduction

3.3 Integer Representation

- 3.3.1 Signed Integer Representation
- 3.3.2 Fixed Point Representation
- 3.3.3 Floating Point Representation

3.4 Signed 0 Magnitude Form

3.5 The Concept of Complement

- 3.5.1 Diminished Radix Complement

3.6 One's Complement

- 3.6.1 One's Complement Arithmetic

3.7 Two's Complement Arithmetic

3.8 Let Us Sum Up

3.9 Further Reading

3.10 Answers to Check Your Progress

3.11 Model Questions

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about integer both signed and unsigned representation
- learn how to represent both fixed point and floating point
- describe signed and magnitude form of integers.
- learn the concept of complements
- learn how to find 1's complement of binary numbers
- calculate 1's complement arithmetic
- learn how to find 2's complement and its arithmetic.

3.2 INTRODUCTION

Computer arithmetic is a branch of computer engineering that deals with methods of representing integers and real values in digital systems and

efficient algorithms for manipulating such numbers by means of hardware circuits or software routines. On the hardware side, various types of adders, subtractors, multipliers, dividers, square-rooters, and circuit techniques for function evaluation are considered. Both abstract structures and technology-specific designs are dealt with. Software aspects of computer arithmetic include complexity, error characteristics, stability, and certifiability of computational algorithms.

Computer arithmetic is a subfield of digital computer organization that deals with the hardware realization of arithmetic functions which has a major thrust to design of hardware algorithms and circuits to enhance the speed of numeric operations. It encompasses the study of number representation, algorithms for operations on numbers, implementations of arithmetic units in hardware, and their use. The basic application of computer arithmetic is in the general purpose systems for fast primitive operations for processor data paths. On the other hand, it has also applications on special purpose systems which are given below:

i) Signal and image processing
ii) Real-time 3D graphics
iii) HDTV, image compression
iv) Network processors (data compression, encryption/decryption)

In this unit we will learn about basic numeric operations. Here we will see how two integer numbers are represented in different format and do arithmetic operations on both unsigned and signed integers that a digital computer system understand.

3.3 INTEGER REPRESENTATION

There are basically two types of integer representation in digital computer system. They are:

i) Unsigned integer representation.
ii) Signed integer representation.

We have already learn how to convert an unsigned integer from one base to another. For example, 25_{10} decimal equivalent to 11001_2 binary numbers. So, for unsigned integer, it is very simple and easy to convert in

digital computer system. But for signed integer its needs some additional calculations.

Signed numbers require additional issues to be addressed. When an integer variable is declared in a program, many programming languages automatically allocate a storage area that includes a sign as the first bit of the storage location. By convention, a "1" in the high-order bit indicates a negative number. The storage location can be as small as an 8-bits (byte) or as large as several words, depending on the programming language and the computer system. The remaining bits (after the sign bit) are used to represent the number itself. How this number is represented depends on the method used.

In a computer system that uses signed-magnitude representation and 8 bits to store integers, 7 bits can be used for the actual representation of the magnitude of the number. This means that the largest integer an 8-bit word can represent is $2^7 – 1$ or 127 (a zero in the high-order bit, followed by 7 ones). The smallest integer is 8 ones, or –127. Therefore, N bits can represent $–2^{(N–1)} – 1$ to $+2^{(N–1)} – 1$.

We know that for n-bit number, the range for natural number is from 0 to $2^n –1$. There are three different schemes or methods to represent signed (negative) and unsigned integer. These are

i) Signed and Magnitude form.
ii) 1's complement form.
iii) 2's complement form

3.3.1 Signed Integer Representation

We know that for n-bit number, the range for natural number is from 0 to $2^n–1$.

For n-bit, we have all together different combination, and we use these different combination to represent numbers, which ranges from 0 to $2^n–1$.

If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.

For n-bit representations, the range is from:

$$-2^{n-1} - 1 \text{ to } + 2^{n-1} - 1$$

For example, if we consider 8-bit number, then range for natural number is from 0 to 255.

But for signed integer the range is from –127 to +127.

3.3.2 Fixed-Point Representation

Let us take an example to show and make clearly how binary numbers is represented in fixed-point.

We know that binary representation of 41.6875 is 101001.1011.

To store this number, we have to store two part of information, the part before decimal point and the part after decimal point. This is known as *fixed-point representation* where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.

If we use 16 bits before decimal point and 8 bits after decimal point, in signed magnitude form, the range is from $-2^{16} - 1$ to $+ 2^{16} - 1$ and precision is $2^{(-8)}$. One bit is required for sign information, so the total size of the number is 25 bits.

[1(sign) + 16 (before decimal point) + 8 (after decimal point)]

1 bit (Sign bit)	16 bits (before decimal point)	8 bits (after decimal point)

Figure 3.1: Fixed-Point Representation

3.3.3 Floating-Point Representation

In digital computers, floating-point numbers consist of three parts: (i) a Sign bit, (ii) an Exponent part (representing the exponent on a power of 2), and (iii) a fractional part called a Significant (which is a fancy word for a Mantissa). For example a floating point representation is given below which a 14 bits model consists of 1 bit as sign bit, 5 bits as Exponent part and 8 bits as Mantissa part.

1 bit (Sign bit)	5 bits (Exponent)	8 bits (Mantissa)

Figure 3.2: Floating-Point Representation

In this representation, numbers are represented by a Mantissa comprising the significant digits and an Exponent part of Radix R.

The format is: $Mantissa * R^{Exponent}$

Numbers are often normalized, such that the decimal point is placed to the right of the first non zero digit.

For example, the decimal number, 5236 is equivalent to $5.236*10^3$

To store this number in floating point representation, we store 5236 in Mantissa part and 3 in Exponent part.

3.4 SIGNED MAGNITUDE FORM

In signed magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.

For an n-bit number, one bit is used to indicate the signed information and remaining (n–1) bits are used to represent the magnitude. Therefore, the range is from $–2^{n–1}–1$ to $+ 2^{n–1}–1$.

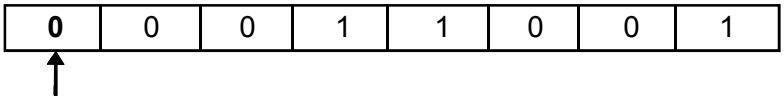
Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

For example, 01011001 represents +169

but 11011001 represents –169

For example, if in a computer, the word size is 1 byte (8 bits), then, +25 will be represented as follows:

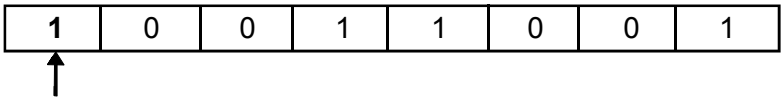
Number in binary notation as binary equivalent of 25_{10} is 11001



MSB (0 for + sign)

–25 will be represented as follows:

Number in binary notation as binary equivalent of 25_{10} is 11001



MSB (1 for - sign)

3.5 THE CONCEPT OF COMPLEMENT

Complement Systems Number theorists have known for hundreds of years that one decimal number can be subtracted from another by adding the difference of the subtrahend from all nines and adding back a carry. This is called taking the nine's complement of the subtrahend, or more formally, finding the diminished radix complement of the subtrahend. The advantage that complement systems give us over signed magnitude is that there is no need to process sign bits separately, but we can still easily check the sign of a number by looking at its high-order bit.

The concept of complements is used to represent signed number.

Let us consider a number system of base-r or radix-r. There are two types of complements.

i) The radix complement or the r's complement.

ii) The diminished radix complement or the (r–1)'s complement.

3.5.1 Diminished Radix Complement

Given a number N in base r having n digits, the (r–1)'s complement of N is defined as $(r^n – 1) – N$.

For decimal numbers, r = 10 and r – 1 = 9, so the 9's complement of N is $(10^n – 1) – N$

For example, 9's complement of 5642 is: $(10^4 – 1) – 5642 = 9999 – 5642 = 4357$

3.6 ONE'S (1'S) COMPLEMENT

It is important to note at this point that although we can find the nine's complement of any decimal number or the one's complement of any binary number, we are most interested in using complement notation to represent negative numbers. We know that performing a subtraction, such as 10–7, can be also be thought of as "adding the opposite'', as in 10 + (–7). Complement notation allows us to simplify subtraction by turning it into addition, but it also gives us a method to represent negative numbers. Because we do not wish to use a special bit to represent the sign (as we did in signed-

magnitude representation), we need to remember that if a number is negative, we should convert it to its complement. The result should have a 1 in the leftmost bit position to indicate the number is negative. If the number is positive, we do not have to convert it to its complement. All positive numbers should have a zero in the leftmost bit position.

3.6.1 One's (1's) Complement Arithmetic

The one's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.

For example, 1's complement of binary number 100010 will be 011101 and for 011011 will be 100100.

Example 3.1: Express 23_{10} and -9_{10} in 8-bit binary one's complement form.

$23_{10} = +(00010111)_2 = 00010111_2$

$-9_{10} = -(00001001)_2 = 11110110_2$

a) If Subtranhend is smaller than the Minuend

In this case we follow the following three simple steps:

Step 1: Find one's complement of the subtrahend.

Step 2: Proceed as in addition

Step 3: Ignored the carry and add 1 to the total end-around-carry.

Example 3.2: Add 23_{10} to -9_{10} using one's complement arithmetic.

Or, Subtract 9 from 23.

Solution:	0 0 0 1 0 1 1 1	(+23)
	+ 1 1 1 1 0 1 1 0	+(–9)
The last carry is added:	0 0 0 0 1 1 0 1	
	+1	
Sum is:	0 0 0 0 1 1 1 0	$+(14)_{10}$

b) If Subtranhend is larger than the minuend

In this case we follow the following three simple steps:

Step 1: Find one's complement of the subtrahend.

Step 2: Proceed as in addition.

Step 3: Complement the result and place a negative sign in front of the result.

Example 3.3: Add 9_{10} to -23_{10} using one's complement arithmetic.

Or, subtract 23 from 9

Solution:

	0 0 0 0 1 0 0 1	(+9)
	+1 1 1 0 1 0 0 0	+(–23)
No carry	1 1 1 1 0 0 0 1	

Now, complementing the result and putting a –ve sign we get,

0 0 0 0 1 1 1 0 = -14_{10}

The primary disadvantage of one's complement is that we still have two representations for zero: 00000000 and 11111111 i.e., +ve 0 and –ve 0. For this and other reasons, computer engineers long ago stopped using one's complement in favor of the more efficient two's complement representation for binary numbers.

Note that total numbers that can be represented using 1's complement are 2^N-1, where N is the word size (number of bits in a word).

3.7 TWO'S (2'S) COMPLEMENT ARITHMETIC

We know that all negative numbers can be represented in 2's complement form. Subtraction between two numbers can be achieved by adding the 2's complement of the number. Two's complement method represents positive numbers in their true form that means their binary equivalents and negative numbers in 2's complement form.

Now we will discuss how to represent of signed numbers in 2's complement form.

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the addition between these numbers we have:

	0	1	0	1	1	1	0	0
	1	0	1	0	0	0	1	1
1	0	0	0	0	0	0	0	0

Since we are considering an eight bit number, so the 9th bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

The 2's complement form of any number is obtained by simply adding 1 to LSB of the 1's complement of that number.

For example 2's complement of 101110 is calculated as follows:

First find the 1's complement of 101110, which is 010001.

Now add 1 with LSB of 010001 is 010010, which is the required 2's complement of 101110.

For example,

i) Using 2's complement representation subtract 20 from 25

Solution: We have 25_{10} = 1 1 0 0 1

Ignore the carry -20_{10} = 0 1 1 0 0 (2's Complement of 20_{10})

1 0 0 1 0 1

So, answer is +5

ii) Add (–20) and (–25) using Using 2's complement representation

Solution: We have -25_{10} = 1 0 0 1 1 1 (2's Complement of 25_{10})

-20_{10} = 1 0 1 1 0 0 (2's Complement of 20_{10})

Ignore carry 1 0 1 0 0 1 1

and the result is –(2's complement of 010011)

So, answer is -101101 = - 45

iii) Using 2's complement representation subtract 25 from 20.

Solution: We have -25_{10} = 0 0 1 1 1 (2's Complement of 25_{10})

+ 20_{10} = 1 0 1 0 0

1 1 0 1 1

As no carry, so the answer is: –(2's complement of 11011)

= –0 0 1 0 1

= –5

Note that total numbers which a word of N bits can present, using 2's complement representation, are 2^N.



CHECK YOUR PROGRESS

Q.1: What are three different schemes or methods to represent signed (negative) and unsigned integer?

Q.2: What is the range of signed integer representation of n bits?

Q.3: What is the format of the floating point representation?

Q.4: What is the signed and magnitude representation of (–20) in 8 bit word length?

Q.5: Find 1's and 2's complement of followings:

i) 100010 ii) 1010110

Q.6: State whether the following statements are true (T) or false (F):

a) For n-bit number, the range for natural number is from 0 to 2^n–1.

b) Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 is used for negative number.

c) The format of floating point representation is The format is: $Mantissa * R^{Exponent}$.

d) The two's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.

e) Given a number N in base r having n digits, the (r–1)'s complement of N is defined as (r^n–1) – N.



3.8 LET US SUM UP

- Computer arithmetic is a branch of computer engineering that deals with methods of representing integers and real values in digital systems and efficient algorithms for manipulating such numbers by means of hardware circuits or software routines.
- There are three different schemes or methods to represent signed (negative) and unsigned integer. These are:

i) Signed and Magnitude form.
ii) 1's complement form.
iii) 2's complement form

- We know that for n-bit number, the range for natural number is from 0 to 2^n–1.
- In digital computers, floating-point numbers consist of three parts: (i) a Sign bit, (ii) an Exponent part (representing the exponent on a power of 2), and (iii) a fractional part called a Significant (which is a fancy word for a Mantissa).
- Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.
- Consider a number system of base-r or radix-r. There are two types of complements.
  i) The radix complement or the r's complement.
  ii) The diminished radix complement or the (r–1)'s complement
- Given a number N in base r having n digits, the (r–1)'s complement of N is defined as $(r^n–1) – N$ **.**
- The one's complement form of any number is obtained by simply changing each 0 in the number to 1 and each 1 in the number to 0.
- The total numbers that can be represented using 1's complement are $2^N–1$, where N is the word size (number of bits in a word).
- The 2's complement form of any number is obtained by simply adding 1 to LSB of the 1's complement of that number.
- The total numbers which a word of N bits can present, using 2's complement representation, are 2^N.



3.9 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Hammacher, Carl. (Fifth Edition) (International Edition, 2002). *Computer Organization*. McGrawHill.

3) Rajaraman, V. (Fouth Edition, 2008). *Fundamentals of Computers*.

4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.

3.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: There are three different schemes or methods to represent signed (negative) and unsigned integer. These are

i) Signed and Magnitude form

ii) 1's complement form

iii) 2's complement form

Ans. to Q. No. 2: For n-bit representations of signed integer, the range is from $-2^{n-1}-1$ to $+2^{n-1}-1$

Ans. to Q. No. 3: In the representation of floating point, numbers are represented by a Mantissa comprising the significant digits and an Exponent part of Radix R.

The format is: $Mantissa * R^{Exponent}$

Ans. to Q. No. 4: –20 will be represented as follows:

Number in binary notation as binary equivalent of 20_{10} is 10100

1	0	0	1	0	1	0	0

MSB (1 for - sign)

Ans. to Q. No. 5: The 1's and 2's complement of followings are

i) 1's complement of 100010 is 011101

2's complement of 100010 is 011110

ii) 1's complement of 1010110 is 0101001

2's complement of 1010110 is 0101010

Ans. to Q. No. 6: a) True, b) False, c) True, d) False, e) True.



3.11 MODEL QUESTIONS

Q.1: What are different schemes to represent signed and unsigned integer in computer arithmetic?

Q.2: How is signed integer represented in signed and magnitude form?

Q.3: What do you mean by floating-point representation of any number in digital computer? What are its different parts?

Q.4: What is 1's complement ? Explain with example.

Q.5: What do you mean by 2's complement? Explain how can a negative integer represented using 2's complement concept.

*** ***** ***

UNIT 4: BASIC ORGANIZATION OF COMPUTER AND DESIGN

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Central Processing Unit

- 4.3.1 Register Representation
- 4.3.2 General Register Organization
- 4.3.3 Bus Organization
- 4.3.4 Stack Organization
- 4.3.5 Instruction Formats

4.4 Computer Registers

4.5 Timing and Control

4.6 Instruction Cycles

4.7 Let Us Sum Up

4.8 Further Reading

4.9 Answers to Check Your Progress

4.10 Model Questions

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn different computer registers
- learn how to execute the program by going through Instruction Cycles
- describe how to design a basic computer.

4.2 INTRODUCTION

In the previous unit, we have studied different digital components of a computer, data representation, computer arithmetic and their implementation, the design of ALU and data path, control unit design etc.

In this unit we will discuss the different major components of CPU, instruction codes, computer registers, bus organization, timing and control, instruction cycles and finally design of a basic computer.

The computer is an electronic device which can store and process data in a continuous way. There are various types of storage systems used in or related to computers like magnetic storage, optical storage and semiconductor storage. In general, magnetic storage is used in hard disk of the computer to store data and information permanently. Optical storage includes CDs, DVDs etc. and it is also a secondary storage device used for store data and information permanently. Semiconductor memory includes different registers, and counters used in different circuits in the computer.

Now the question is how is the semiconductor memory made. We have already studied about different logic gates and combinational circuits. Combinational circuits are made up of different basic logic gates like AND, OR and NOT. When we adjoin time and storage or memory element with combinational circuits, it becomes sequential circuit. Different flip flops are made up of combinational circuits and sequential circuits. One flip flop can hold one bit of data. To make a simple register of 8 bits capacity, we need 8 flip flops. Counters are made up of different registers. Flip flops, registers, counters etc are called semiconductor memory, and are used to build the major part of Central Processing Unit (CPU). The inner components of a computer are connected to the Central Processing Unit (CPU) and the memory unit via different connectors called bus. A computer has many registers. To communicate with one register to another, different path must be provided. A common bus system will provided the path which enable transferring information between registers. Transferring information between registers is called micro operation. A micro operation is an operation performed on the information stored in the registers. Computer performs specific operations on the basis of commands given to them. These commands are known as instructions.

4.3 CENTRAL PROCESSING UNIT

The part of computers which perform all the data processing operations is called Central Processing Unit or CPU. The CPU consists of three major parts. They are–

i) Register Set
ii) Arithmetic and Logic Unit or ALU and
iii) Control Unit

The Register set stores data and information, intermediate data and results obtained during the execution of instructions. The Arithmetic and Logic Unit or ALU performed the required micro-operations for executing the instructions. The control unit controls the different microoperations in between registers and instructs ALU what to do.

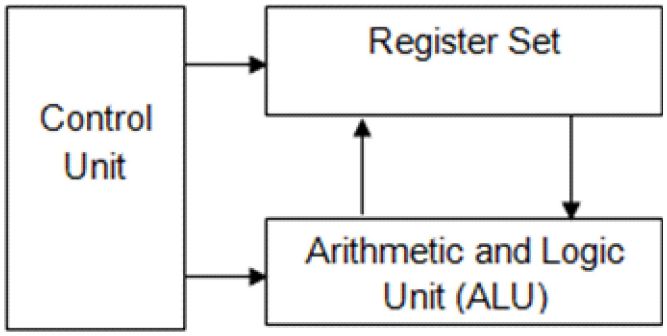


Figure 4.1: Components of CPU

4.3.1 Register Representation

Computer registers are of mainly two types: *General purpose registers* and *Special purpose registers*. General purpose registers say, n numbers of registers are assigned with capital letters like R_0, R_1, …, …, R_{n-1}. Special purpose registers are assigned names on the basis of their functions as PC (Program Counter), MAR (Memory Address Register), IR (Instruction Register), MBR (Memory Buffer Register) etc. Different types of register have different types of job. Registers are composed of different flip flops. Each flip flops can hold one bit of data. The individual flip flop in an n-bit register are numbered in sequence from 0 through n–1, starting from 0 in the rightmost position and increasing the numbers towards left as shown in the figure 4.2.

The information transferred from one register R1 to R2 is denoted by–

$$R2 \leftarrow R1$$

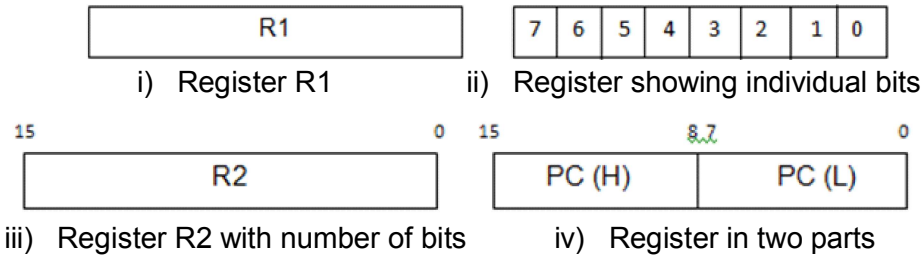


Figure 4.2: Block Diagram of Register

4.3.3 General Register Organization

In the CPU, large numbers of registers are connected to each other through a common bus system to perform different micro-operations. A bus organization for seven CPU registers R1, R2, R3, R4, R5, R6, R7 is shown in the figure 4.3. The outputs of each register is connected to two multiplexers (MUX) to form two buses A and B. The selection lines in each multiplexer select one register or input data for the particular bus. Buses A and B form the inputs to a common ALU. The result of the micro operation is available for data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder.

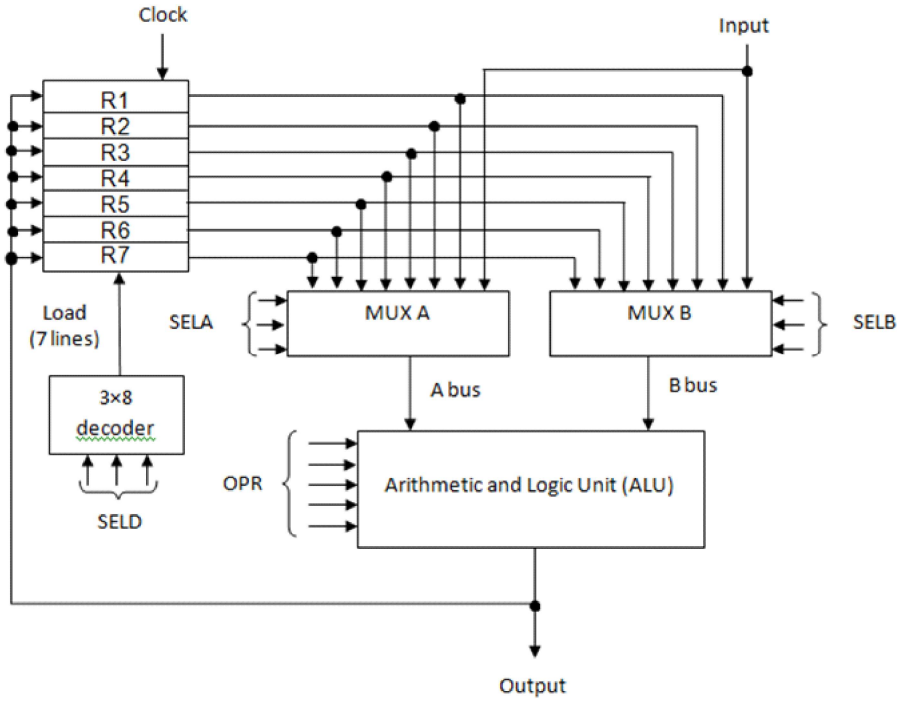


Figure 4.3: General Register and Bus Organization

Let us perform the operation: $R1 \leftarrow R2 + R3$ the control must provide binary selection variables to the following selector inputs according to the clock cycles–

a) Multiplexer A (MUX A) selector(SELA): To place the content of register R2 into bus A.
b) Multiplexer B (MUX B) selector(SELB): To place the content of register R3 into bus B.
c) Arithmetic and Logic Unit (ALU) operation selector (OPR): To select ADD operation so that ALU performs the arithmetic operation A + B.
d) Decoder destination selector(SELD): To transfer the content of the output bus into R1.

Finally we get the result of the operation (R2 + R3) in the register R1.

4.3.4 BUS Organization

A bus organization for seven processor register is shown in the figure 4.3. Each register is connected to two multiplexers (MUXA and MUXB) to form input buses A and B. The selection lines of each multiplexer select one register for the particular bus. Buses A and B are connected to the Arithmetic and Logic Unit (ALU). In ALU, the particular functions are performed and the result of the micro-operation goes through the output bus into the input of all registers. The destination register that receives the information from the output bus is selected by a 3×8 decoder.

Common Bus System: Here we are considering a basic computer having eight registers, a memory unit of size 2^{12} KB and a control unit. Paths must be provided to transfer information from one register to another through a common bus. In the figure 4.4, the output of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2,S_1, and S_0. The functional table 4.1 shows the different register selected against the binary number of decimal value from 1 to 7.

Table 4.1 Function table

S_2	S_1	S_0	Register
0	0	1	MAR
0	1	0	PC
0	1	1	MDR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

Four registers MDR, AC, IR and TR have 16 bits each. Two registers MAR and PC have 12 bits each since they hold a memory address. When the component of AR and PC are applied to the 16-

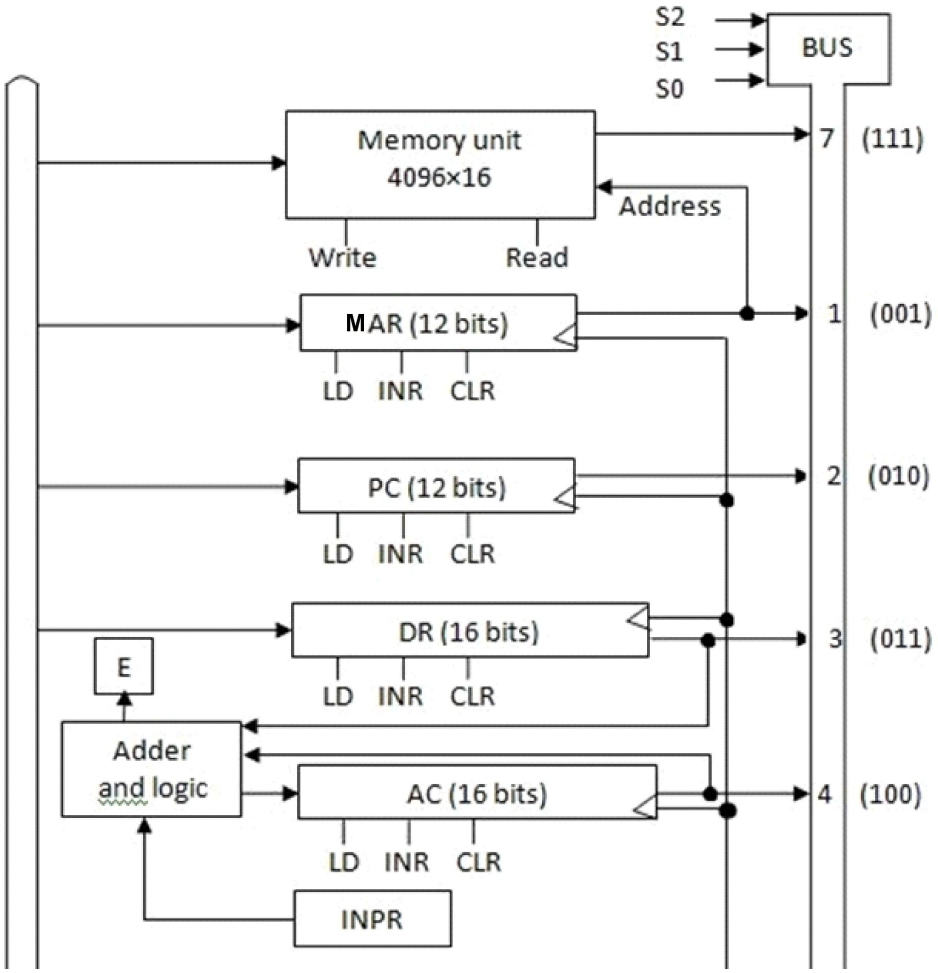


Figure 4.4: Common Bus Organization

(This figure is to be edited, AR should be MAR and DR is MDR)

bit common bus, the four most significant bits are set to 0's. The input and output register INPR and OUTR have 8 bits each and communicate with the eight least significant bits of the bus. INRP receives a character from an input device which is then transferred to AC through adder and logic. OUTR receives a character from AC and deliver it to an output device. The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.



CHECK YOUR PROGRESS

Q.1: Define microoperations.

.....

.....

Q.2: What is a stack?

.....

.....

Q.3: How much data can a flip-flop hold?

.....

Q.4: What are buses?

.....

.....

4.3.5 Stack Organization

Most CPU or different registers of the computers use stack as functional characteristics like last-in, first-out (LIFO). A stack is a storage device which stores information in such a manner that the items stored in the last must be put out first. The register that holds the address for the stack and always points to the TOP of the stack, is called a stack pointer. The two operations in a stack are insertion and deletion of data. The operation of insertion is called PUSH operation and the operation of deleting is called POP operation.

These operations are simulated by incrementing and decrementing the stack pointer register.

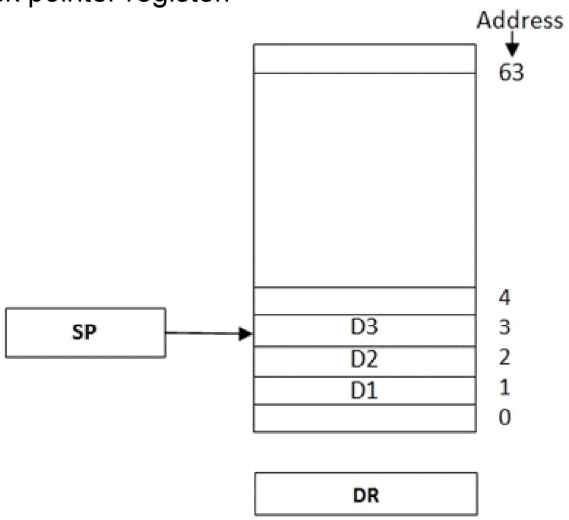


Figure 4.5: Block Diagram of a 64 Word

A stack can be implemented in a portion of large set of registers like random access memory in Central Processing Unit. Figure 4.4 shows the organization of a 64 word register stack. The stack pointer register SP is currently on top of the stack. Three data are placed in the stack D1, D2 and D3 in the memory address 1, 2 and 3 respectively. Data D3 is now on top of the stack and SP is now 3. To remove the top item, the stack is popped by decrementing the content of SP. Data D2 is now on top of the stack since SP holds address 2. Like this to insert a new data, the stack is pushed by incrementing SP and writing new word in the next higher location. MDR is the data register that holds the binary data to be written or read out of the stack.

4.3.6 Instruction Formats

An instruction is a command which is given by the user to the computer to perform a specific task or operation on the given data. Instruction includes operation code or op-code and operand,

Instruction = Op-code + Operand

The operation code or Op-code field is an instruction (group of bits) that defines different operations like add, subtract, complement

and shift. These operations are performed on some data called operands. Operands residing in memory and register are specified by their memory address and register address respectively. A register address is a binary number of n bits that defines one of 2^n registers in the CPU. Thus a CPU with $2^4 = 16$ processor registers (R_0 through R_{15}) will have a register address field of four bits.

Instructions may be of different lengths containing different numbers of addresses. The number of an address field in the instruction format of a computer system depends on the internal organization of registers. There are three types of CPU organization. They are–

i) Single accumulator organization
ii) General register organization
iii) Stack organization

Single Accumulator Organization: In general all the results of operations are stored in the accumulator register written as AC. In accumulator organization all the operations are performed with an implied accumulator register. The instruction format in this type of computer uses only one address field. For example,

ADDX, AC ← AC + M[X]

The ADD X instruction means to add the content at memory location x, i.e M[X] to the content of accumulator. Thus, the previous value of the accumulator will be replaced by the new result.

General Register Organization: In general register organization, it needs two or three addresses to perform different operations within the memory location. In two address instruction, both operand addresses are specified. The result is placed in any one of the specified addresses. In a three address instruction, two addresses are used for two operands and one address is for the result.

ADD R1, R2 R1 ← R1 + R2

ADD R1, R2, R3 R1 ← R2 + R3

In the above, the first instruction consists of only two registers addresses, R1 and R2. The contents of R1 and R2 will be added

and the result will be stored in R1. Hence R1 will act as both source and destination registers. In the second instruction there are three registers R1, R2 and R3. The result of R2+R3 will be stored in the third register R1. In this case R2 and R3 are the source register and R1 is the destination.

Stack Organization: The stack organization would have PUSH and POP instructions which require an address field. Thus the instruction will push the word at address X to the top of the stack. The stack pointer SP is updated automatically. The operation instruction does not contain any address field because the operation is performed on two topmost operands of the stack.

PUS X, TOP ⟵ M[X]

The instruction ADD consists of only operation code with no address field. This instruction pops the top two operands from the stack, add the numbers and then PUSH the result into the stack.

To illustrate the following CPU organizations using three address, two address, one address and zero address instruction, let us take one example of arithmetic expression:

X = (A+B) / (C+D)

Three Address Instruction

ADD R1, A, B R1 ⟵ M[A] + M[B]
ADD R2, C, D R2 ⟵ M[C] + M[D]
DIV X, R1, R2 X ⟵ R1/R2

M[A], M[B], M[C], M[D] denotes the operand at memory address symbolized by A, B, C, D respectively.

Two Address Instruction

MOV R1, A R1 ⟵ M[A]
ADD R1, B R1 ⟵ R1 + M[B]
MOV R2, C R2 ⟵ M[C]
ADD R2, D R2 ⟵ R2 + M[D]
DIV R1, R2 R1 ⟵ R1 / R2
MOV X, R1 M[X] ⟵ R1

One Address Instruction

LOAD A	AC	← M[A]
ADD B	AC	← AC + M[B]
STORE T	M[T]	← AC
LOAD C	AC	← M[C]
ADD D	AC	← AC + M[D]
DIV T	AC	← M[T] / AC
STORE X	M[X]	← AC

Zero Address Instruction: For zero address instruction, first we have to convert the statement using reverse polish notation. Here TOS indicates top of the stack. Hence,

X = (A+B) / (C+D) = (AB+)/(CD+) = AB+CD+/

PUSH A	TOS	← A
PUSH B	TOS	← B
ADD	TOS	← A + B
PUSH C	TOS	← C
PUSH D	TOS	← D
ADD	TOS	← C + D
DIV	TOS	← (A+B)/(C+D)
POP X	M[X]	← TOS

4.4 COMPUTER REGISTERS

There are different types of computer register used for different purposes. In general, memory data register(MDR) holds the memory operand, memory address register(MAR) holds the address for memory, accumulator is the processor register which holds the results of the processing data, program counter holds the address of instructions, temporary register holds the temporary data and the input and output register holds the input and output character.

The computer needs processor register for manipulating data and a register for holding a memory address. Depending on different functionalities, different registers have different bits of memory. Different basic registers and memory are shown in figure 4.6 with 4096 KB Main Memory and 16 bit word length.

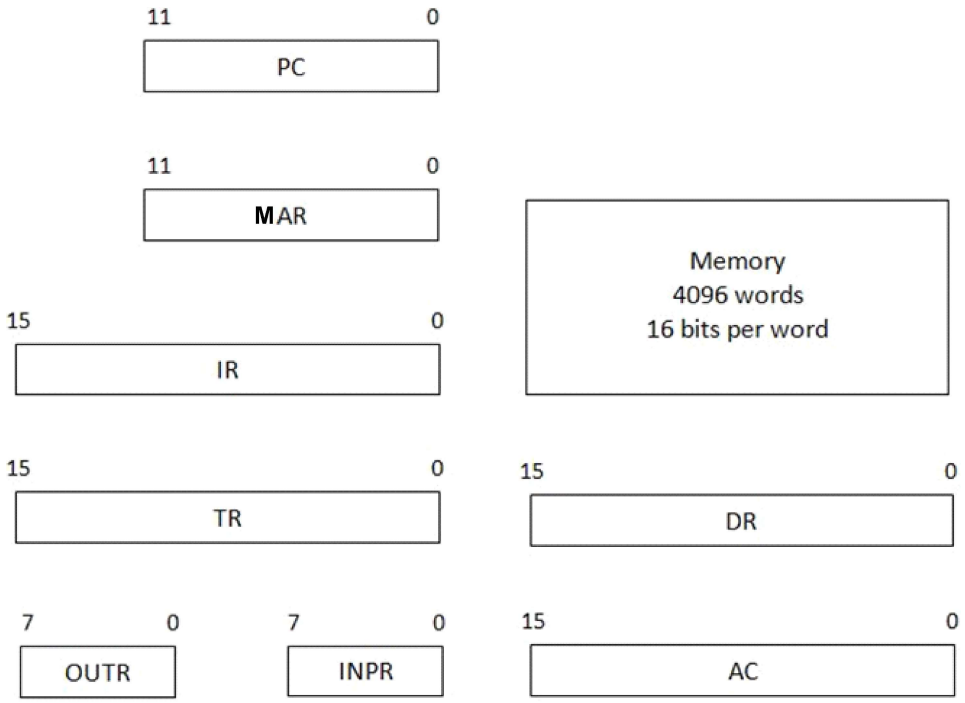


Figure 4.6: Basic Computer Register and Memory

Table 4.2: List of Registers for the Basic Computer

Register Symbol	Number of bits	Register Name	Function
MDR	16	Data register	Holds memory operand
MAR	12	Address register	Holds address for memory
AC	16	Accumulator	Process register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

The table 4.2 clearly shows different registers against their holding capacity and different functions. The accumulator (AC) register is a general purpose register which has 16 bits. The memory address register (AR or MAR) has 12 bits since this is the width of a memory address.

4.5 TIMING AND CONTROL

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all the registers including control unit in the system. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in the processor registers and microoperations for the accumulators.

Figure 4.7 shows the block diagram of a control unit. It consists of two decoders, a sequence counter and a number of control logic gates. An instruction reads from the memory is placed in the instruction register (IR), divided into three parts; the I bit, the operation code and bits 0 through 11. The operation code 12, 13 and 14 are decoded with a 3 × 8 decoder. The eight output of the decoder D_0 through D_7 applied to the control logic gates. Bit 15 of the instruction is transferred to a flip flop I. bits 0 through 11 are applied to the control logic gates. The four bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

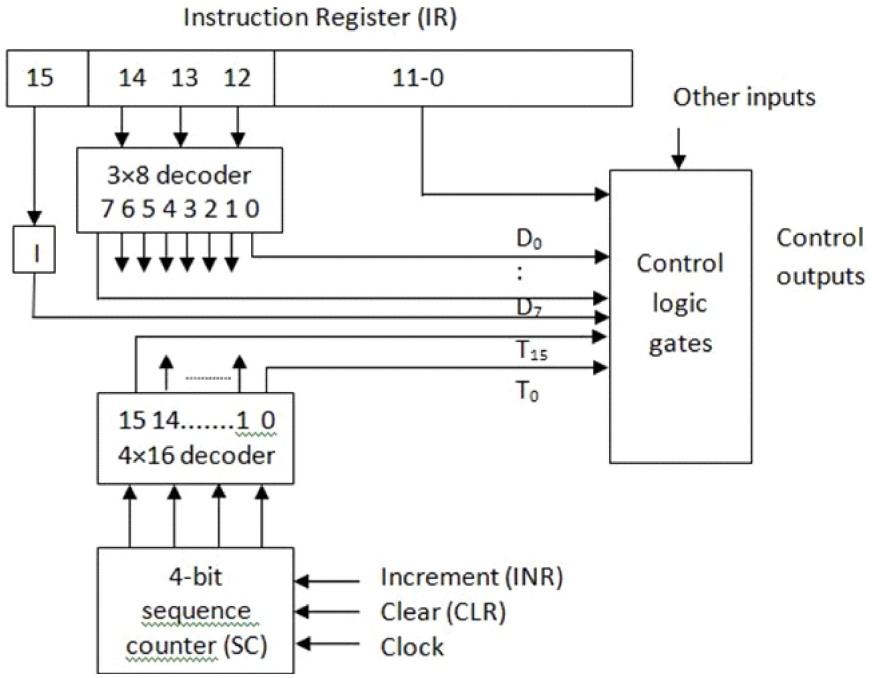


Figure 4.7: Control unit of Basic Computer

4.6 INSTRUCTION CYCLES

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. In the basic computer each instruction cycle consists of the following phases:

i) Fetch an instruction from memory
ii) Decode the instruction
iii) Read the operands or address of the operand from memory
iv) Execute the instruction.

After completion of step iv, the control goes back to step1 to fetch, decode and execute the next instruction. This process continues indefinitely.

Fetch and Decode: Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signal go through a sequence T_0, T_1, T_2 and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: MAR ⟵ PC
T_1: IR ⟵ M[AR], PC ⟵ PC + 1
T_2: D_0, ..., ..., D_7 ⟵ Decoder IR(12-14), AR ⟵ IR(0-11), I ⟵ IR(15)

Since only MAR is connected to the address inputs of memory, it is necessary to transfer the address from PC to MAR during the clock transition associated with timing signal T_0. The instruction read from memory is then placed in the instruction register IR with clock transition associated with timing signal T_1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip flop I, and the address part of the instruction is transferred to MAR. Note that SC is incremented after each clock pulse to produce the sequence T_0, T_1, T_2.

Figure 4.8 shows how the first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to MAR we must apply timing signal T_0 to achieve the following connection:

i) Place the content of PC on to the bus by making the bus selection inputs S_2 ,S_1 ,S_0 equal to 010.

ii) Transfer the content of the bus to MAR by enabling the LD input as MAR

The next clock transition initiates the transfer from PC to MAR since T_0=1. In order to implement the second statement

T_0: IR ⟵M[MAR], ⟵PCPC+1

It is necessary to use timing signal T_1 to provide the following connections in the bus system.

i) Enable the read input of memory.

ii) Place the content of memory onto the bus by making S_2 S_1 S_0 = 111

iii) Transfer the content of the bus to IR by enabling the LD input of IR.

iv) Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since T_1 = 1.

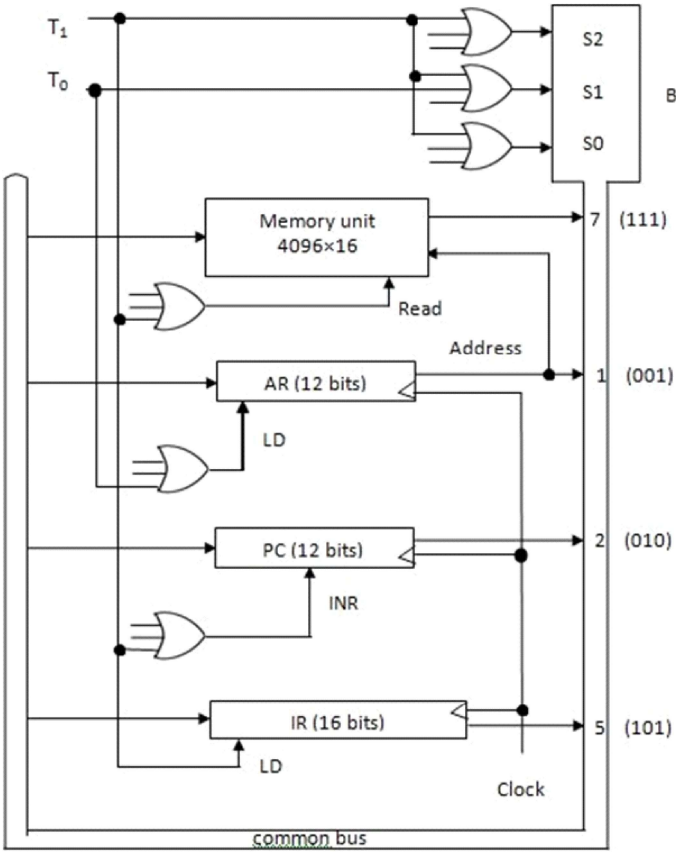


Figure 4.8: Register Transfer for Fetch Phase

Figure 4.8 duplicates a portion of the bus system and shows how T_0 and T_1 are connected to the control inputs of the registers, the memory and the bus selection inputs.



4.7 LET US SUM UP

- One flip flop can hold one bit of data.
- Transferring information between registers is called microoperation.
- To make a simple register, at least four flip flops are required.
- Transferring information between registers is called microoperation.
- The Register set stores data and information, intermediate data and results obtained during the execution of instructions.
- The basic computer has eight registers, a memory unit and a control unit.
- A stack is a storage device which store information in such a manner that the items stored in the last must be put out in first.
- The two operations in a stack are insertion and deletion of data.
- A stack can be implemented in a portion of large set of registers like random access memory in CPU.
- An instruction is a command which is given by the user to the computer to perform a specific task or operation on the given data.
- A register address is a binary number of n bits that defines one of 2^n registers in the CPU.
- In general all the results of operations are stored in the accumulator register AC.
- The timing for all registers in the basic computer is controlled by a master clock generator.
- In the basic computer each instruction cycle consists of the following phases: Fetch an instruction from memory, Decode the instruction, Read the effective address from memory, Execute the instruction.



4.8 FURTHER READING

1) Floyd, Thomas L. *Digital Fundamentals*. Pearson Education.

2) Mano, M. M. *Digital Logic and Computer Design*. Pearson Education.

3) Mano, M. M. *Computer System Architecture*. Pearson Education.

4) Rajaraman, V. & Radhakrishnan, T. *Computer Organization and Architecture*. PHI Publication.



4.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: A microoperation is an operation performed on the information stored in the registers.

Ans. to Q. No. 2: A stack is a storage device which store information in such a manner that the items stored in the last must be put out in first.

Ans. to Q. No. 3: One bit

Ans. to Q. No. 4: A bus will provide the path which enable transferring information between registers.



4.10 MODEL QUESTIONS

A) Short Answer Questions:

Q.1: How can you represent a register?

Q.2: What are the major parts of a CPU?

Q.3: Define PUSH operation in a stack.

Q.4: What will be the maximum number of registers to denote with 4 bits address field?

Q.5: What are the different types of CPU organizations?

Q.6: What are the different phases of each instruction cycles?

Q.7: What are stack organizations.

Q.8: List the different registers with functions of a basic computer.

Q.9: Define fetch and decode?

B) Long Answer Questions:

Q.1: With a suitable operation briefly describe the General Register Organization of a computer.

Q.2: Briefly describe a common bus system.

Q.3: Explain different types of addressing modes.

Q.4: Describe the control unit of a basic computer. Give suitable diagrams.

*** ***** ***

UNIT 5: INSTRUCTION SETS

UNIT STRUCTURE

5.1 Learning Objectives

5.2 Introduction

5.3 Elements of a Machine Instruction

5.4 Instruction Representation

5.5 Instruction Types

5.6 Number of Addresses

5.7 Types of Operands

5.8 Types of Operations

5.8.1 Data Transfer Operation

5.8.2 Data Processing Operation

5.8.3 Program Sequencing and Control Operation

5.9 Instruction Formats

5.10 Assembly Language Notation

5.11 Let Us Sum Up

5.12 Further Reading

5.13 Answers to Check Your Progress

5.14 Model Questions

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the elements of machine instructions
- learn about instruction format, representation and types of instructions
- explain different types of operands and operations
- learn about assembly language notations.

5.2 INTRODUCTION

In the previous unit we have learnt the numbers that are represented in the binary number system and the 2's complement system. We have already discussed the fundamental issues related to the arithmetic operations

used to support computations in computer. In this unit, we will focus on the way the programs are executed in a computer from the machine instruction viewpoint.

The program instructions and the data operands are stored in the memory. The instructions are brought from the memory to the processor and then executed to perform a given task. Operands and results must also be moved between the memory and the processor. The tasks carried out by the computer consist of some small steps. The computer must have instructions capable of performing the different types of operations. In this unit, we will discuss about the machine instructions and the different types.

5.3 ELEMENTS OF A MACHINE INSTRUCTION

The operation of a computer is determined by the instruction executed by the CPU. Such instructions are referred to as machine instructions or computer instructions. A collection of such instructions that the computer can execute is called an instruction set.

There are different elements of a machine instruction. These are as follows:

1) Operation Code.

   The operation code specifies the operation to be performed. For example, ADD, MOV, MULT etc. The operation is specified by a binary code.

2) Source operand reference.

   Source operand references are the operands that are the inputs for the operation. i.e., operands on which operation is to be performed.

3) Result operand reference.

   The operation may produce a result.

4) Next instruction reference.

After the execution of an instruction is complete, the Next instruction reference tells where from the next instruction is to be fetched.

Generally, the next instruction immediately follows the current instruction and there is no explicit reference to the next instruction. However, in some cases, the next instruction may be in the main memory or in the

secondary memory. In such cases, there is an explicit reference to the next instruction, and then the main memory or virtual memory address must be supplied.

5.4 INSTRUCTION REPRESENTATION

Each of the machine instruction is represented by a sequence of bits in the computer. The instructions are divided into fields corresponding to the constituent elements of the instruction. The machine instructions are highly machine specific.

An example of an instruction is shown in the figure below. Let us assume that the CPU is a 16-bit CPU. 4-bits are used for the operation code. Thus there can be $2^4 = 16$ different sets of instructions. The instruction has two operands each of 6-bits. Thus, there can be $2^6 = 64$ different operands for each operand reference.

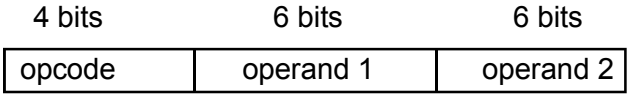


Figure. 5.1: A Simple Instruction Format

To deal with binary representation is not an easy task and so a symbolic representation of the machine instructions is used. The opcodes are represented by abbreviations called *mnemonics*. Common examples are:

ADD	Add
SUB	Subtract
MULT	Multiply
DIV	Division
LOAD	Load data from memory to CPU
STORE	Store data to memory from CPU

Let us take the following machine instruction as an example,

ADD R, A

This instruction means add the value contained in memory location

A to the contents of the register R.

5.5 INSTRUCTION TYPES

A computer should have a set of instructions that allows the users to formulate any data processing works. Instructions can be classified according to their operations and address reference.

According to the operations, the instructions can be classified as

a) Data Processing
b) Data Storage
c) Data Movement
d) Control

Data Processing: The data processing instructions include both logic and arithmetic instructions. The logic instructions perform the logical operations on the bits of the word. The arithmetic instructions provide computational capabilities for processing numeric data.

Data Storage: The logical and arithmetic operations are performed on the data in the processor registers. So there must be memory instructions for transferring the data or moving the data from the memory to the processor registers before operation and transfer data from CPU registers to memory after operations.

Data movement: The I/O instructions constitute the data movement instructions. These instructions are needed to transfer the data and the programs between the memory registers and input-output devices.

Control: The test and branch instructions are the control instructions. The test instructions test the value of a word or the status of a computation. The branch instructions are used to branch to a different set of instructions.

5.6 NUMBER OF ADDRESSES

The processor architecture can also be described in terms of the number of addresses contained in each instruction. According to the address reference, the instructions can be classified as three address instructions, two address instructions, one address instructions and zero address

instructions.

Three address instruction: Symbolically, the three address instruction can be represented as follows:

ADD A, B, C

Where A, B and C are the variables. A and B are called Source operands and variable C is called destination operand. ADD is the operation to be performed on the operands.

Two address instruction: Symbolically, it can be represented as follows:

ADD A, B

This instruction adds up the contents in A and B and stores the resultant in B overwriting the previous contents of B. Here, A is the source operand and B is both source as well as destination operand.

One address instruction: Symbolically, it can be represented as follows:

ADD A

This instruction adds the contents of A to the contents of the processor register called accumulator and then the result is stored back to the accumulator destroying the previous contents. Another example of a one address instruction is:

LOAD A

It copies the contents of the memory location A into the accumulator.

Zero address instruction: These types of instructions are used in machines that stores operands in a pushdown stack. A stack stores the locations in last-in-first-out order.



CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

i) The ..... are directly executed by the CPU of a computer.

ii) The ..... specifies the operation to be performed.

iii) Machine instruction is represented by a sequence of ..... in the computer.

iv) Instructions can be classified according to their operations and .....

v) The arithmetic instructions provide ..... capabilities for processing numeric data.

vi) According to the address reference, the machine instructions can be classified as ....., ..., ... and ..... instructions.

vii) The ..... and ..... instructions are the control instructions.

5.7 TYPES OF OPERANDS

Every instruction has two parts– *operands* and *operation code* (opcode). Operand is another name for data. There may be different types of operands. There may be different types of operands:

a) Addresses
b) Numbers
c) Characters
d) Logical data

Addresses: sometimes, calculations must be performed on the operand reference in an instruction to determine physical address.

Numbers: The common numeric data types are integer, point and decimal.

Characters: The common form of data for documentation purpose is character strings. Most of the computers use ASCII (American Standard Code for Information Interchange) code for characters represented by 7-bit pattern and 8-bit patterns. The 8^{th} bit may be set to zero or used as a parity bit for error detection. The EBCDIC (Extended Binary Coded Decimal Interchange Code) code is also used to encode characters.

Logical data: The units of data are bit, byte, word or double word. Bit means either a 0 or 1. When data is viewed as a 1-bit data, it is logical data. The logical data are used to store Boolean or binary data items. Using logical data, the bits of data can be manipulated.

5.8 TYPES OF OPERATIONS

A set of instructions are given to the computer to carry out different data processing tasks. The following are the basic types of operations that the machine instructions should be able to carry out.

- Data Transfer
- Data Processing
- Program Sequencing and Control

5.8.1 Data Transfer Operations

The data transfer operations are the most fundamental type of operations. Such transfer operations that are included here are as given below.

a) Data transfer between memory and CPU registers
b) Data transfers between CPU registers
c) Data transfer between processor and input-output devices

These instructions must specify the location of the source and the destination operands. Also the length of the data to be transferred and the mode of addressing must also be specified.

a) Data transfer between memory and CPU registers: The arithmetic and logical operations are performed on the data stored in the processor registers. Thus, before any operation, the data needs to be transferred from the memory to the CPU registers and after the operation, the resultant data has to be transferred from those registers to the memory. The two basic operations that are involved in these memory access are **Load** (Read) and **Store** (Write).

LOAD operation fetches word from memory to the processor. **STORE** operation transfers the data from the processor to the memory. The data or instructions which are transferred between processor and memory may be a byte or a word. The processor has a number of registers.

Example, R2 ⟵ [LOC]

This instruction transfers the contents of memory location to the register R2.

b) Data transfers between CPU registers: Data transfers also occur between the CPU registers. If the source and destination are registers, then the processor simply causes data to be transferred from one register to another. This operation is internal to the processor. As for example, R3 ⟵ [R2]

This instruction transfers data from CPU register R2 to the CPU register R3.

c) Data transfer between processor and input-output devices: The transfer of data between the input devices and processor and the output devices is called I/O data transfer. For example, R1 ⟵ [DATA IN]

This instruction states that the contents of I/O register, DATA IN are transferred to the processor register R1. DATA IN and DATA OUT are the registers by which the processor reads the contents from the keyboard and sends the data for display.

The rate of output data transfer is much higher than the rate of the input data transfer. However, the speed of a processor is much higher than these and can execute millions of instructions per second. So, to overcome the speed difference between these devices, synchronization mechanism should be used for proper transfer of data between them.

As shown in the figure below, the $\mathbf{S_{IN}}$ and $\mathbf{S_{OUT}}$ status bits are used to synchronize data transfer between display and the processor respectively.

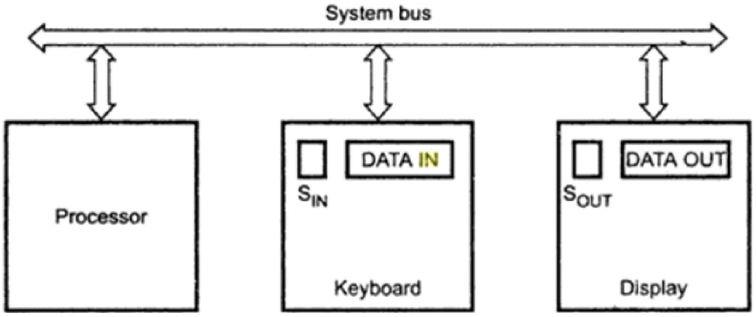


Figure 5.2: Bus Connection for Processor, Keyboard and Display

When a key is pressed, the corresponding character code is stored in DATA IN and the S_{IN} bit is set to 1. When the processor finds the S_{IN} = 1, it reads the contents of DATA IN register. After completing the read operation, the S_{IN} is automatically set to 0. If another key is pressed, the corresponding character is entered in the DATA IN register and the S_{IN} is set to 1 and the process is repeated again.

Similarly, when a character is to be transferred from processor to the display, the DATA OUT register and the S_{OUT} status bits are used. When the processor wants to transfer data to the display unit, the S_{OUT} is set to 1 and after the transfer is completed the S_{OUT} bit is cleared to 0.

5.8.2 Data Processing Operations

Data processing operations include the following types of operations.

- Arithmetic operations,
- Logical operations,
- Shift and rotate operations

The arithmetic operations basically include the following:

Add,
Subtract,
Multiply,
Divide,
Increment,
Decrement,
Negate.

For example, to add the contents of the registers R1 and R2, and to store the results in another register R3, the following instruction is used.

ADD R1, R2, R3

The logical operations include the following:

AND,
OR,

NOT,

XOR,

Compare,

Shift,

Rotate

For example, the expression:

AND R1, R2, R3

States that the contents of the processor registers R1 and R2 are logically AND operated and the result is stored in R3.

The **Compare** operation makes logical or arithmetic comparisons of two or more operands. With a logical shift, the bits of a word are shifted left or right. There are two logical shift instructions: logical shift left and logical shift right.



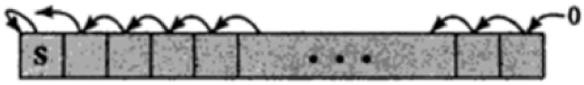
(a) Logical Right Shift



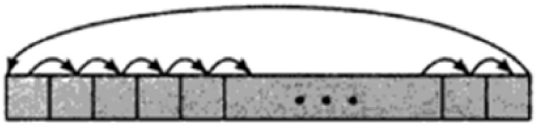
(b) Logical Left Shift



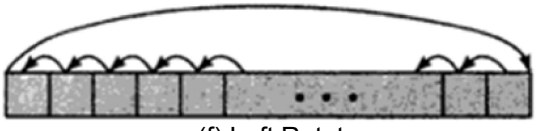
(c) Arithmetic Right Shift



(d) Arithmetic Left Shift



(e) Right Rotate



(f) Left Rotate

Figure 5.3: Shift and Rotate Operation

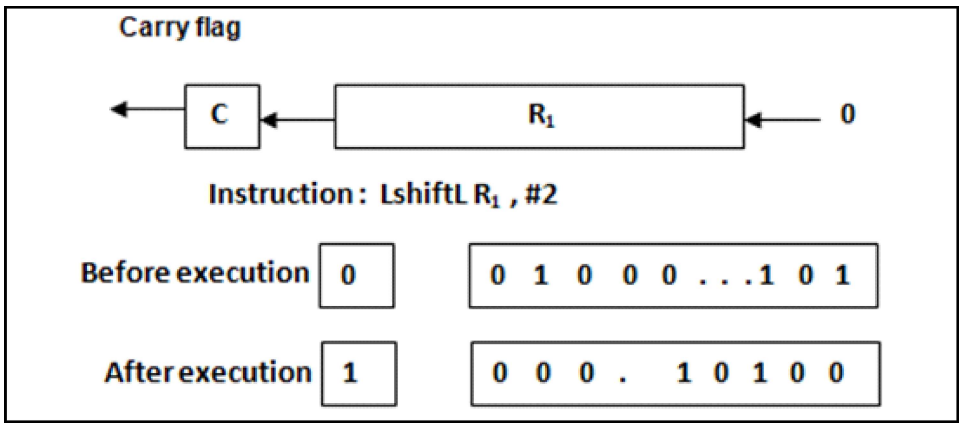
On one end, the bit shifted out is lost. On the other end, a 0 is shifted in. The logical shifts are used for isolating fields within a word. The 0s that are shifted into the word displace the unwanted information which is shifted off from the other end.

The general syntax for logical shift operations are:

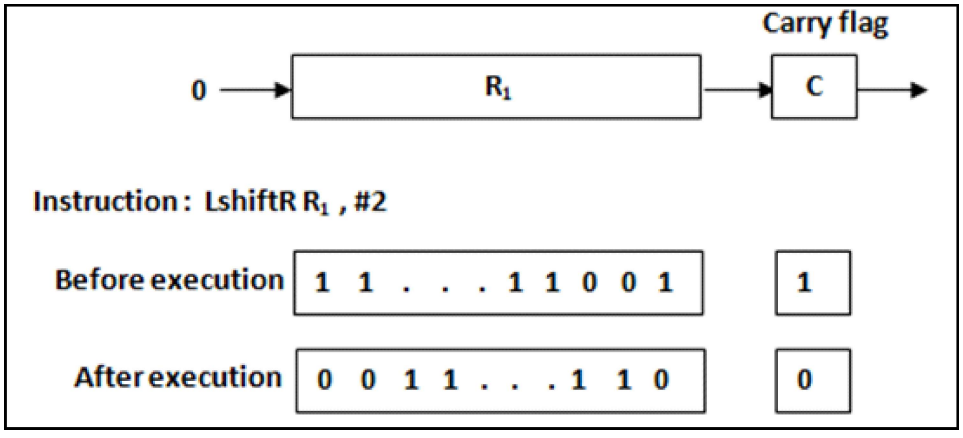
LshiftL dst, count

LshiftR dst, count

The count may be a number or the contents of a processor register. The figure (Figure. 5.4) shows the operation of LshiftL and LshiftR operations.



(a) Example of LShiftL instruction



(b) Example of LShiftR instruction

Figure: 5.4 : LshiftL and LshiftR operations

Rotate is also called cyclic shift operation. In shift operations, the bits shifted out of the operand are lost except for the last bit which is retained in the Carry flag (C). On the other hand, the rotate operation

preserves all the bits being operated on. One of the uses of the rotate operation is to bring each bit successively into the leftmost bit, where it can be identified by testing the sign of the data. There are two basic types of rotate operations– rotate left and rotate right.

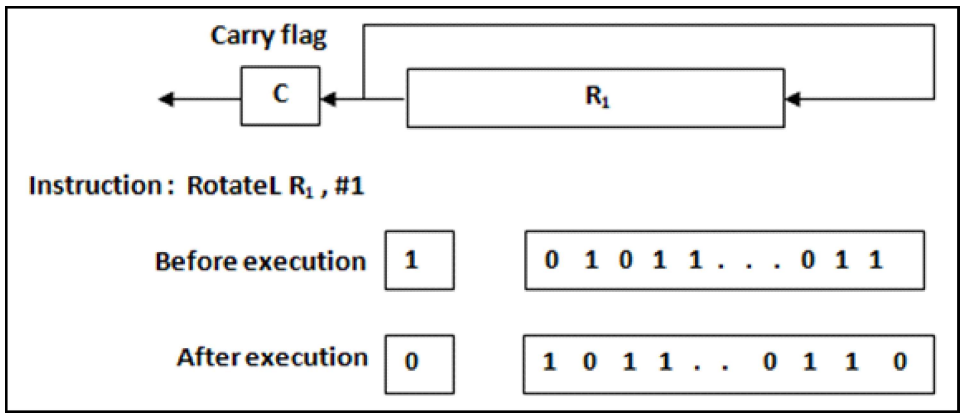


Figure: 5.5: Example of RotateL Instruction

5.8.3 Program Sequencing and Control

The program sequencing and control mainly includes the test and branch operations. Test operations are used to test the value of data and the branch operations are used to branch to a different set of operations depending on the decision made. Programs sometimes need decision making. The computer is asked to perform one thing if one condition is met and another thing if another condition holds.

A branch instruction is also called a jump instruction. One of the operand of a branch instruction is the address of the next instruction to be executed. Most often, the instruction is a conditional branch instruction. That is, a jump is made only if a condition is met otherwise the next instruction in the sequence is executed. If in a branch instruction, the branch is always made, then it is an unconditional branch.

A branch can be either a forward (an instruction with a higher address) or a backward (an instruction with a lower address). The figure below shows how an unconditional and a conditional branch

can be used to create a repeating loop of instructions. The instructions in locations 202 through 210 will be executed repeatedly until the result of subtracting Y from X is 0.

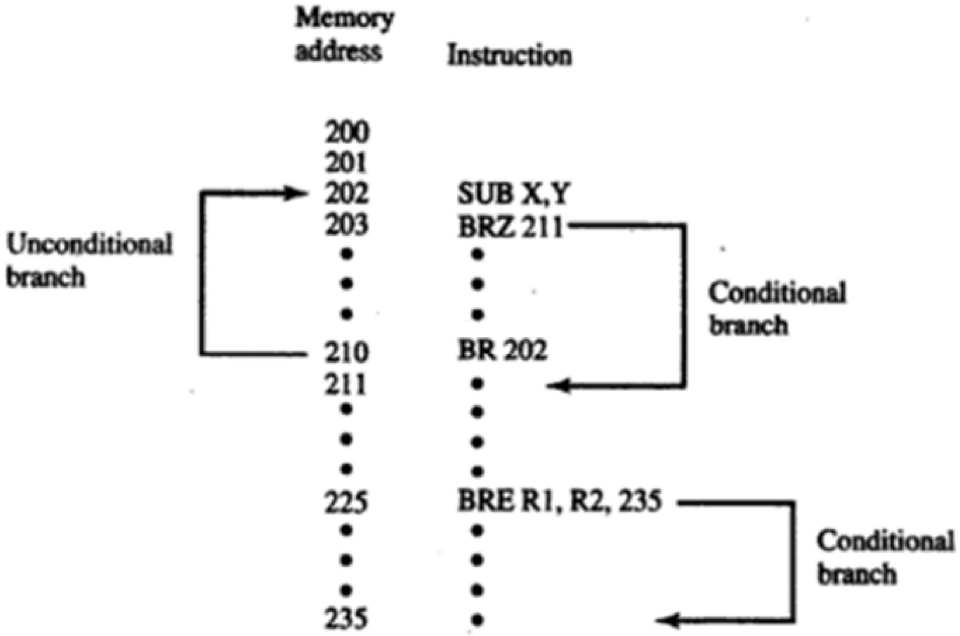


Figure.5.6: Branch Instructions

Skip instructions are also transfer of control instructions. The skip implies that an instruction be skipped and thus the implied address of the skip instruction is equal to the address of the next instruction plus one instruction length.

5.9 INSTRUCTION FORMATS

The CPU fetches the instruction from the memory, decodes it and executes the instruction according to its type. The structure of an instruction is called instruction format. It defines the layout of the bits of an instruction in terms of its constituent fields. The instruction format consists of an operation code or opcode, and zero or more operands. The operands may be implicit or explicit. The explicit operands are referenced using one of the addressing modes discussed next.

An instruction is of various lengths depending upon the number of addresses it contains. The design of an instruction format is complex. The key design issues are discussed below.

Instruction length: The instruction length is affected by the memory size, organization, bus structure, processor speed and processor complexity. The programmers want more opcodes, more operands, more addressing modes and greater address ranges. Shorter programs can be written to do a particular task with more opcodes and more operands. More addressing modes also gives the programmers flexibility in implementing certain functions. These requirements of more number of things requires more bits and so results in longer instructions. However, sometimes longer instructions may be not that useful and be a waste of space.

There are other considerations also beyond the length. Either the length of the instruction should be equal to the memory transfer length or one should be a multiple of the other. A related consideration is the memory transfer rate. The processor speed is very fast and can execute more number of instructions than the number that is fetched from the memory. So a solution to this can be to use a cache memory, another solution can be to use shorter instructions. Thus 16-bit instructions can be fetched at twice the rate of 32-bit instructions but probably can be executed less than twice as fast.

Allocation of bits: Another difficult issue with the design of the instruction format is how to allocate the bits in that format. The trade-offs are complex. There are some factors that can be useful to determine the use of addressing bits.

- Number of addressing modes. Addressing modes may be implicit or explicit.
- Number of operands
- Register versus memory. With registers only a few bits are needed to specify the register. The more registers can be used for operand references, the fewer bits are needed.
- Address range. The range of addresses that can reference memory is related to the number of address bits. As this has a severe limitation, direct addressing is rarely used. With displacement addressing, the range is opened up to the length of the address register.
- Address granularity. Granularity of addresses is another important factor for addresses that references memory rather than registers.

5.10 ASSEMBLY LANGUAGE NOTATION

Assembly language notations are used to represent machine instructions and programs. These notations use assembly language formats. Example of some assembly notations are as follows.

MOVE R1, R2

This assembly language instruction involves the transfer of contents between the registers. The contents of register R1 are transferred to R2. The contents of R1 are unchanged but those of R2 are overwritten.

ADD R1, R2, R3

This expression states that the contents of the registers R1 and R2 are added and the result is stored in the register R3.

The assembly language notations have three fields– operation, source and destination.



CHECK YOUR PROGRESS

Q.2: Fill in the blanks:

i) ..... and ..... are the two parts of an instruction.

ii) When data is viewed as a ..... data, it is logical data.

iii) Logical data are used to store ..... items.

iv) The ..... and ..... operations are performed on the data stored in the processor registers.

v) The two basic operations that are involved in memory access are ..... and .....

vi) ..... Operation transfers the data from the processor to the memory.

vii) The transfer of data between the input devices and processor and the output devices is called ..... transfer.

viii) There are two logical shift instructions called ..... and .....

ix) Rotate is also called ..... operation.

x) A branch instruction is also called a ..... instruction.



5.11 LET US SUM UP

- The operation of a computer is determined by the instruction executed by the CPU. A collection of such instructions that the computer can execute is called an *instruction set*.
- Each of the machine instruction is represented by a sequence of bits in the computer.
- The opcodes are represented by abbreviations called *mnemonics*.
- The machine instructions are represented by a sequence of bits in the computer.
- Instructions can be classified according to their operations and address reference.
- According to the operations, the instructions can be classified as Data Processing, Data Storage, Data Movement, and Control.
- According to the address reference, the instructions can be classified as three address instructions, two address instructions, one address instructions and zero address instructions.
- Every instruction has two parts– operands and operation code (opcode).
- The basic types of operations that the machine instructions should be able to carry out are Data transfer, Data processing and Program sequencing and control.
- Data transfer operations may be:
  - Data transfer between memory and CPU registers,
  - Data transfers between CPU registers,
  - Data transfer between processor and input-output devices.
- The two basic operations that are involved in memory access are Load (Read) and Store (Write).

- Data processing operations includes,
  - Arithmetic operations,
  - Logical operations,
  - Shift and rotate operations.
- The *Compare* operation makes logical or arithmetic comparisons of two or more operands.
- With a logical shift, the bits of a word are shifted left or right.
- Rotate is also called cyclic shift operation.
- In shift operations, the bits shifted out of the operand are lost except for the last bit which is retained in the Carry flag (C).
- Rotate operation preserves all the bits being operated on.
- Test operations are used to test the value of data and the branch operations are used to branch to a different set of operations depending on the decision made.
- A branch instruction is also called a jump instruction. One of the operand of a branch instruction is the address of the next instruction to be executed.
- A branch can be either a forward (an instruction with a higher address) or a backward (an instruction with a lower address).
- Skip instructions are also transfer of control instructions.
- The structure of an instruction is called instruction format. It defines the layout of the bits of an instruction in terms of its constituent fields.
- Assembly language notations are used to represent machine instructions and programs.
- The assembly language notations have three fields namely operation, source and destination.



5.12 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Gill, D. N. S. & Dixit, J. B. (2008). *Digital Design and Computer Organisation*. Firewall Media.

3) Null, L. & Lobur, J. (2014). *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers.

4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.



5.13 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Machine instructions, ii) operation code, iii) bits, iv) address reference, v) computational, vi) three address, two address, one address, zero address, vii) test, branch

Ans. to Q. No. 2: i) Operands, operation code; ii) 1-bit; iii) Boolean data; iv) Arithmetic, logical; v) Read, Write; vi) Store; vii) I/O data; viii) logical shift left, logical shift right; ix) cyclic shift; x) jump



5.14 MODEL QUESTIONS

Q.1: What are the elements of a machine instruction? Explain.

Q.2: How can a machine instruction be represented?

Q.3: Explain the various types of operands and operations.

Q.4: What are Shift and Rotate logical operations?

Q.5: Describe the Assembly language notation.

*** ***** ***

UNIT 6: ADDRESSING MODES

UNIT STRUCTURE

6.1 Learning Objectives

6.2 Introduction

6.3 Addressing

6.4 Immediate Addressing

6.5 Direct Addressing

6.6 Indirect Addressing

6.7 Register

6.8 Register Indirect

6.9 Relative Index

6.10 Let Us Sum Up

6.11 Further Reading

6.12 Answers to Check Your Progress

6.13 Model Questions

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about addressing
- describe how addressing is done
- describe different addressing modes for accessing register and memory operands.

6.2 INTRODUCTION

In the previous unit we have learnt what the machine instructions does and their formats and types. We have examined the type of operands and operations that may be specified by machine instructions. Also we have gone through the assembly language notation. In this unit, we will learn about how the address of an operand is specified, and how are the bits of an instruction organized to define the operand addresses and operation of that instruction.

An instruction is a group of bits that instruct the computer to perform a specific operation. A sequence of instructions is called a computer program. An instruction specifies the operation in the operation code field on the data specified by the operands in the operand field. There are different types of addressing capabilities in an instruction format. Addressing is an instruction design issue. There are different addressing modes that allow us to specify where the operands are located. An addressing mode can specify a location in memory, or a register or a constant. There are certain modes that allow shorter addresses, and some allow us to determine the location of the actual operand, also called the effective address of the operand dynamically. The typical addressing modes that are employed in the processor of a computer are discussed in this unit.

6.3 ADDRESSING

The techniques of specifying the address of the data are known as addressing modes. Thus the term addressing refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting the address field of an instruction before the instruction is actually executed. Computers use addressing mode techniques for the following purposes:

a) To reduce the number of bits in the field of instruction,
b) Giving programming versatility to the user by providing facilities such as pointers to memory
c) Specifying rules for modifying or interpreting address field of the instruction.

There are a variety of addressing techniques that are been applied. The most common of them are:

- Immediate
- Direct
- Indirect
- Register
- Register Indirect

The computer architectures provide more than one of these addressing modes. So, there is a question that how the processor can determine which addressing mode is being used in a particular instruction. There are several approaches for this to be done. Often, the different symbols/assembler syntax are used for different addressing modes. Also, one or more bits in the instruction format can be used as a *mode* field.

To explain the addressing modes, we use the following notation:

A = contents of an address field in the instruction that refers to a memory

R = contents of an address field in the instruction that refers to a register

EA = actual (effective) address of the location containing the referenced operand

(X) = contents of location X

The different addressing modes are explained in details in the sections below. Let us take an example to understand the various addressing modes. Suppose the two word instruction “LOAD AC”. The instruction is at

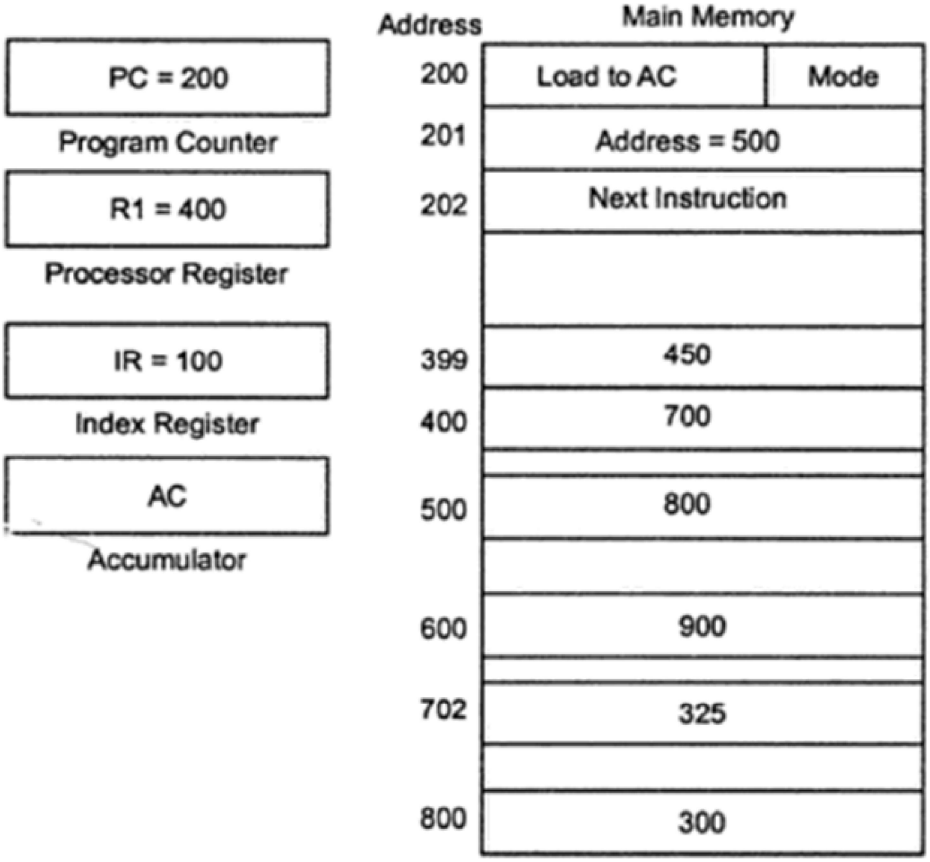


Figure 6.1. Example for Addressing Mode

address 200 and 201. The address field is 500. The first word of the instruction specifies the operation code and mode. The second word specifies the address part. The PC (Program Counter) which fetches the next instruction has the value 200. The content of the processor register R1 is suppose 400 and that of the Index register is 100. The Accumulator, AC receives the operand after the instruction is executed.

6.4 IMMEDIATE ADDRESSING

This is the simplest form of addressing. In the immediate mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field. The operand can be a byte, word or double word of data. Immediate addressing is so-named because the value to be referenced immediately follows the operation code in the instruction. Thus, the data to be operated on is part of the instruction.

OPERAND = A

The instruction format for immediate addressing mode is shown in Figure 6.2 below:

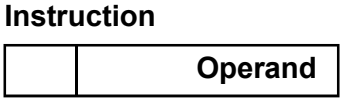


Figure 6.2: Instruction Format for Immediate Addressing Mode

In this mode, the source operand is a constant. The immediate data must be preceded by the "#" sign. It is very fast because the value to be loaded is included in the instruction. However, as the value is to be loaded is fixed at compile time, it is not very flexible.

In the figure 6.1, discussed in section 6.3, in the immediate mode of addressing the second word of the instruction is taken as the operand. i.e., the operand is 500.

This mode of addressing can be used to load information into any of the registers.

6.5 DIRECT ADDRESSING

When the operands are specified in the memory addressing mode, direct access to main memory data segment is required. However it results in slower processing of data. In direct addressing, the address field contains the effective address of the operand. The *Effective Address* is the location or the address where operand is present. It is defined to be the memory address obtained from the computation by the given addressing mode.

Direct addressing is so-named because the value to be referenced is obtained by specifying its memory address directly in the instruction.

EA = A

It requires only one memory reference and no special calculation. In the figure 6.1, discussed in section 6.3, the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

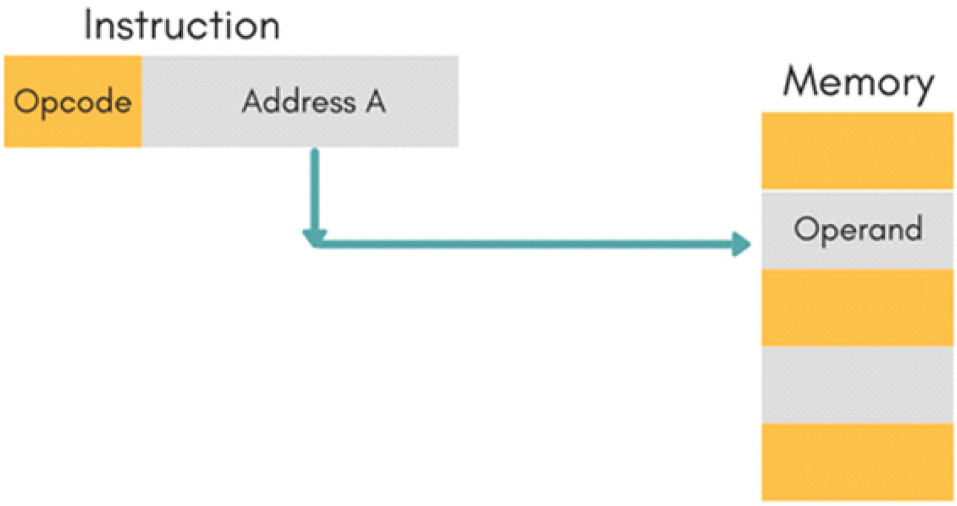


Figure 6.3: Direct Addressing

In direct memory addressing, one of the operands refers to a memory location and the other operand references a register.

Direct addressing is quite fast because although the value to be loaded is not included in the instruction, it is quickly accessible. It is also flexible than immediate addressing because the value to be loaded is given in the address.



CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

i) The way in which the operand of an instruction is specified is referred to as .....

ii) In the ....., the operand is specified in the instruction itself.

iii) Immediate addressing is so-named because the value to be referenced immediately follows the ..... in the instruction.

iv) The addressing mode specifies a rule for interpreting the ..... field of an instruction before the ..... is actually executed.

v) The immediate data must be preceded by the ..... sign.

vi) The ..... is the location or the address where operand is present.

vii) ..... requires only one memory reference and no special calculation.

viii) In direct memory addressing, one of the operands refers to a ..... and the other operand references a .....

6.6 INDIRECT ADDRESSING

Indirect addressing is a powerful addressing mode that provides an exceptional level of flexibility. The limitation of direct addressing is that it provides only a limited address space as the length of the address field is usually less than the word length. To overcome this limitation, one solution can be to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing.

EA = (A)

The parentheses means "*Contents of*".

If the address bits of the instruction code are used as an address of the actual operand, it is termed as indirect addressing. Two memory references are needed to access data. First one is to access the address of the operand and the next one to access the data from that address.

The address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.

In the figure 6.1 discussed in Section 6.3, the effective address is stored in memory at address 500. Therefore the effective address is 800 and the operand is 300.

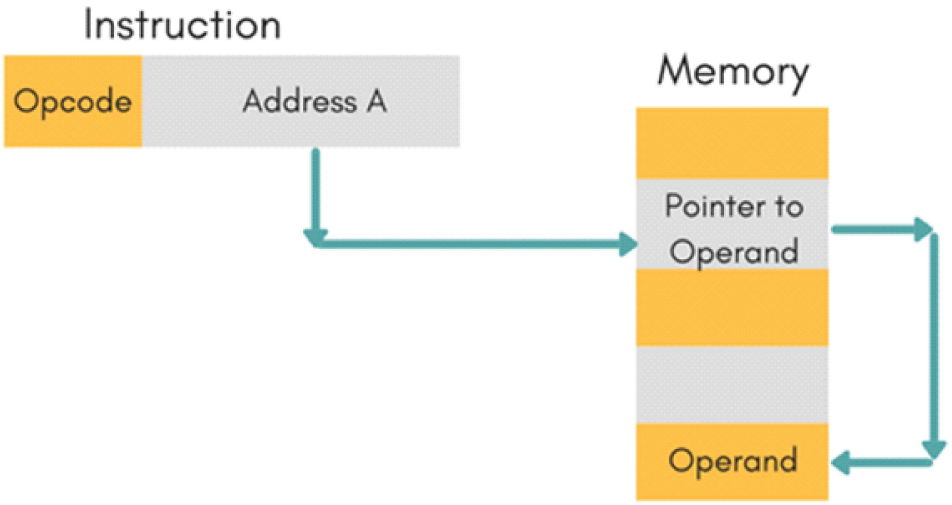


Figure 6.4: Indirect Addressing

The advantage of this addressing mode is that for a word length of N, an address space of 2^N is now available. However, if *k* is the length of the address field, the number of the effective addresses that may be referenced at any time is limited to 2^k. The bits in the address field specify a memory address that is to be used as a pointer to the operand.

The disadvantage is that instruction execution requires two memory references to fetch the operand. One is to get its address and the second is to get its value.

6.7 REGISTER

Register addressing and direct addressing are almost similar with the only difference that the address field refers to a register in Register addressing rather than a main memory address.

EA = R

In register addressing mode, instead of memory a register is used to specify the operand. In the register mode, in the figure 6.1, discussed in section 6.3, the operand is in R1and 400 is loaded into AC. There is no effective address in this case.

The advantages of register addressing are that only a small address field is needed in the instruction and no time-consuming memory reference is required. The disadvantage of register addressing is that the address space is very limited. The access time for the registers is much lesser as compared to the main memory access. As there are only a limited number of registers as that of the main memory locations, so the use of the registers should be efficient. If the operand in the register that has been brought from the main memory is used for multiple operations, then the register addressing is used efficiently. However, if the operand in the register is used for only once in the processor and then after that it is returned back to the main memory, then it is a wastage step.

6.8 REGISTER INDIRECT

Register indirect addressing is also called indexed addressing or base addressing. To load a value from memory into a register using register-indirect addressing, a second register, known as the base register is used. This base register holds the actual memory address that the program is interested in. In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.

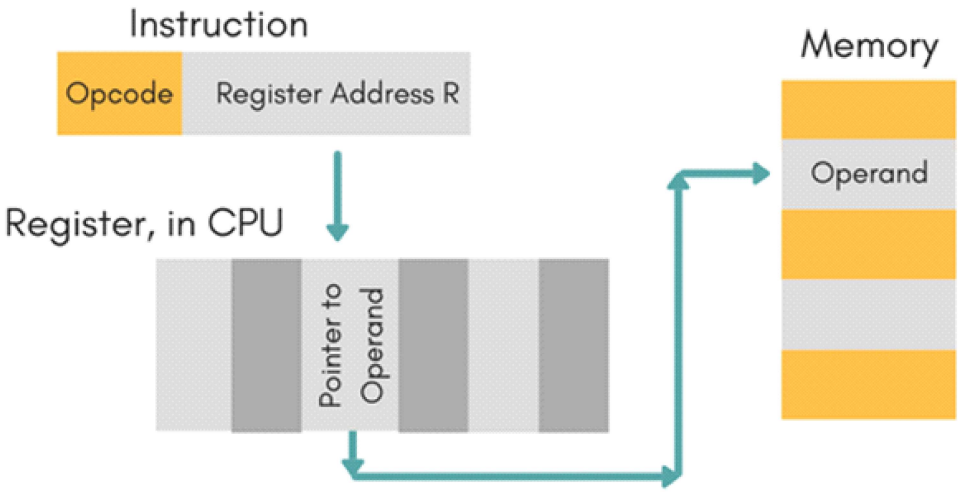


Figure 6.5: Register Addressing



CHECK YOUR PROGRESS

Q.2: Fill in the blanks:

i) The techniques of specifying the address of the data are known as .....

ii) An instruction specifies the operation in the ..... field on the data specified by the operands in the operand field.

iii) ..... is a powerful addressing mode that provides an exceptional level of flexibility.

iv) The address field of instruction, in indirect mode of addressing, gives the address where the ..... is stored in memory.

v) The bits in the address field specify a memory address that is to be used as a ..... to the operand.

vi) The address field in Register addressing refers to a ..... rather than a main memory address.

vii) The memory access time for the registers is much ..... as compared to the main memory addresses.

viii) Register indirect addressing is also called ..... addressing.

ix) The ..... holds the actual memory address that the program is interested in.

x) The use of the registers in addressing should be ..... as there are a limited number of registers.



6.10 LET US SUM UP

- An instruction is a group of bits that instruct the computer to perform a specific operation.
- An instruction specifies the operation in the operation code field on the data specified by the operands in the operand field.
- An addressing mode can specify a location in memory, or a register or a constant.
- The techniques of specifying the address of the data are known as addressing modes.
- There are a variety of addressing techniques that are been applied. The most common of them are immediate, direct, indirect, register and register indirect.
- In the immediate mode, the operand is specified in the instruction itself.
- Immediate addressing is so-named because the value to be referenced immediately follows the operation code in the instruction.
- In direct addressing, the address field contains the effective address of the operand.
- *Effective Address* is defined to be the memory address obtained from the computation by the given addressing mode.
- In direct memory addressing, one of the operands refers to a memory location and the other operand references a register.
- If the address bits of the instruction code are used as an actual operand, it is termed as indirect addressing.
- Register addressing and direct addressing are almost similar with the only difference that the address field refers to a register in Register addressing rather than a main memory address.

- As there are only a limited number of registers as that of the main memory locations, so the use of the registers should be efficient.
- Register indirect addressing is also called indexed addressing or base addressing.
- In register indirect addressing the base register holds the actual memory address that the program is interested in.



6.11 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Gill, D. N. S. & Dixit, J. B. (2008). *Digital Design and Computer Organisation*. Firewall Media.
3) Null, L. & Lobur, J. (2014). *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers.
4) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.



6.12 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Addressing; ii) immediate mode; iii) operation code; iv) address, operand; v) #; vi) Effective Address; vii) Direct addressing; viii) memory location, register

Ans. to Q. No. 2: i) addressing modes; ii) operation code; iii) Indirect addressing; iv) effective address; v) pointer; vi) register; vii) lesser; viii) indexed; ix) base register; x) efficient



6.13 MODEL QUESTIONS

Q.1: What is the purpose of using address mode techniques by a computer? Summarize the various addressing modes.

Q.2: What are the differences between direct and indirect addressing modes?

*** ***** ***

UNIT 7: INPUT-OUTPUT ORGANIZATION

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Input-Output Devices

7.4 Input-Output Interface

7.5 Different I/O Techniques

- 7.5.1 Programmed I/O
- 7.5.2 Interrupt-Driven I/O
- 7.5.3 Direct Memory Access (DMA)
- 7.5.4 I/O Processors

7.6 Let Us Sum Up

7.7 Further Reading

7.8 Answers to Check Your Progress

7.9 Model Questions

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the input and output devices
- learn how data transfer takes place between computer and peripherals
- learn about the interrupts
- define Input-output processors.

7.2 INTRODUCTION

In the previous unit we have learnt different types of addressing modes. This unit discusses how data is fed to the computer and the way they are being processed, that is, how the data is being transferred to and from the external devices to the CPU and the memory.

7.3 INPUT-OUTPUT DEVICES

The computer communicates with the users with the help of input and output devices. Input devices enter data and instructions into the computer for processing and the output devices display the result obtained for the users to see. The input-output devices attached to the computer are also known as peripherals. There are three types of peripherals: input, output and input-output peripherals. Most common peripherals are:

- Keyboard,
- Monitor,
- Printers,
- The auxiliary storage devices such as magnetic disks and tapes.

A brief description of their functions are given below :

- **Keyboard:** Keyboard allows the user to enter alphanumeric information. On pressing a key, a binary coded character, typically 7 or 8 bits in length is sent to the computer. The most commonly used code is a 7-bit code referred to as ASCII (American Standard Code For Information Interchange). Each character in this code is represented by a unique 7-bit binary code; thus a total of 128 different characters can be represented.
- **Monitors:** There are different types of video monitors but most commonly used one is the Cathode Ray Tube (CRT) monitor. Another one that becomes popular now-a-days is LCD (Liquid Crystal Display) monitor. The monitor displays a cursor on the screen which marks the position on the screen where the next character is to be inserted.
- **Printers:** Printers produce a hard copy of the output data. Based on the technology used printers may be either impact or non-impact. Impact printers operates by using a hammer to strike a character against an inked ribbon; the impact then causes an image of the character to be printed. Types of impact printers are dot-matrix, daisy wheel, line printers. Non-impact printers do not touch the paper while printing. Non-impact printers like Laser printers uses a rotating photographic drum to imprint the character images and then transfer the pattern onto the paper. Inkjet, deskjet and thermal printers also fall under this category.

- **Magnetic Tapes and Disks:** The magnetic tape drives and disk drives are used to read and write data to and from the magnetic tapes and the disks. The surfaces of these output devices are coated with magnetic material so as to store data in them. These surfaces can be magnetized and the presence of a magnetic field represents a '1' bit and the absence of a magnetic field represents a '0' bit. The magnetic disk is a circular metal disk that is coated on both sides to store data. The magnetic tape stores data in a sequential manner. To process a data in a tape ,it must begin searching at the beginning and check each record until the desired record is found.

  In case of magnetic disks, the data can be stored and accessed directly and so it is fast. Floppy disks, hard disks are examples of magnetic disks.



CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

a) The Keyboard is a ..... device.

b) Daisy Wheel is a ..... printer and the Deskjet is a ..... printer.

c) Phosphor on being struck by ..... emits light on the monitor screen.

d) Magnetic tape stores data on its ..... sides.

7.4 INPUT-OUTPUT INTERFACE

The I/O Interface is responsible for exchange of data between the peripherals and internal storage of the computer. Instead of directly connecting the peripherals to the system bus, an I/O Interface is used in between because,

1) The mode of operation of the different peripherals is different from the operation of the CPU and memory.
2) The data transfer rate of the peripherals is much slower compared to that of CPU and memory.

3) Peripherals use different data formats than the word format in the CPU and memory.

The interface units are hardware components between the CPU and the peripherals that supervise and synchronize all the I/O transfers.

Two main types of interfaces are CPU Interface that corresponds to the system bus and input-output interface that is tailored to the nature and operation of the peripheral devices. The major functions of the I/O Interface units are:

1) Data Conversion; conversion between digital and analog signals.
2) Synchronization; synchronizes speeds of the CPU and other peripherals.
3) Device Selection; selection of the I/O device by the CPU in a queue manner.

Input-Output devices are attached to the processor with the help of the I/O Interface as shown in the figure 7.1. The I/O interface circuit also has an address decoder, control circuits and data and status registers. Whenever the CPU has to communicate with a particular device, it places its address in the address lines; the address decoder in the address lines then helps in recognizing the address of its connected I/O device. The I/O interface can also generate addresses associated with the device that it controls. The control circuits accept the detailed control information from the CPU. The I/O interface also holds the responsibility of data buffering. The data registers in it holds the data being transferred to and from the interface. This is necessary as the data transfer rate into and out of the CPU or memory is quite high as compared to the peripheral devices. The status registers provides the current status information relevant to the input-output device operation. The data and status registers are connected to the data bus.

The input-output interface can detect errors and then after detection reports them to the CPU.



Figure 7.1: Processor and the I/O Interface



CHECK YOUR PROGRESS

Q.2: Write True or False:

a) The I/O Interface is a hardware component in the computer.

b) The Control lines of the System Bus carries the addresses of the peripherals.

c) Main memory can transfer data at high speed compared to the peripherals.

d) The Status register in the I/O Interface holds the data that is being transferred.

7.5 DIFFERENT I/O TECHNIQUES

There are different I/O techniques in the computer. Some may involve the CPU for transferring data between the computer and the input-output devices and others may directly transfer data from and to the memory units. The three possible techniques for I/O operations are:

- Programmed I/O,
- Interrupt-Driven I/O,
- Direct Memory Transfer (DMA)

7.5.1 Programmed I/O

In the Programmed I/O, data transfer takes place between the peripherals and the processor during program execution.

When the CPU encounters an I/O instruction, while executing a computer program, it immediately sends the address of the particular I/O device in the address lines of the I/O bus and also at the same time it issues a command to the particular I/O interface in the control lines of the I/O bus.

There are four types of commands that an interface may receive:

- Control command
- Status command
- Read command
- Write command

A *Control command* is used to activate the peripheral and tell it what to do. This command depends on the type of the particular peripheral.

A *Status command* is used to test the various status of the peripheral and the I/O interface.

A *Read command* causes the interface to get the data from the peripheral and place it in its data registers. The interface then places the data in the data lines of the bus for the processor when it requests for it.

A *Write command* causes the interface to transfer the data from the bus into its data registers. When it completes transferring data to its data registers, it then transmits that data to the particular I/O device.

In Programmed I/O, the CPU must constantly check the status of the I/O interface. That is, on issuing a command to the I/O interface,

the processor must wait until the I/O operation is complete. This is a time consuming process as it keeps the CPU needlessly busy.

7.5.2 Interrupt-Driven I/O

In Programmed I/O the processor has to constantly monitor the status of the interface to check when it is ready for data transfer. This causes wastage of the CPU time. An alternative to this method is the Interrupt-Driven method, where after issuing a command to the interface, the CPU can switch to execution of other programs in the computer. When the interface is ready for the transfer, it will immediately send an interrupt signal to the CPU. The CPU will then do the data transfer and resume back to what it was doing before it received the interrupt signal. One of the bus control lines, called an Interrupt Request Line, is dedicated for this purpose.

In this method, the advantage is that the CPU need not spend time waiting for an I/O operation to be completed, thereby increasing its efficiency.

The routine that is executed when there is an interrupt request is called Interrupt Service Routine (ISR). When the CPU gets an interrupt signal, while it is executing an instruction of a program, say instruction I, it first finishes execution of that instruction. It then stores the current contents of the Program Counter (PC), which now points to the next instruction i+1, and stores it in a memory stack. This is its return address after execution of the ISR. The control then branches to the service routine that processes the required I/O transfer.

Determining the I/O Device Requesting Interrupt: When the CPU receives an interrupt signal over the interrupt request line, it must determine which I/O interface has sent it. There are different techniques for determining this. They are:

1) Software Poll
2) Vectored Interrupt
3) Bus arbitration

1) **Software Poll:** When the CPU detects an interrupt, it branches to an interrupt-handler. This interrupt-handler poll each I/O interface to determine which interface caused the interrupt. The interface which signaled the interrupt responds positively to it. This is time-consuming.
2) **Vectored Interrupt:** This is a more efficient technique which uses a daisy chain. It provides a hardware poll. The interrupts are signaled through a common Interrupt request line. The interrupt acknowledge line is daisy-chained through the I/O interface. On receiving an interrupt, CPU sends an acknowledge signal through it. The interface which sent the interrupt responds by placing its vector on the data lines.
3) **Bus Arbitration:** At a time only one interface can send an interrupt signal. When the CPU detects the interrupt signal, it acknowledges through the interrupt acknowledge line. The interface signaling the interrupt then places its vector on the data lines.

Handling Multiple Interrupts: There may be multiple interrupts simultaneously. In those cases the CPU has to decide which one to service first. Priorities are thus assigned to the I/O interfaces to determine the order in which the interrupts are serviced. The devices having higher speeds such as the disks are given higher priorities, while the slower devices such as keyboards are given low priority. Also higher priority is given to the interrupts which if delayed may cause serious consequences. So, when two device interfaces sends interrupt requests to the processor at the same time, the device having higher priority will be serviced first.

7.5.3 Direct Memory Access

When large blocks of data are to be transferred between memory and high-speed I/O devices, a more efficient approach is used. This alternative technique is the Direct Memory Access (DMA) technique. It eliminates the continuous intervention of the CPU in transferring data.

DMA transfers are performed by a DMA controller, which is a control unit in the I/O interface. This unit carries out the functions of the processor when accessing the main memory during the data transfers. In DMA transfers, the controller takes control of the memory buses from the CPU and does the transfer between the peripheral and the memory.



Figure 7.2: DMA Controllers in a Computer System

When the CPU gets an instruction that involves huge amount of data transfer while executing a computer program, it issues a command to the DMA Controller sending the address of the I/O device, the number of words to be transferred and whether a read/ write operation is to be performed. Thus the DMA, CPU just initializes the data transfer and become free for other works. i.e., the data transfer operation is completed by the DMA controller without the continuous involvement of the CPU.

The transfer may be in two ways. They are Burst transfer and Cycle stealing.

Burst Transfer: By Burst Transfer a block of words is transferred in a continuous burst. This mode is needed for fast devices where data cannot be stopped until the entire block is transferred. Then the controller informs the CPU by sending an interrupt signal.

Cycle Stealing: The controller transfers one word at a time and return the bus control to the CPU. The CPU then temporarily suspends its operation for one memory cycle to allow the DMA transfer to steal its memory cycle.

The DMA controller uses a register for storing the starting address of the word to be transferred, a register for storing the word count of the transfer and another register contains the status and the control flags. The status and the control registers are shown below:

IRQ IE R/W Done

IRQ : Interrupt Request

IE : Interrupt Enabled

R/W : Read or Write operation

The IRQ bit is set to 1 when the controller has requested an interrupt. When the data transfer is completed, it sets the IE to 1 thereby to raise an interrupt. When it is ready for the next command, after completion, the Done bit is set to 1. When the R/W bit is 1, the controller performs a Read operation, otherwise a Write operation is performed.

Whenever a word is transferred, the DMA Controller increments its address register to point to the next word to be transferred and decrements its word count register. When finally the word count becomes zero, the controller stops any further transfer and informs the CPU by raising an interrupt.



CHECK YOUR PROGRESS

Q.3: Write True or False:

a) There are only 2 techniques for the I/O operations in the computer.

b) Read command in Programmed I/O causes the interface to get the data from I/O devices and place it in its data registers.

c) Interrupt Driven I/O wastes the CPU time needlessly.

d) Interrupt Service Routine is an interrupt signal.

e) In Direct Memory Access, the data transfer takes place directly between the peripheral and the memory.

f) The DMA Controller carries out the functions of the processor when accessing the main memory during the data transfers.

7.5.4 I/O Processors

An Input-Output processor (IOP) is a processor in addition to the CPU. It directly communicates with the input and output devices. The IOP need not interfere with the tasks of the CPU as they themselves can fetch and execute the instructions. Other than fetching and executing I/O instructions the Input Output processor can also do processing tasks like arithmetic, logic and branching.

Data from the devices are first collected into the IOP and then transferred to the memory directly by stealing one memory cycle from the CPU. Similarly data is transferred from memory to IOP first and the the gathered data is sent to the respective I/O devices.



7.6 LET US SUM UP

- Data transfer between the peripherals and the computer takes place through an Input-Output Interface.
- Each input/output device is connected to the system bus with the help of the I/O interface.
- There are three different I/O techniques :
  1) Programmed I/O
  2) Interrupt-Driven I/O
  3) Direct Memory Access (DMA)
- In Programmed I/O, the processor must constantly check the status of the interface until the I/O task is completed, which is a wastage of the CPU time.

- In Interrupt-driven I/O, the processor is free to perform other tasks while the I/O operation is going on. On its completion, the interface sends an interrupt signal to the CPU.
- In DMA, data is transferred directly between the memory and the external devices without intervening the CPU.



7.7 FURTHER READING

1) Mano, M. M. (2005). *Computer System Architecture*. Dorling Kindesley Pearson.

2) Zaky, S. G., Hamacher, V. C. & Vranesic, Z. G. (1996). *Computer Organization*. New York: McGraw-Hill.



7.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) Input; b) Impact, Non-impact; c) electrons

Ans. to Q. No. 2: a) True, b) False, c) True, d) False

Ans. to Q. No. 3: a) False, b) True, c) False, d) False, e) True, f) True



7.9 MODEL QUESTIONS

Q.1: Briefly describe the work of the I/O Interface in data transferring.

Q.2: Describe the functions of the I/O Interface units in a computer.

Q.3: What is the difference between Programmed I/O and Interrupt Driven I/O? What are their advantages and disadvantages?

Q.4: What is the task of the DMA Controller when there is a request for memory transfer?

Q.5: What are the methods for determining which I/O device has requested an interrupt?

Q.6: Why are priorities being assigned to the external devices?

*** ***** ***

UNIT 8 : INTRODUCTION TO CACHE AND VIRTUAL MEMORY

UNIT STRUCTURE

8.1 Learning Objective

8.2 Introduction

8.3 Memory Hierarchy

8.4 Semiconductor memories

8.5 Internal organization of typical RAM and ROM Memory

- 8.5.1 Internal Organisation of RAM
- 8.5.2 Internal Organisation of ROM

8.6 Switches

8.7 Cache memory

8.8 Mapping functions

8.9 Virtual memory

8.10 Locality of reference

8.11 Paging

8.12 Cache Coherence Problem

8.13 Let Us Sum Up

8.14 Further Reading

8.15 Answers to Check Your Progress

8.16 Model Questions

8.1 LEARNING OBJECTIVES

After going through this unit a learner will be able to

- identify the different levels of the memory hierarchy and describe their characteristics
- identify and explain the different types of semiconductor memories
- describe the internal organization of a typical RAM and ROM memory
- identify the types of cache memory
- explain the concept of virtual memory and its advantages
- describe the two types of locality of reference and their importance

in computer performance optimization

- define paging and explain how it is used to manage memory allocation in modern operating systems.

8.2 INTRODUCTION

Cache and virtual memory are two essential components of modern computer architecture. Cache memory is a small and fast type of memory that stores frequently used data, allowing the CPU to access it quickly. Virtual memory, on the other hand, is a technique that allows a computer to use more memory than it physically has by temporarily transferring data from RAM to a hard disk. In this introduction, we will provide an overview of cache and virtual memory, their benefits, and how they work together to improve the performance of a computer system.

In this unit we will first introduce you to the type of memories and about semiconductor memories. Then we will discuss about organisation of RAM and ROM, switches and cache memory. At the end of the unit you will be introduced to the concept of mapping function, virtual memory, locality of reference, paging and cache coherence problem

8.3 MEMORY HIERARCHY

Memory Hierarchy describes the order in which a computer system's storage devices are used to store data and instructions for the CPU to carry out. It is created to achieve an equilibrium between capacity, cost, and speed. Based on the access time, capacity, and cost of each level, the memory hierarchy is divided into different categories. Registers, cache memory, main memory, and secondary storage are some of these categories.

- **Registers:** The CPU stores data and instructions in registers, the smallest and fastest type of memory. The processor has immediate access to them because they are situated on the processor chip. Registers are used to store both the instructions that the processor needs to execute and the data that it is currently processing, such as operands and addresses. Among all the memory types in the hierarchy,

registers have the least amount of storage space.

- **Cache Memory:** Cache memory is a small amount of fast memory that is located between the processor and main memory. It is designed to reduce the average access time of the processor to data stored in main memory. Cache memory holds the most frequently used data and instructions. When the processor needs data, it first checks the cache memory. If the data is not found in cache memory, it is fetched from the main memory and stored in cache memory for future use.

  Cache memory is divided into two types: Level 1 (L1) and Level 2 (L2). L1 cache memory is the closest to the processor, while L2 cache memory is located farther away from the processor. L1 cache memory has a smaller capacity and is faster than L2 cache memory.
- **Main Memory:** Main memory, also known as Random Access Memory (RAM), is the memory used by the computer to store data and instructions that are currently being used by the processor. It is a volatile memory, which means that its contents are lost when the power is turned off. Main memory is slower than cache memory but has a larger capacity.
- **Secondary Storage:** Secondary storage refers to the long-term storage devices used by the computer, such as hard disk drives, solid-state drives, and optical drives. Secondary storage is used to store data and instructions that are not currently being used by the processor. Unlike main memory, secondary storage is non-volatile, which means that its contents are retained even when the power is turned off. Secondary storage has the largest capacity among all the memory types in the hierarchy but is slower than main memory.

In summary, the memory hierarchy is designed to provide a balance between speed, cost, and capacity. The processor uses registers and cache memory to store data and instructions that are currently being processed, while main memory and secondary storage are used to store data and instructions that are not currently being used. The hierarchy allows the processor to access data quickly by first checking the smallest and fastest memory types before accessing larger and slower ones.

8.4 SEMICONDUCTOR MEMORIES

Semiconductor memories are a type of electronic memory that uses semiconductor technology to store and retrieve digital information. They are widely used in modern computing devices such as smartphones, laptops, and servers due to their high-speed access and low power consumption. Semiconductor memories can be categorized into two types: volatile and non-volatile.

1. Volatile Semiconductor Memories: Volatile semiconductor memories lose their data when the power is turned off. The two most common types of volatile semiconductor memories are Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM).
   - SRAM: SRAM is a type of volatile semiconductor memory that uses flip-flop circuits to store data. It is faster and consumes less power than DRAM. SRAM is used for cache memory in processors and other high-speed memory applications.
   - DRAM: DRAM is a type of volatile semiconductor memory that uses capacitors to store data. It is slower and consumes more power than SRAM. DRAM is used as main memory in computers and other devices that require large amounts of memory.
2. Non-Volatile Semiconductor Memories: Non-volatile semiconductor memories retain their data even when the power is turned off. The most common types of non-volatile semiconductor memories are Read-Only Memory (ROM), Programmable Read-Only Memory (PROM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), and Flash Memory.
   - ROM: ROM is a type of non-volatile semiconductor memory that is pre-programmed with data during manufacturing. It is used to store permanent data such as firmware, boot loaders, and operating system.
   - PROM: PROM is a type of non-volatile semiconductor memory that can be programmed once after manufacturing. It is used in

applications where the data needs to be programmed only once, such as calibration data, product identification, and serial numbers.

- EPROM: EPROM is a type of non-volatile semiconductor memory that can be erased and reprogrammed multiple times. It is used in applications where the data needs to be changed frequently, such as firmware updates and configuration data.
- EEPROM: EEPROM is a type of non-volatile semiconductor memory that can be erased and reprogrammed electrically. It is used in applications where the data needs to be changed frequently, but the erasing and reprogramming process needs to be fast and easy.
- Flash Memory: Flash memory is a type of non-volatile semiconductor memory that is widely used in portable devices such as smartphones, digital cameras, and USB drives. It can be erased and reprogrammed in blocks, making it faster and more efficient than EEPROM.

In summary, semiconductor memories are a type of electronic memory that uses semiconductor technology to store and retrieve digital information. They can be categorized into two types: volatile and non-volatile. Volatile semiconductor memories lose their data when the power is turned off, while non-volatile semiconductor memories retain their data even when the power is turned off. Semiconductor memories are widely used in modern computing devices due to their high-speed access and low power consumption.

8.5 INTERNAL ORGANIZATION OF TYPICAL RAM AND ROM MEMORY

Internal organization of typical RAM and ROM memory is an important aspect of computer memory architecture. RAM or Random-Access Memory is a type of volatile semiconductor memory that allows data to be read or written in any order, making it ideal for temporary storage of data during program execution. The internal organization of a typical RAM

includes memory cells, word lines, bit lines, and address decoders that work together to provide high-speed access to data. On the other hand, a Read-Only Memory or ROM is a type of non-volatile semiconductor memory that is pre-programmed with data during manufacturing. The internal organization of a typical ROM includes memory cells, address decoders, and word lines that work together to provide permanent storage of data.

8.5.1 Internal Organisation of RAM

Random Access Memory (RAM) is a type of volatile semiconductor memory that is commonly used as the main memory in computers and other digital devices. The internal organization of a typical RAM can vary depending on the specific type and manufacturer, but there are some common elements that most RAMs share.

1. Memory Cells: The basic unit of a RAM is a memory cell, which is responsible for storing one bit of information. A memory cell typically consists of a transistor and a capacitor. The transistor acts as a switch, allowing the capacitor to either hold a charge (representing a binary 1) or discharge (representing a binary 0).
2. Address Decoder: In order to access a specific memory location in a RAM, the device needs to send an address signal to the memory. The address decoder is responsible for decoding this signal and selecting the appropriate memory cells for reading or writing.
3. Sense Amplifiers: After a memory cell has been accessed, the sense amplifiers are responsible for amplifying the signal from the memory cell to a level that can be read by the device's input/output (I/O) system.
4. Control Logic: The control logic in a RAM is responsible for managing the various operations that the memory can perform, such as reading, writing, refreshing, and activating the different memory cells.

5. Data Input/Output: The data input/output (I/O) lines of a RAM are responsible for transferring data between the memory and the device's processor or other I/O devices. These lines are typically arranged in groups of eight (for byte-level access) or sixteen (for word-level access).
6. Refresh Circuitry: RAM is a volatile memory, which means that the data it stores will be lost if the power supply is turned off. To prevent this, a refresh circuitry is included in the RAM that continuously refreshes the data stored in the memory cells, typically every few milliseconds.

In summary, the internal organization of a typical RAM includes memory cells, an address decoder, sense amplifiers, control logic, data input/output, and refresh circuitry. The specific implementation of each of these components may vary depending on the type and manufacturer of the RAM.

8.5.2 Internal Organisation of ROM

A Read-Only Memory (ROM) is a type of non-volatile semiconductor memory that is pre-programmed with data during manufacturing. The internal organization of a typical ROM is designed to store data permanently, and it is different from other types of semiconductor memories such as Random-Access Memory (RAM) or Flash Memory. Here are the key components of a typical ROM:

1. Address Decoder: The Address Decoder is responsible for decoding the address signals received from the memory controller and selecting the appropriate memory cell or word line. It uses a combination of logic gates and transistors to generate the select signals.
2. Word Lines: Word Lines are a set of conductive lines that connect the memory cells in a ROM. Each word line represents a unique memory address, and it is activated by the Address Decoder when the memory controller sends the appropriate

address signal.

3. Memory Cells: Memory Cells are the fundamental building blocks of a ROM. They are responsible for storing the data permanently, and they are typically composed of a transistor and a fuse. The state of the memory cell is determined by the presence or absence of a fuse.
4. Data Output Circuit: The Data Output Circuit is responsible for amplifying the signal from the selected memory cell and sending it to the memory controller as a digital output. It uses a combination of amplifiers and logic gates to ensure that the signal is stable and accurate.



CHECK YOUR PROGRESS

Q 1: What is the memory hierarchy?

.....

.....

Q 2: What are the two types of semiconductor memories?

.....

.....

Q 3: What is the difference between SRAM and DRAM?

.....

.....

The internal organization of a typical ROM is designed to provide fast and reliable access to pre-programmed data. The Address Decoder selects the appropriate word line based on the memory controller's address signal, and the Memory Cells store the data permanently using fuses or other non-volatile elements. The Data Output Circuit ensures that the digital output signal is stable and accurate, providing reliable data access to the memory controller.

8.6 SWITCHES

Switches play an important role in the operation of electronic memory devices such as RAM and ROM. They are used to control the flow of electricity through the circuits, allowing data to be stored, retrieved, and processed. There are two main types of switches used in electronic memory devices: transistors and fuses.

1. Transistors: Transistors are electronic devices that can act as switches or amplifiers. They are commonly used in RAM to store and retrieve data. In a RAM cell, a transistor acts as a switch that controls the flow of electricity between the memory cell and the bit line. When the transistor is turned on, electricity flows from the bit line into the memory cell, and the data stored in the cell can be read. When the transistor is turned off, the data is retained in the memory cell until it is overwritten.
2. Fuses: Fuses are electrical components that are designed to burn out when a certain amount of current flows through them. They are commonly used in ROM to store permanent data. In a ROM cell, a fuse acts as a switch that controls the presence or absence of a current path between the memory cell and the word line. When the fuse is intact, electricity flows from the word line into the memory cell, and the data stored in the cell can be read. When the fuse is burned out, the memory cell is permanently disabled, and the corresponding data cannot be read.
3. Address Decoders: Address decoders are electronic circuits that are used to select a specific memory cell or word line in the memory array. In RAM, the address decoder receives the memory controller's address signals and activates the appropriate word line and bit line. In ROM, the address decoder receives the memory controller's address signals and activates the appropriate word line, enabling the corresponding fuse to be read or written.

In summary, switches are an important component of electronic memory devices such as RAM and ROM. Transistors are used in RAM to control the flow of electricity between the memory cell and the bit line, while

fuses are used in ROM to control the presence or absence of a current path between the memory cell and the word line. Address decoders are used in both RAM and ROM to select a specific memory cell or word line in the memory array. By using switches and address decoders, electronic memory devices can provide fast and reliable access to data, making them an essential component of modern computing systems.

8.7 CACHE MEMORY

Cache memory is a small and fast type of computer memory that is used to temporarily store frequently accessed data or instructions. It is an essential component of modern computer architecture, providing faster access to data and reducing the overall latency of computer systems. In this article, we will discuss cache memory in detail, including its types, organization, and benefits.

Types of Cache Memory: There are mainly two types of cache memory: Level 1 (L1) and Level 2 (L2). L1 cache memory is a small, high-speed memory that is located inside the CPU, and L2 cache memory is a larger, slower memory that is located outside the CPU.

L1 Cache Memory: L1 cache memory is the primary cache memory in a computer system, and it is typically the fastest and smallest memory in the system. It is integrated directly into the CPU, allowing for faster access to data and instructions. The L1 cache memory is divided into two parts: instruction cache and data cache. The instruction cache stores frequently used instructions, and the data cache stores frequently accessed data.

L2 Cache Memory: L2 cache memory is a larger and slower cache memory that is located outside the CPU. It is typically located on the motherboard or integrated into the CPU package. L2 cache memory is used to store data that is not frequently accessed, or data that is too large to fit in the L1 cache memory. It is usually larger than L1 cache memory, but slower in terms of access time.

Organization of Cache Memory: Cache memory is organized into lines or blocks, each containing multiple bytes of data or instructions. When the CPU needs to access data or instructions, it first checks the L1 cache

memory. If the required data or instruction is not present in the L1 cache, the CPU checks the L2 cache memory. If the data or instruction is still not found, the CPU accesses the main memory. When data or instructions are read from the main memory, they are copied into the cache memory to improve the overall performance of the system.

Benefits of Cache Memory: The use of cache memory provides several benefits, including:

1. Faster Access to Data: Cache memory provides faster access to frequently accessed data or instructions, reducing the overall latency of the system.
2. Reduced Memory Accesses: By storing frequently accessed data or instructions in the cache memory, the number of accesses to the main memory is reduced, improving the overall performance of the system.
3. Improved System Performance: Cache memory improves the overall performance of the system by reducing the latency of data accesses and reducing the number of accesses to the main memory.

In summary, cache memory is a small and fast type of computer memory that is used to temporarily store frequently accessed data or instructions. It is an essential component of modern computer architecture, providing faster access to data and reducing the overall latency of computer systems. By using cache memory, computer systems can achieve higher performance and better responsiveness, making it an important component of modern computing.

8.8 MAPPING FUNCTION

Mapping functions are an essential aspect of computer memory management. They determine how the data is stored and accessed in memory, and they affect the performance of the entire system. In this article, we will discuss the different types of mapping functions used in computer memory, including direct mapping, fully associative mapping, and set-associative mapping.

1. Direct Mapping: Direct mapping is the simplest type of mapping function used in computer memory. In direct mapping, each block of

memory is mapped to a unique location in the cache memory. The mapping is done using a simple formula that calculates the index of the cache location based on the memory address of the data.

Direct mapping is efficient in terms of hardware implementation, but it can lead to conflicts in the cache memory. Conflicts occur when two blocks of memory are mapped to the same cache location, causing one block to be evicted from the cache memory. Direct mapping is typically used in L1 cache memory, where the cache size is small, and conflicts are less likely to occur.

2. Fully Associative Mapping: Fully associative mapping is the opposite of direct mapping, where each block of memory can be stored in any cache location. In fully associative mapping, the cache memory is searched for a particular block of memory, regardless of its location.

   Fully associative mapping provides the highest hit rate, as there is no limit to where a particular block of memory can be stored in the cache memory. However, it requires more hardware resources and is more expensive to implement than direct mapping.

   Fully associative mapping is typically used in L2 cache memory, where the cache size is larger, and conflicts are more likely to occur.

3. Set-Associative Mapping: Set-associative mapping is a compromise between direct mapping and fully associative mapping. In set-associative mapping, the cache memory is divided into sets, and each set contains a small number of cache locations. Each block of memory is mapped to one of the cache locations in the set.

   Set-associative mapping reduces the likelihood of conflicts in the cache memory, as each set has its own set of cache locations. It is less expensive to implement than fully associative mapping, but more flexible than direct mapping.

   Set-associative mapping is typically used in L3 cache memory, where the cache size is even larger than L2 cache memory, and conflicts are more likely to occur.

In summary, mapping functions are an essential aspect of computer memory management. They determine how the data is

stored and accessed in memory, and they affect the performance of the entire system. There are three types of mapping functions used in computer memory: direct mapping, fully associative mapping, and set-associative mapping. Each has its advantages and disadvantages, and they are typically used in different levels of cache memory.

8.9 VIRTUAL MEMORY

Virtual memory is a technique used by modern operating systems to increase the amount of memory available to running programs. It provides a way for programs to use more memory than is physically available in a computer's RAM by temporarily transferring data to and from the hard disk. This allows programs to run as if they have more memory than they actually do.

The concept of virtual memory is based on the idea of paging. In paging, the computer's RAM is divided into fixed-size blocks called pages. When a program requests memory, the operating system assigns it one or more pages. If the requested page is not currently in RAM, the operating system retrieves it from the hard disk and stores it in an available page. The operating system keeps track of which pages are in RAM and which are on the hard disk. When a program accesses a page that is not currently in RAM, a page fault occurs, and the operating system retrieves the page from the hard disk and puts it into an available page in RAM.

The main advantage of virtual memory is that it allows programs to use more memory than is physically available in the computer's RAM. This can be particularly useful for large programs that require a lot of memory to run. Virtual memory also makes it possible to run multiple programs simultaneously, since each program can be assigned its own set of pages in RAM.

Another advantage of virtual memory is that it allows the operating system to protect memory from unauthorized access. Each program is assigned its own set of pages in RAM, and the operating system can prevent one program from accessing the pages assigned to another program. This can help prevent bugs and malicious programs from causing damage to

the computer.

However, virtual memory also has some disadvantages. The main disadvantage is that accessing data stored in virtual memory is slower than accessing data stored in RAM. This is because data must be transferred between RAM and the hard disk, which is much slower than accessing data stored in RAM directly. This can cause programs to run more slowly when they are accessing data stored in virtual memory.

Overall, virtual memory is a useful technique that allows programs to use more memory than is physically available in the computer's RAM. While it has some disadvantages, the benefits of virtual memory generally outweigh the drawbacks, and it is an important component of modern operating systems.



CHECK YOUR PROGRESS

Q 4: What are the two main types of switches used in electronic memory devices?

.....

.....

Q 5: What is cache memory and what are its benefits?

.....

.....

Q 6: What are the three types of mapping functions used in computer memory?

.....

.....

8.10 LOCALITYOFREFERENCE

Locality of reference is a fundamental principle in computer science that refers to the tendency of programs to access the same memory locations repeatedly, rather than accessing random locations. It is a property of memory access patterns that can be observed in many types of programs, from simple algorithms to complex applications.

The concept of locality of reference is important for understanding how computer memory is used and how performance can be optimized. There are two main types of locality of reference: temporal and spatial locality.

Temporal locality refers to the tendency of programs to access the same memory locations repeatedly over a short period of time. For example, a program that processes data in a loop may access the same data multiple times during each iteration of the loop. By keeping this data in the cache or in registers, the program can avoid the overhead of repeatedly accessing it from main memory.

Spatial locality refers to the tendency of programs to access memory locations that are near each other in memory. For example, a program that processes an array of data may access adjacent elements of the array sequentially. By keeping these elements in the cache or in registers, the program can take advantage of the faster access times and reduce the number of cache misses.

There are several techniques that can be used to take advantage of the principle of locality of reference. Caching is one such technique that is widely used in computer systems. Caches are small, fast memory devices that store recently accessed data and instructions. By keeping frequently accessed data and instructions in the cache, the computer can reduce the number of memory accesses and improve performance.

Another technique that can be used to take advantage of locality of reference is prefetching. Prefetching involves predicting which memory locations are likely to be accessed in the near future and fetching them into the cache before they are actually needed. This can reduce the latency of memory accesses and improve performance.

In summary, locality of reference is a fundamental principle that plays a critical role in computer system design and performance optimization. By understanding and exploiting the temporal and spatial locality of memory access patterns, computer systems can achieve better performance and efficiency.

8.11 PAGING

Paging is a memory management technique used by modern operating systems to manage the allocation of physical memory to running programs. It allows the operating system to divide the physical memory into small fixed-size blocks called pages, and to allocate pages to running programs as needed. Paging is used in combination with virtual memory to increase the amount of memory available to programs.

In paging, the physical memory is divided into fixed-size blocks called frames, and the virtual memory is divided into fixed-size blocks called pages. Each program is allocated a set of pages in virtual memory, and these pages are mapped to frames in physical memory by the operating system. The mapping between virtual pages and physical frames is maintained in a page table, which is used by the operating system to translate memory addresses used by the program into physical memory addresses.

When a program accesses a page that is not currently in physical memory, a page fault occurs. The operating system retrieves the required page from disk and stores it in a free frame in physical memory. If there are no free frames, the operating system selects a frame to evict, typically using a page replacement algorithm, to make space for the required page. The evicted page is written to disk if it has been modified since it was last read from disk.

Paging has several advantages over other memory management techniques. First, it allows the operating system to allocate physical memory to running programs on demand, rather than requiring that all memory be allocated when the program is started. This allows programs to use more memory than is physically available, and can improve the performance of the system by reducing the amount of swapping required.

Second, paging allows the operating system to protect memory from unauthorized access. Each program is allocated its own set of pages in virtual memory, and the operating system can prevent one program from accessing the pages allocated to another program. This can help prevent bugs and malicious programs from causing damage to the system.

Finally, paging allows the operating system to optimize memory usage by using a page replacement algorithm to select the most appropriate page to evict when physical memory is full. There are many page replacement algorithms that can be used, including least recently used (LRU), least frequently used (LFU), and random replacement. The choice of algorithm can have a significant impact on the performance of the system.

However, paging also has some disadvantages. First, the overhead of managing the page table and handling page faults can reduce system performance. Second, the need to transfer pages between physical memory and disk can result in slower access times compared to using physical memory alone. Finally, the use of paging can result in fragmentation of physical memory, which can reduce the amount of available memory.

In summary, paging is a memory management technique used by modern operating systems to manage the allocation of physical memory to running programs. It allows programs to use more memory than is physically available, provides memory protection, and optimizes memory usage. However, it also has some overhead and can result in slower access times and fragmentation of physical memory.

8.12 CACHE COHERENCE PROBLEM

The cache coherence problem is a fundamental challenge in computer architecture that arises when multiple processors or cores in a system have their own private caches, which can lead to inconsistencies in the shared memory data between them. This problem arises due to the fact that each processor or core may have its own copy of a particular data block, and if any processor updates its copy of the data, the other processors may not be aware of the update.

In a multiprocessor system, the caches are typically organized in a hierarchy, with each processor having its own private cache, which is closer to the processor than the main memory. This cache hierarchy is used to improve the performance of the system by reducing the latency of memory accesses. However, this also leads to the cache coherence problem.

There are several techniques that can be used to maintain cache coherence in a multiprocessor system. The most common technique is called the snooping protocol. In this protocol, each cache controller monitors the bus to determine whether any other processor is accessing a shared memory location. If a processor writes to a shared memory location, the cache controller broadcasts an invalidation message to all other processors that may have a copy of the data block. This forces them to invalidate their copy of the data block and to fetch the latest copy from main memory or from the processor that made the update.

Another technique that can be used to maintain cache coherence is called the directory protocol. In this protocol, each cache controller maintains a directory that keeps track of which processors have a copy of each data block. When a processor updates a data block, it sends a message to the directory, which updates the status of the data block in the directory and sends a message to all other processors that have a copy of the data block to invalidate their copy. When a processor needs to read a data block, it sends a message to the directory, which determines which processor has the latest copy of the data block and sends it back to the requesting processor.

Cache coherence is important because it ensures that all processors in a multiprocessor system have a consistent view of the shared memory data. Inconsistencies in the shared memory data can lead to incorrect program behaviour, such as data corruption or program crashes. Therefore, it is essential that the cache coherence problem be addressed in any multiprocessor system.

In summary, the cache coherence problem is a fundamental challenge in computer architecture that arises when multiple processors or cores in a system have their own private caches. Techniques such as the snooping protocol and the directory protocol can be used to maintain cache coherence and ensure that all processors in a multiprocessor system have a consistent view of the shared memory data.



CHECK YOUR PROGRESS

Q 7: What is locality of reference?

.....

.....

.....

Q 8: What is paging?

.....

.....

.....

Q 9: What is the cache coherence problem?

.....

.....

.....



8.13 LET US SUM UP

- The memory hierarchy in computer systems is designed to achieve a balance between speed, capacity, and cost. It consists of four levels, namely registers, cache memory, main memory, and secondary storage.
- Semiconductor memories are a type of electronic memory widely used in modern computing devices such as smartphones, laptops, and servers. They can be categorized into volatile and non-volatile types. Volatile semiconductor memories lose their data when power is turned off, while non-volatile semiconductor memories retain their data even when the power is turned off.
- The most common types of semiconductor memories include SRAM, DRAM, ROM, PROM, EPROM, EEPROM, and Flash Memory. They have different characteristics in terms of speed, power consumption, and reprogrammability, which make them suitable for different applications in computer systems and other electronic devices.

- RAM and ROM are two types of semiconductor memory used in computers and other digital devices. RAM is volatile and allows data to be read or written in any order, while ROM is non-volatile and pre-programmed with data during manufacturing.
- The internal organization of a typical RAM includes memory cells, an address decoder, sense amplifiers, control logic, data input/output, and refresh circuitry. The specific implementation of each of these components may vary depending on the type and manufacturer of the RAM. The internal organization of a typical ROM includes address decoders, word lines, memory cells, and data output circuitry.
- Switches play an important role in the operation of electronic memory devices such as RAM and ROM. Transistors and fuses are the two main types of switches used in electronic memory devices, with transistors used in RAM to store and retrieve data and fuses used in ROM to permanently store pre-programmed data.
- Cache memory is a small and fast type of computer memory that stores frequently accessed data or instructions, providing faster access to data and reducing the overall latency of computer systems. It is organized into lines or blocks, each containing multiple bytes of data or instructions.
- Mapping functions are an essential aspect of computer memory management that determine how data is stored and accessed in memory. The three types of mapping functions are direct mapping, fully associative mapping, and set-associative mapping. Each has its advantages and disadvantages, and they are typically used in different levels of cache memory.
- Virtual memory is a technique used by modern operating systems to increase the amount of usable memory available to a computer system. It uses a combination of RAM and hard drive space to simulate additional RAM, allowing programs to run even if they require more memory than physically available. Virtual memory can improve system performance but may also lead to performance issues if not managed properly.
- Locality of reference is a principle in computer science that refers to

the tendency of programs to access the same memory locations repeatedly, which can be exploited to improve computer performance through techniques such as caching and prefetching.

- Paging is a memory management technique used by modern operating systems that allows the allocation of physical memory to running programs on demand, increases the amount of memory available to programs, and protects memory from unauthorized access. However, it also has some overhead and can result in slower access times and fragmentation of physical memory.
- The cache coherence problem arises when multiple processors or cores in a system have their own private caches, which can lead to inconsistencies in the shared memory data between them. Techniques such as snooping, directory-based coherence, and cache-invalidation protocols are used to solve this problem.



8.14 FURTHER READINGS

1) K. Hwang and F.A. Briggs. (2018). Computer Architecture and Parallel Processing. Tata McGraw-Hill Education.

2) V. Heuring and H. Jordan. (2017). Computer Systems Design and Architecture. Pearson Education India.



8.15 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: The organization of a computer's storage devices based on speed, capacity, and cost

Ans to Q No 2: Volatile and non-volatile.

Ans to Q No 3: SRAM is faster and consumes less power than DRAM, but DRAM has a larger capacity and is used as main memory in computers.

Ans to Q No 4: Transistors and fuses.

Ans to Q No 5: Cache memory is a small and fast type of memory used to

temporarily store frequently accessed data or instructions, providing faster access to data and reducing the overall latency of computer systems.

Ans to Q No 6: Direct mapping, fully associative mapping, and set-associative mapping.

Ans to Q No 7: Locality of reference is a principle in computer science that refers to the tendency of programs to access the same memory locations repeatedly.

Ans to Q No 8: Paging is a memory management technique used by modern operating systems to manage the allocation of physical memory to running programs.

Ans to Q No 9: The cache coherence problem is a fundamental challenge in computer architecture that arises when multiple processors or cores in a system have their own private caches, which can lead to inconsistencies in the shared memory data between them.



8.16 MODEL QUESTIONS

Q 1: Explain the concept of the memory hierarchy and its significance in modern computing devices. How does the memory hierarchy achieve a balance between speed, cost, and capacity?

Q 2: Describe the two types of semiconductor memories and their respective features.

Q 3: Explain the role of switches in electronic memory devices such as RAM and ROM.

Q 4: What is cache memory, and how does it work to improve computer performance?

Q 5: Explain in detail how locality of reference is used in computer systems to improve performance. Give examples of techniques that take advantage of temporal and spatial locality, and describe how they work.

Q 6: Explain in detail how a cache works in a computer system.

*** ***** ***

UNIT 9 : MEMORY AND I/O ACCESS

UNIT STRUCTURE

9.1 Learning Objective

9.2 Introduction

9.3 Memory Read Write Operations

9.4 Concept of Handshaking

9.5 Polling Techniques

- 9.5.1 Serial Polling
- 9.5.2 Hub Polling

9.6 Interrupt driven I/O

9.7 Priority and Daisy Chaining Technique

9.8 Memory Mapping

9.9 Let Us Sum Up

9.10 Further Readings

9.11 Answers to Check Your Progresses

9.12 Possible Questions

9.1 LEARNING OBJECTIVES

After going through this unit, you will be able to-

- identify the process of memory read and write operations
- explain the concept of handshaking in computer memory
- describe the four signals involved in the handshaking process
- differentiate between serial and hub polling techniques in computer memory
- evaluate the efficiency of polling techniques based on the number of memory devices in a system.

9.2 INTRODUCTION

Memory and input/output (I/O) devices are essential components of modern computing systems. Memory devices are used to store and retrieve data and instructions that are required for the operation of a computer,

while I/O devices are used to communicate with the external world, including devices such as keyboards, mice, printers, and network interfaces. In this way, memory and I/O devices work together to enable the processing of data and the interaction of the computer with its environment.

The unit discusses the important concepts related to computer memory, including memory read and write operations, handshaking, and polling techniques. It explains how these concepts are crucial for the proper functioning of computers and enable them to store information, execute programs, and perform complex calculations. The unit also highlights the need for high-speed and efficient data transfer in modern computer systems.

9.3 MEMORY READ WRITE OPERATIONS

Memory read and write operations refer to the process of accessing and manipulating data in computer memory. These operations are critical to the functioning of computers and are used in all types of computing devices, from personal computers to supercomputers.

Memory Read Operations: Memory read operation is the process of retrieving data from computer memory. When a program needs to access data from memory, it sends a read request to the memory controller, which retrieves the data from the specified memory location and transfers it to the program. The read operation can be performed on any type of memory, including RAM, ROM, and cache memory.

The process of memory read operations is initiated by the processor by sending an address to the memory controller. The address specifies the memory location where the data is stored. The memory controller then retrieves the data from that location and sends it back to the processor. The retrieved data is temporarily stored in the processor's register or cache memory, where it can be accessed by the program.

Memory Write Operations: Memory write operation is the process of storing data into computer memory. When a program needs to modify the value of a variable, it sends a write request to the memory controller, which stores the new value at the specified memory location. The write operation can also be performed on any type of memory, including RAM,

ROM, and cache memory.

The process of memory write operations is initiated by the processor by sending an address and the new data to the memory controller. The address specifies the memory location where the new data is to be stored. The memory controller then writes the new data into that location. Once the write operation is complete, the new data is stored in memory, and it can be accessed by other programs.

Memory read and write operations are critical to the functioning of modern computers. They enable programs to access and manipulate data stored in memory, allowing computers to perform complex calculations, store information, and execute programs. The efficiency of memory read and write operations can have a significant impact on the overall performance of a computer system, making it essential for memory controllers to be designed for high-speed and efficient data transfer.

9.4 CONCEPT OF HANDSHAKING

In computer memory, handshaking refers to the process of establishing communication between the memory controller and the memory device. It involves a series of signals that are exchanged between the two devices to ensure that data is transmitted accurately and reliably.

Handshaking is essential in computer memory as it helps to prevent errors that can occur during data transfer. The memory controller and the memory device must be synchronized to ensure that data is transferred at the correct time and in the correct sequence. This is particularly important in modern computer systems where the speed of data transfer is critical for optimal performance.

The handshaking process typically involves a series of signals that are exchanged between the memory controller and the memory device. These signals include:

1. Address Signal - The memory controller sends an address signal to the memory device indicating the location of the data to be read or written.

2. Data Signal - The memory controller sends a data signal to the memory device when data is to be written, or the memory device sends a data signal to the memory controller when data is to be read.
3. Control Signal - The control signal is used to indicate the start and end of a transaction. It signals the memory device that the data transfer is about to begin and when it is complete.
4. Acknowledgment Signal - The memory device sends an acknowledgment signal to the memory controller to indicate that the data has been received correctly.

The handshaking process ensures that the memory controller and the memory device are synchronized and that data is transmitted accurately and reliably. The process also helps to prevent errors that can occur during data transfer, such as data corruption, data loss, or incorrect data sequencing.

In summary, handshaking is a critical concept in computer memory as it helps to ensure accurate and reliable data transfer. By exchanging a series of signals between the memory controller and the memory device, handshaking helps to synchronize the two devices, prevent errors, and improve the overall performance of the computer system.

9.5 POLLING TECHNIQUES

Polling is a technique used in computer memory to manage data transfer between the memory controller and the memory device. It involves the memory controller periodically checking the status of the memory device to determine if it is ready to transfer data. There are two primary polling techniques used in computer memory: Serial polling and Hub polling.

9.5.1 Serial Polling

Serial polling is a polling technique in which the memory controller checks the status of each memory device in a serial manner, one at a time. This technique is used in systems where there is a small number of memory devices that need to be polled.

In serial polling, the memory controller sends a polling signal to the first memory device and waits for a response. If the first memory device is ready, it sends an acknowledge signal back to the memory controller, indicating that it is ready to transfer data. The memory controller then proceeds to poll the next memory device and repeats the process until all the memory devices have been polled.

Serial polling is simple to implement and is suitable for systems with a small number of memory devices. However, it can be slow and inefficient in systems with a large number of memory devices, as each device must be polled individually.

9.5.2 Hub Polling

Hub polling is a polling technique in which a hub device is used to manage the polling of multiple memory devices. In this technique, the memory controller sends a polling signal to the hub device, which then polls each memory device in parallel. The hub device collects the responses from each memory device and sends them back to the memory controller.

Hub polling is more efficient than serial polling, as multiple memory devices can be polled simultaneously. It is suitable for systems with a large number of memory devices or systems that require high-speed data transfer.

Polling techniques are used in computer memory to manage data transfer between the memory controller and the memory device. Serial polling is suitable for systems with a small number of memory devices, while hub polling is more efficient and suitable for systems with a large number of memory devices. The choice of polling technique depends on the system requirements, including the number of memory devices, data transfer speed, and system complexity.



CHECK YOUR PROGRESS

Q 1: What is memory read operation?

.....

.....

Q 2: What is the purpose of handshaking in computer memory?

.....

.....

Q 3: What is hub polling?

.....

.....

9.6 INTERRUPT DRIVEN I/O

Interrupt-driven I/O is a technique used in computer memory to manage input/output operations. It involves the use of interrupts to notify the processor when data is ready to be transferred between the memory controller and the memory device.

When a memory device has data to transfer, it sends an interrupt request to the processor, which interrupts the current program and switches to an interrupt handler. The interrupt handler then communicates with the memory device and transfers the data to or from the memory device as required.

Interrupt-driven I/O has several advantages over other I/O techniques. Firstly, it is more efficient as it allows the processor to continue executing other instructions while waiting for data to be transferred. This means that the processor can perform other tasks while waiting for I/O operations to complete, improving system performance.

Secondly, interrupt-driven I/O is more flexible as it allows the processor to handle multiple I/O requests simultaneously. This means that multiple memory devices can be serviced at the same time, increasing the overall I/O throughput.

Thirdly, interrupt-driven I/O is more reliable as it ensures that no data is lost or corrupted during the transfer process. The interrupt handler

ensures that the data transfer is complete before returning control to the processor, reducing the risk of errors.

However, interrupt-driven I/O also has some drawbacks. It requires additional hardware and software support to manage the interrupt requests and handlers, which can increase system complexity. It also requires careful handling to avoid interrupt storms, where multiple interrupts are generated simultaneously, which can cause system instability.

In summary, interrupt-driven I/O is a technique used in computer memory to manage input/output operations. It uses interrupts to notify the processor when data is ready to be transferred, providing a more efficient, flexible, and reliable I/O solution. However, it also requires additional hardware and software support and careful handling to avoid system instability.

9.7 PRIORITY AND DAISY CHAINING TECHNIQUE

Priority and daisy chaining are two techniques used in computer memory to manage interrupt requests from multiple devices. These techniques help ensure that interrupts are handled in a timely and efficient manner.

Priority Technique: The priority technique is a method of handling interrupt requests based on the priority of the device generating the request. Each device is assigned a priority level, with the highest priority devices being serviced first. When an interrupt request is received, the processor checks the priority level of the device and services the highest priority request first.

The priority technique ensures that the most critical interrupt requests are handled first, improving system responsiveness and reducing the risk of data loss or corruption. However, it may also result in lower priority requests being delayed, which can affect the overall system performance.

Daisy Chaining Technique: The daisy chaining technique is a method of handling interrupt requests in which multiple devices are connected in a chain, with each device passing the interrupt request to the next device in the chain. When an interrupt request is generated by a device,

it is sent to the first device in the chain, which checks if the interrupt is for that device. If it is not, the request is passed on to the next device in the chain until it reaches the correct device. Once the correct device is identified, it services the interrupt request and sends an acknowledgement signal back to the previous device in the chain, which then passes on the acknowledgement to the next device in the chain.

The daisy chaining technique ensures that interrupt requests are handled in a specific order, with the first device in the chain being serviced first. It is a simple and efficient method of handling interrupt requests and is commonly used in systems with a large number of devices. However, it can be slow and may result in delays in interrupt handling, particularly if the chain is long.

Priority and daisy chaining techniques are used in computer memory to manage interrupt requests from multiple devices. The priority technique ensures that the most critical interrupt requests are handled first, while the daisy chaining technique is a simple and efficient method of handling interrupt requests in systems with a large number of devices. The choice of technique depends on the specific system requirements and the number and priority levels of the devices.



CHECK YOUR PROGRESS

Q 4: What is interrupt-driven I/O?

.....

.....

Q 5: What are the advantages of interrupt-driven I/O?

.....

.....

Q 6: What are priority and daisy chaining techniques used for?

.....

.....



9.8 LET US SUM UP

- Memory read and write operations refer to the process of accessing and manipulating data in computer memory. These operations are critical to the functioning of computers and are used in all types of computing devices.
- The process of memory read operations is initiated by the processor by sending an address to the memory controller, which retrieves the data from the specified memory location and transfers it to the program. The process of memory write operations is initiated by the processor by sending an address and the new data to the memory controller, which stores the new value at the specified memory location.
- Handshaking refers to the process of establishing communication between the memory controller and the memory device. It involves a series of signals that are exchanged between the two devices to ensure that data is transmitted accurately and reliably.
- Polling is a technique used in computer memory to manage data transfer between the memory controller and the memory device. It involves the memory controller periodically checking the status of the memory device to determine if it is ready to transfer data. There are two primary polling techniques used in computer memory: serial polling and hub polling.
- Interrupt-driven I/O is an alternative to polling, where the memory device interrupts the processor when it is ready to transfer data. This can improve the efficiency of data transfer and reduce the load on the processor.
- Memory mapping is a technique used to address memory in a computer system, where each memory location is assigned a unique address. This can be used to map memory to specific devices or to provide access to memory for certain programs.
- Priority and daisy chaining techniques can be used to manage multiple

devices that need to access memory simultaneously, by assigning different priorities to each device and allowing them to access memory in a specific sequence.



9.9 FURTHER READINGS

1) Tanenbaum, A. S., & Woodhull, A. S. (2015). Operating systems: Design and implementation (3rd ed.). Pearson.

2) Hennessy, J. L., & Patterson, D. A. (2017). Computer architecture: A quantitative approach (6th ed.). Morgan Kaufmann.

3) Bryant, R. E., &O'Hallaron, D. R. (2015). Computer systems: A programmer's perspective (3rd ed.). Pearson.



9.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: Memory read operation is the process of retrieving data from computer memory.

Ans to Q No 2: Handshaking is essential in computer memory to ensure accurate and reliable data transfer.

Ans to Q No 3: Hub polling is a polling technique in which a hub device is used to manage the polling of multiple memory devices.

Ans to Q No 4: Interrupt-driven I/O is a technique used in computer memory to manage input/output operations using interrupts.

Ans to Q No 5: Interrupt-driven I/O allows for efficient, flexible, and reliable I/O operations, and allows the processor to handle multiple I/O requests simultaneously, improving system performance.

Ans to Q No 6: Priority and daisy chaining techniques are used to manage interrupt requests from multiple devices in computer memory, ensuring timely and efficient handling of the requests.



9.11 MODEL QUESTIONS

Q 1: Describe the process of memory read operations in detail. What is the role of the processor and the memory controller in this process? How is the retrieved data temporarily stored in the computer system?

Q 2: Explain the concept of handshaking in computer memory. Why is it essential for the memory controller and the memory device to be synchronized during data transfer? What are the primary signals involved in the handshaking process?

Q 3: Compare and contrast the two primary polling techniques used in computer memory: Serial polling and Hub polling. What are the advantages and disadvantages of each technique? In which type of systems is each technique most suitable?

Q 4: How do memory write operations work? What is the role of the processor and the memory controller in this process? What happens after the new data is stored in memory? How are memory write operations critical to the functioning of modern computers?

Q 5: Describe the interrupt-driven I/O technique in computer memory. How does it work, and what are the advantages of using this technique? How does it compare to the polling technique?

Q 6: Explain the priority and daisy chaining technique used in computer memory. What is the purpose of assigning priorities to different devices? How does daisy chaining help to simplify the communication between devices in a computer system?

*** ***** ***

UNIT 10: MEMORY AND I/O INTERFACING

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 I/O Interface

- 10.3.1 Functions of I/O Interfaces
- 10.3.2 Connection of I/O Buses to I/O Devices

10.4 I/O Processor

- 10.4.1 Block Diagram of I/O Processor
- 10.4.2 CPU- I/O Processor Communication

10.5 Priority Encoder

- 10.5.1 Definition of Encoder
- 10.5.2 Priority Encoder and its features
- 10.5.3 Applications of Priority Encoder

10.6 I/O Scheduling

- 10.6.1 Techniques of I/O Operations
- 10.6.2 Types of I/O Scheduling Techniques

10.7 Let Us Sum Up

10.8 Further Reading

10.9 Answers to Check Your Progress

10.10 Model Questions

10.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about I/O interface and its different functions
- learn how I/O buses are connected to I/O devices
- learn about I/O Processor and its functinality with CPU
- learn about Priority encoder and its various features and applications
- learn about different types of I/O operations
- learn about I/O scheduling techniques and its different types.

10.2 INTRODUCTION

We know that input-output subsystem of a computer system, referred to as I/O, provides an efficient mode of communication between the central system of computer and the outside environment. Programs and data must be entered first into computer memory for processing and results obtained from computations must be recorded or displayed for outer world. Keyboard is the most common input device for entering information into a computer system that allows a user to enter alphanumeric information directly. On the other hand the CPU is an extremely fast device capable of performing operations at very high speed. I/O devices attached to the computer system are also known as peripherals. There are three types of peripherals such as input, output and input-output peripherals. These I/O devices may be analog or digital, serial or parallel. These peripherals are electromagnetic and electromechanical devices of some complexity. Video monitors are the most common used peripherals. Printers provide a permanent record on paper of computer output data or text. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.

Some important characteristics of I/O devices are given below:

- They are shared by multiple programs.
- They are interrupt driven and low-level control is complex.

10.3 I/O INTERFACE

Input-output interface provides a method for transferring information between internal storage and external I/O devices. I/O devices connected to a computer system need special communication links for interfacing them with the CPU. The purpose of the communication link is to resolve the differences that exist between the CPU and each I/O device. The major issues are given below:

a) Peripherals are electromagnetic and electromechanical devices and their manner of operation is different from the operation of the CPU and memory which are electronic devices. Therefore, a conversion of signal values may be required.

b) The data transfer rate of I/O devices is usually slower than the transfer rate of the CPU and as a result, a synchronization mechanism may be needed.

c) Data codes and format in peripherals differ from the word format in the CPU and memory.

d) The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other I/O devices connected to the CPU.

To resolve these differences, the computer system includes special hardware components between the CPU and I/O devices to supervise and synchronize all I/O transfers. These components are called interface units because they interface between the (CPU) processor bus and the I/O devices. In digital computer system the interface is referred as a complementary set of signal connection points between two parts of a system. Basically, there are two types of interfaces which are (i) CPU interface that correspond to the system bus and (ii) I/O interface that depends on the nature of I/O device.

10.3.1 Functions of I/O Interfaces

The main functions of I/O interfaces are given below:

a) Data conversion: Data conversion refers to conversion between digital and analog signals.

b) Synchronization: It refers to matching operation speed of CPU and other I/O devices.

c) Device selection: It refers to the selection I/O device by CPU in a queue manner. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

10.3.2 Connection of I/O Bus to I/O Devices

I/O bus consists of data lines, address lines and control lines. Each I/O device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus and

interprets them for the I/O devices as well as provides signals for the I/O controller. It also synchronizes the data flow and supervises the transfer between I/O devices and CPU. Each I/O device has its own controller that operates the particular electromechanical device.

The CPU is attached to all peripheral interfaces through the I/O bus. To communicate with a particular device, the CPU places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address then it activates the path between the bus lines and the device that it controls. All I/O devices whose address does not correspond to the address in the bus are disabled by their interface.

At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selects respond to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached I/O units. The interpretation of the command depends on the I/O device that the CPU is addressing. There are four type of commands that an interface may receive, which are– (i) control command, (ii) status command, (iii) data output command and (vi) data input command.

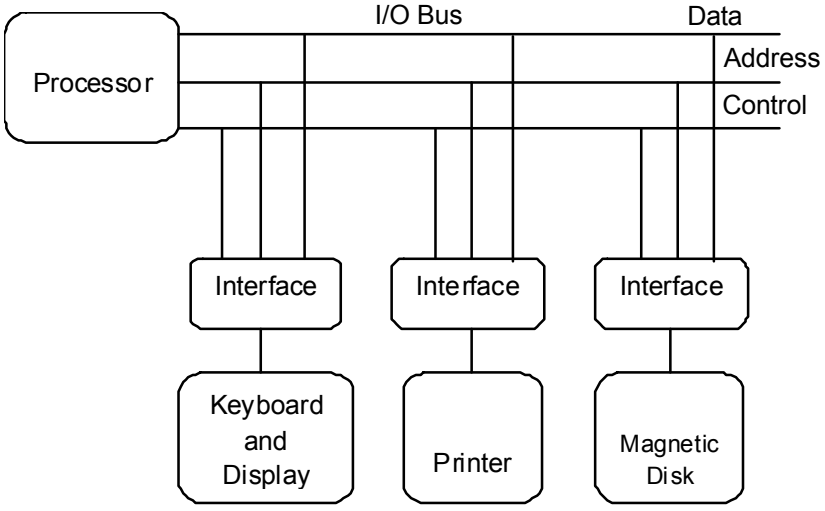


Figure 10.1: Connection of I/O Bus to I/O Devices through Interfaces

10.4 I/O PROCESSOR

An input-output processor (IOP) is a processor with direct memory access capability. In this case, the computer system is divided into a memory unit and number of processors. Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

10.4.1 Block Diagram of I/O Processor

Here the block diagram of a computer along with various I/O Processors is given below. The memory unit occupies the central position and can communicate with each processor. The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program.

The IOP operates independent from CPU and transfer data between peripherals and memory.

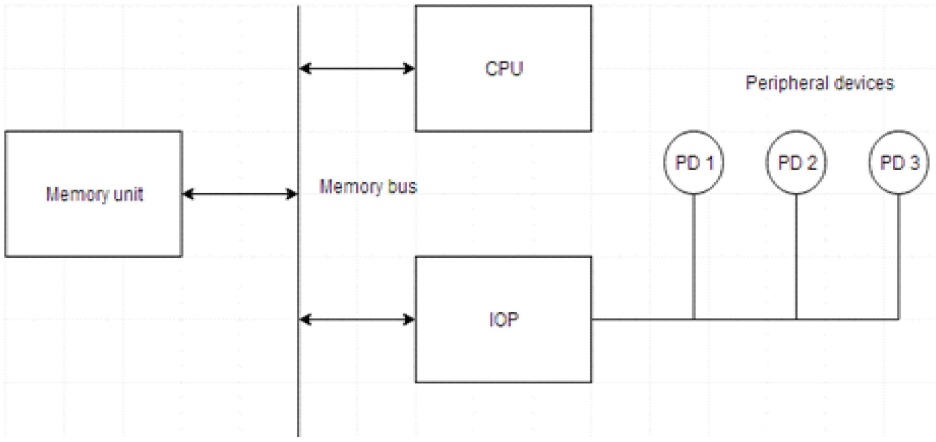


Figure 10.2: Block Diagram of I/O Processor

The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation. The

CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt. Instructions that are read from memory by an IOP are also called *commands* to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory as well as when it is time to execute the I/O program.

10.4.2 CPU–I/O Processor Communication

The communication between CPU and I/O processor may take various forms, 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. First CPU sends an instruction to test the I/O processor path. The I/O processor responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the I/O processor and I/O devices. If all is in order, then CPU sends the instruction to start I/O transfer. CPU can also continue with another program while the I/O processor is busy with I/O program. When I/O processor 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 I/O processor. Then the I/O processor responds by placing the contents of its status report into specified memory location.

The I/O processor takes care of all data transfer between different I/O units and the memory while CPU is processing another program. The I/O processor and CPU are competing for the use of memory so that the number of devices can be in operation is limited by the access time of the memory.

10.5 PRIORITY ENCODER

The Priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having high priority will take precedence.

10.5.1 Definition of Encoder

The general definition of an encoder is “An encoder is a device, circuit, transducer, software program, algorithm or person that converts information from one format or code to another, for the purposes of standardization, speed or compression.” Now, talking about a digital encoder, the Digital Encoder is a combinational circuit that generates a specific code at its outputs such as binary or BCD in response to one or more active inputs.

Digital encoders produce outputs depending on the number of data input lines. An “n-bit” binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 encoders.

10.5.2 Priority Encoder and Its Features

The Priority Encoder allocates a priority level to each input. The priority encoders output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored. Priority encoder is a circuit that converts multiple binary inputs into binary representation of the index of active input bit with the highest priority. Each of input has assigned priority. The least significant bit has the highest priority and the most significant bit the lowest. If more than one input is active at the same time the input having highest priority will take precedence. The priority encoder component has an additional output indicating that output signal is valid when at least one input is active. Priority encoders are often used when multiple components are to share common resource or in interrupt controllers.

Using a parallel prefix computation (PPC) speeds up calculation and improves the circuit timing.

We may also define in another terms as the Priority encoder is a circuit that converts multiple binary inputs into a binary representation of the index of active input bit with the highest priority. Each of input has assigned priority where the least significant bit has the highest priority and the most significant bit the lowest. If more than one input is active at the same time the input having highest priority will take precedence. Additional output signal indicates that output grant is valid.

A priority encoder provide n bits of binary coded output representing the position of the highest order active input of 2^n inputs. If two or more inputs are high at the same time, the input having the highest priority will take precedence.

Some important features of Priority Encoder is given below:

- Parameterized priority encoding capable of both small and large (up to 8k) input width.
- Valid output indicating the output is valid.
- Large encoders built as a structure of smaller encoders.
- Simple or fast parallel implementation using parallel prefix computation.
- Optional pipeline registers on the output of each stage of hierarchy.
- Small building block width selected independently from total input width controlling the number of hierarchy stages and latency of pipeline versions.
- Clock enable inputs for pipeline stages.

10.5.3 Priority Encoder and Its Applications

Some important applications of priority encoder are given below:

- Priority encoders can be used to reduce the number of wires

needed in a particular circuits or application that have multiple inputs. For example, we may assume that a microcomputer needs to read the 104 keys of a standard QWERTY keyboard where only one key would be pressed either “HIGH” or “LOW” at any one time.

- Another more common application of priority encoder is in magnetic positional control as used on ships navigation or for robotic arm positioning etc. For example, the angular or rotary position of a compass is converted into a digital code by a 74LS148 8-to-3 line priority encoder and input to the systems computer to provide navigational data. Magnets and reed switches could be used at each compass point to indicate the needles angular position.
- Other applications especially for Priority Encoders may include detecting interrupts in microprocessor applications. In this case the microprocessor uses interrupts to allow peripheral devices such as the disk drive, scanner, mouse, or printer etc, to communicate with it, but the microprocessor can only “talk” to one peripheral device at a time so needs some way of knowing when a particular peripheral device wants to communicate with it.

  The processor does this by using “Interrupt Requests” or “IRQ” signals to assign priority to all the peripheral devices to ensure that the most important peripheral device is serviced first. The order of importance of the devices will depend upon their connection to the priority encoder.

10.6 I/O SCHEDULING

Input/output (I/O) scheduling is the method that computer’s operating systems use to decide in which order the block I/O operations will be submitted to memory or storage volumes. I/O scheduling is sometimes called disk scheduling. I/O scheduling usually has to work with hard disk drives that have long access times for requests placed far away from the

current position of the disk head.

I/O schedulers can have many purposes depending on the goals. Some important common purposes are given below:

- To minimize time wasted by hard disk seeks.
- To guarantee that certain requests will be issued before a particular deadline occurs.
- To prioritize a certain processes' I/O requests.
- To give a share of the disk bandwidth to each running process.
- To use hardware efficiently.

Suppose the I/O request issues a system call to the OS. ‰If desired disk drive or controller is available, request is served immediately. ‰If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the OS has to choose which pending request to service next.

10.6.1 Techniques of I/O Operations

There are three techniques for performing I/O operation which are:

i) Programmed I/O,
ii) Interrupt-driven I/O and
iii) Direct Memory Access (DMA)

i) Programmed I/O: In this case, the processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding.

ii) Interrupt-driven I/O: In this case of I/O operations, the processor issues an I/O command on behalf of a process in two cases: *non-blocking* and *blocking*.

In the case of non-blocking, the processor continues to execute instructions from the process that issued the I/O command. But for blocking case, the next instruction the processor executes is

from the OS, which will put the current process in a blocked state and schedule another process.

iii) Direct Memory Access (DMA): In this case of I/O operation, a DMA module controls the exchange of data between main memory and an I/O module without hampering the working load of a processor. There are mainly three steps in DMA transfers, which are given below:

- Processor sets up the DMA transfer by supplying identity of device, operation to perform, starting memory address that is source or destination of data, number of bytes to be transferred.
- DMA controller starts the operation (arbitrates for the bus, supplies address, reads or writes data), until the entire block is transferred.
- DMA controller interrupts the processor, which then takes the necessary actions.

10.6.2 Types of I/O Scheduling Techniques

Although there are lots of other algorithms that reduce the seek time of all requests, here we will only concentrate on the following I/O scheduling algorithms.

i) First Come-First Serve (FCFS)

ii) Shortest Seek Time First (SSTF)

iii) Elevator (SCAN)

iv) Circular SCAN (C-SCAN)

i) First Come-First Serve (FCFS): All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it

took to move from one request to the next.

ii) Shortest Seek Time First (SSTF): In this case, request is serviced according to next shortest distance. The process would continue until all the process is taken care of. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.

iii) Elevator (SCAN): This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.

iv) Circular SCAN (C-SCAN): In this Circular scanning technique it works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement.



CHECK YOUR PROGRESS

Q.1: State whether the following statements are true (T) or false (F):

a) I/O devices attached to the computer system are also known as peripherals.

b) I/O devices are not electromagnetic and electromechanical devices of some complexity.

c) The data transfer rate of I/O devices is usually slower than the transfer rate of the CPU.

d) Device selection: It refers to the selection I/O device by CPU in a stack manner.

e) Data codes and format in peripherals differ from the word format in the CPU and memory.

f) The computer system includes special hardware components between the CPU and I/O devices to supervise and synchronize all I/O transfers known as interface.

g) I/O bus consists of only data lines and address lines.

h) The Priority encoder is a circuit that implements the priority function.

i) Input/output (I/O) scheduling is the method that computer's operating systems use to decide in which order the block I/O operations.

j) A DMA module controls the exchange of data between main memory and an I/O module hampering the working load of a processor.



10.7 LET US SUM UP

- We know that input-output subsystem of a computer system, referred to as I/O, provides an efficient mode of communication between the central system of computer and the outside environment.
- I/O devices attached to the computer system are also known as peripherals. There are three types of peripherals such as input, output and input-output peripherals.
- Input-output interface provides a method for transferring information between internal storage and external I/O devices. I/O devices connected to a computer system need special communication links for interfacing them with the CPU.
- Basically, there are two types of interfaces which are (i) CPU interface that correspond to the system bus and (ii) I/O interface that depends on the nature of I/O device.
- An input-output processor (IOP) is a processor with direct memory access capability. Each IOP controls and manage the input-output

tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions.

- The I/O processor (IOP) takes care of all data transfer between different I/O units and the memory while CPU is processing another program.
- The Priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having high priority will take precedence.
- We may also define in another terms as the Priority encoder is a circuit that converts multiple binary inputs into a binary representation of the index of active input bit with the highest priority.
- A priority encoder provide n bits of binary coded output representing the position of the highest order active input of 2^n inputs.
- Input/output (I/O) scheduling is the method that computer's operating systems use to decide in which order the block I/O operations will be submitted to memory or storage volumes. I/O scheduling is sometimes called disk scheduling.
- There are three techniques for performing I/O operation which are: (i) Programmed I/O, (ii) Interrupt-driven I/O and (iii) Direct Memory Access (DMA).
- In the programmed I/O, the processor issues an I/O command on behalf of a process to an I/O module; that process then busy waits for the operation to be completed before proceeding.
- In interrupt-driven I/O, for the I/O operations, the processor issues an I/O command on behalf of a process in two cases: (i) non-blocking and (ii) blocking.
- DMA module controls the exchange of data between main memory and an I/O module without hampering the working load of a processor.
- Although there are lots of other algorithms that reduce the seek time of all requests, Some I/O scheduling algorithms are:
  a) First Come-First Serve (FCFS)
  b) Shortest Seek Time First (SSTF)
  c) Elevator (SCAN)
  d) Circular SCAN (C-SCAN)



10.8 FURTHER READING

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI.
2) Hammacher, Carl. (Fifth Edition, 2002). *Computer Organization*. McGraw Hill International Edition.
3) Rajaraman, V. (Fouth Edition, 2008). *Fundamentals of Computers*.
4) Stallings, William. (2004). *Computer Organization and Architecture: Designing for Performance*. PHI.



10.9 ANSWERS CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) True, b) False, c) True, d) False, e) True, f) True, g) False, h) True, i) True, j) False



10.10 MODEL QUESTIONS

Q.1: What do you mean by I/O devices? Explain different I/O devices.

Q.2: What is I/O interface? Describe different issues and functions of I/O interfaces in the computer system.

Q.3: Explain with block diagram how does the connection of I/O bus to I/O devices work.

Q.4: What is I/O processor? Explain its working principle.

Q.5: What do you mean by priority encoder? Briefly discuss the Significance of priority encoder in digital computer system.

Q.6: Explain different techniques of I/O operations.

Q.7: What is I/O scheduling technique? Explain its different types.

*** ***** ***

UNIT 11: EXTERNAL MEMORY

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Magnetic Disk

11.4 Magnetic Read-Write Operation

11.5 Track

11.6 Sector

11.7 Inter Track and Inter Sector Gap

11.8 Cylinder

11.9 Fixed Head Disk

11.10 Seek Time

11.11 Access Time

11.12 Transfer Time

11.13 Rotational Delay

11.14 RAID

11.15 Optical Memory

11.16 Magnetic Tap

11.17 Let Us Sum Up

11.18 Further Reading

11.19 Answers to Check Your Progress

11.20 Model Questions

11.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about different external memory
- explain read-write operation
- describe different parts associated with external memory like track, sector, cylinder, etc.
- learn about seek time, access time and rotational delay
- learn about RAID, optical memory etc.

11.2 INTRODUCTION

In the previous unit we have learnt about the I/O processor and the interfacing of the I/O processors with the different devices. Also we have gone through the input and output devices.

In this unit, we will describe secondary storage devices, magnetic disk, optical memory, magnetic tape and various terms associated with these memories.

The speed and capacity of the memory impacts the performance of the computer. Magnetic disk is a random access medium. That is the data on the disk can be found almost immediately rather than sequentially going through the data. The data on the disk can be erased and reused virtually indefinitely. It is designed to reside in a protective case or cartridge to protect it from dust and other external interference.

The magnetic tape is another important storage device that is a sequential one. Also, the optical storage technology is important in all types of computer systems. The CD and the DVDs have also become increasingly important.

11.3 MAGNETIC DISK

Magnetic disks are the widely used as secondary storage device. They are mainly used for bulk storage. Unlike the magnetic tapes, the data on the magnetic disks can be read randomly. A magnetic disk is a thin, circular metal plate the surface is coated with a magnetic oxide layer usually on both the sides. These have a high storage capacity, reliable and have capability to access the stored data directly. The disks are mounted on a rotating shaft. Each magnetic disk has a head which consists of a magnetizing coil and a magnetic yoke. Information is stored on the disks which are represented as magnetic spots. Presence of a magnetic spot represents a 1 bit and the absence of a magnetic spot represents a 0 bit.

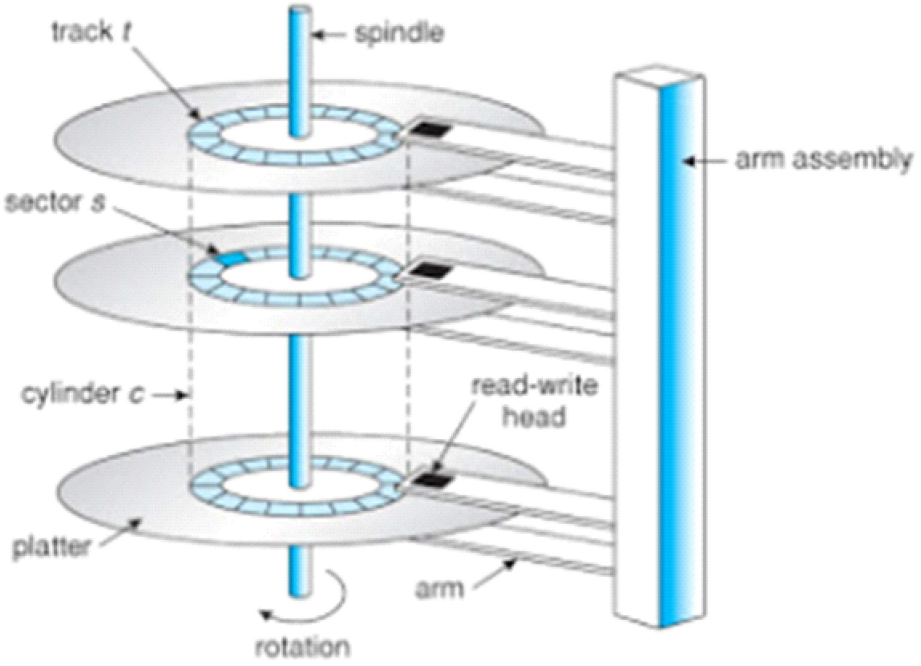


Figure 11.1

The individual disks are spaced on the shaft to allow room for a read/write mechanism to move between them.

Information is stored on the disk by magnetizing the surface in a particular direction as shown in the figure below.

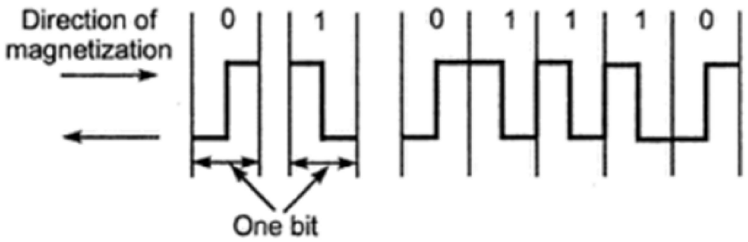


Figure 11.2: Bit Representation of Magnetic Disk

In most disk storage devices, several disks are assembled to form a disk pack as shown below:

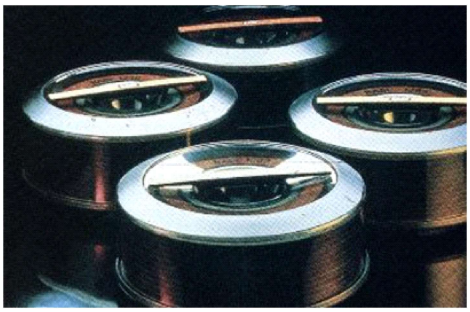


Figure 11.3

The top and bottom surfaces are not used for storing data because they are likely to become scratched or nicked. A disk pack may contain anywhere from five to one hundred disks.

The advantage of the magnetic disk over the magnetic tapes is that the access time is much faster.

11.4 MAGNETIC READ WRITE OPERATIONS

The data on a disk is read or written by the read/write heads located between the disks. Most disk units have one read/write head for each disk recording surface. All the heads are permanently connected to an access mechanism.

In some systems, there are two heads– a read head and a write head. During a Read/Write operation, the head is stationary and the platter rotates beneath it.

A read/write mechanism is used to achieve the access to read or write on the magnetic disk. When the device is used to read from a particular magnetic medium, it detects the individual magnetized areas and translates them into electrical pulses. In performing the write operations, the device magnetizes the appropriate areas on the magnetic medium thereby erasing the previously stored data, if any. Also, the electrical impulses from the

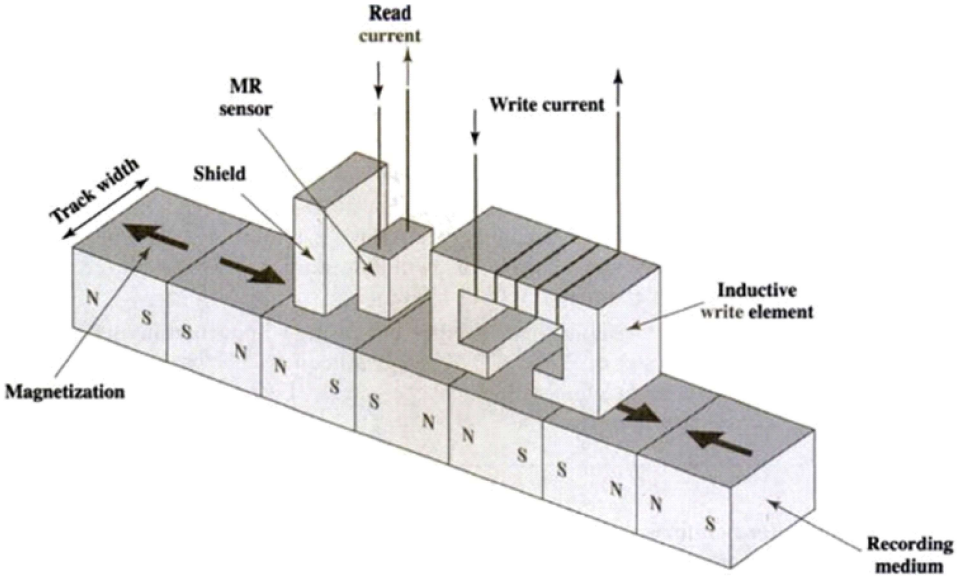


Figure 11.4

processor are converted into magnetic spots on the disks for writing to it. When electric pulses are passed through the write head, a magnetic field is produced. These magnetic patterns are recorded on the surface of the disk below creating different for the positive and the negative currents. If the direction of the current is reversed, the direction of the magnetization of the recording medium is also reversed.

The figure above shows the inductive Read/Write head. During a read operation, a magnetic field moving relative to the coil produces an electrical current in the coil. When the surface of the disk passes under the head, it produces the same current that was recorded.

The head consists of a partially shielded magnetoresistive (MR) sensor. By passing a current through the MR sensor, the resistive changes are detected as voltage signals.



Figure 11.5

11.5 TRACK

The data on the magnetic disk is organized in a concentric set of circles called track. There can be from a few hundred to a few thousand tracks on a single disk surface. Each of the tracks has the same width as the head and the adjacent tracks are separated by gaps. Data is stored serially on each track. The tracks are numbered from the outermost to the innermost circle starting with zero.

Each track on a disk can store the same amount of data even though the tracks get smaller toward the center of the disk. This results higher bit density in inner tracks than that of the outer tracks. Consider a disk pack with 4,000 usable tracks (20 surfaces x 200 tracks per surface) on which 7,294 characters can be stored on each track. The disk pack could conceivably store 29,176,000 characters of data (4,000 tracks x 7,294 characters per track).

11.6 SECTOR

Each track is divided into sectors. The sectors refers to the number of fixed size areas (imaginary pie slices) that can be accessed by one of the read/write heads, in one rotation of the disk, without the head having to change its position. The number of bytes stored in each sector is same. Thus, the size of the inner sectors decides the storage capacity for all other sectors on the disk. Sectors are typically 512 bytes in size, although there is an initiative to increase the sector size to 4096 bytes.

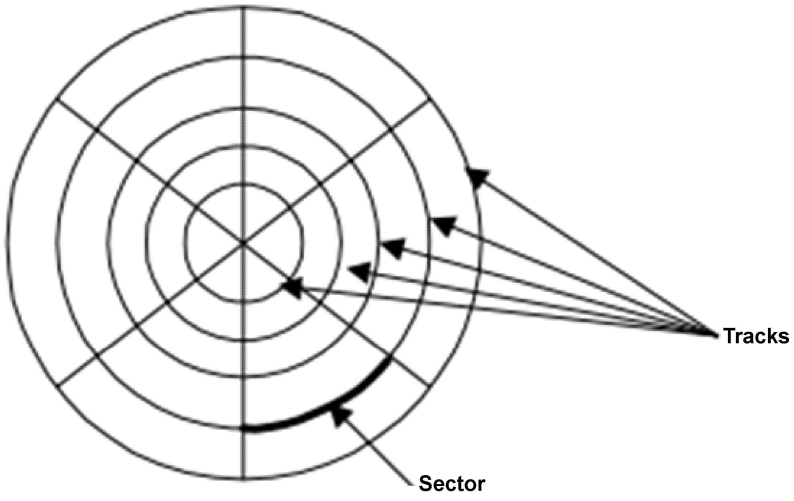


Figure 11.6

11.7 INTER TRACK AND INTER SECTOR GAP

The gap in between the tracks in a magnetic disk is called the Inter-track gap and the gap between the sectors in the disks is called inter-sector gap. The inter-track gaps help protects data integrity. To avoid, imposition

unreasonable precision requirements on the system, adjacent sectors are separated by inter-track (inter-sector) gaps.

Some means are needed to locate sector positions within a track. Clearly there must be some starting points on the track and a way of identifying the start and end of each sector.These requirements are handled by means of a control data recorded on the disk. Thus, the disk is formatted with some extra data used only by the disk drive and not accessible to the user.

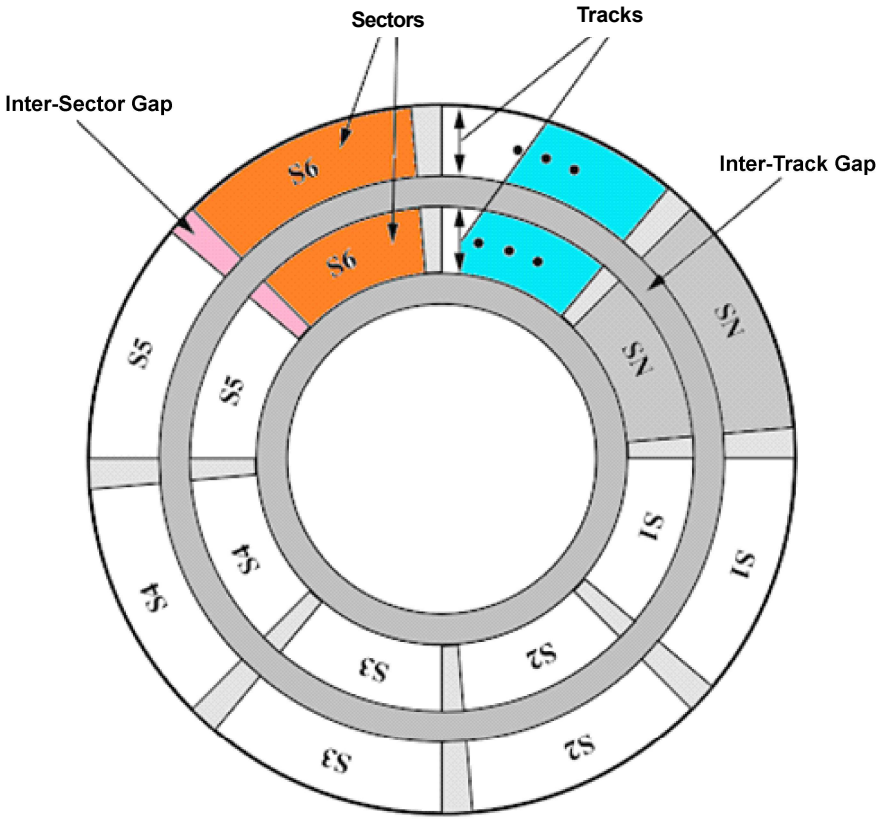


Figure 11.7

11.8 CYLINDER

To have a large storage system in the computer, multiple magnetic disks are maintained and used together. The disks or the platters are stacked on top of each other with a common spindle. The spindle turns or rotates the disks of the pack.

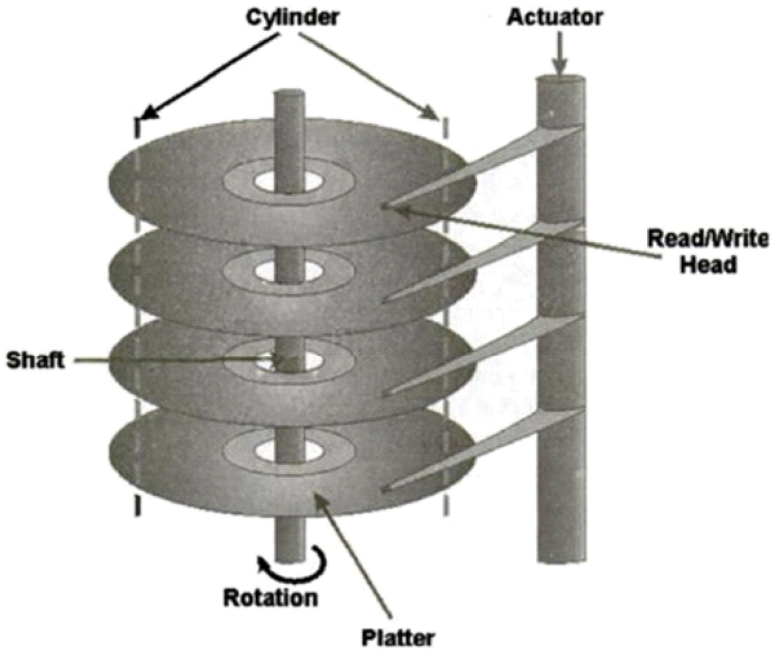


Figure 11.8

There is a gap between the platters so that there is space for the magnetic read/write head. For each side of each platter, there is a magnetic read/write head. The head is mounted on arms which can move the heads towards the central spindle or towards the edge. This concept of stacking the disks is called cylinder. A cylinder is made up of all the tracks of the same number from all the metal disks. Thus cylinder is an imaginary view of all the matching tracks on top of each other as shown in the figure above. Thus the storage capacity of a magnetic disk can be defined as the following:

Storage capacity of a magnetic disk = Number of recording surfaces
x Number of tracks per surface
x Number of sectors per track
x Number of bytes per sector

11.9 FIXED HEAD DISK

There are two types of disks depending upon the two methods of accessing data. One is moving head and the other is fixed head.

A fixed head disk has many read/write heads on each disk surface as the number of tracks which are mounted on the access arm as shown in the figure below.

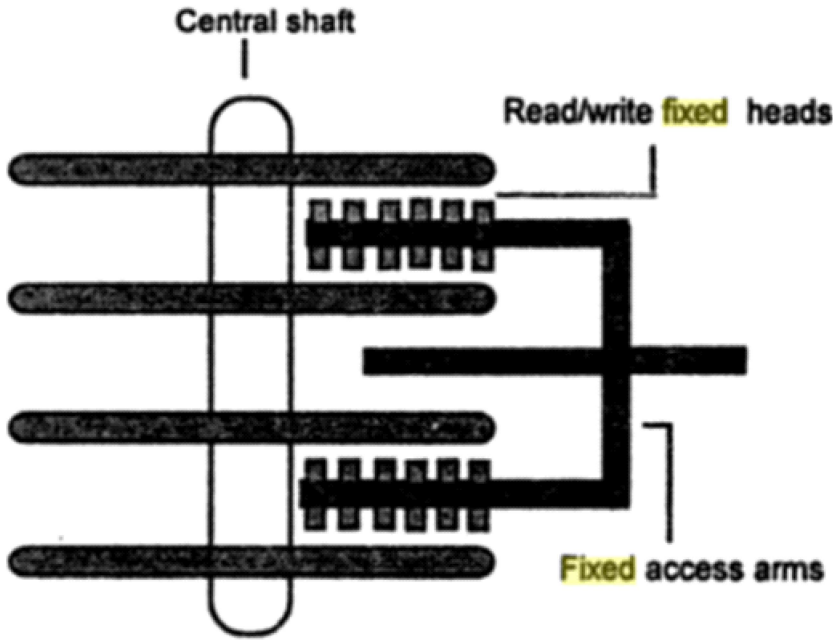


Figure 11.9

The head does not move as each track has a separate read/write head. The moving head disk systems on the other hand have one read/ write head for each disk surface which is mounted on the access arm. The head in such systems moves horizontally on the disk surface to read or write data.

The access speed of the fixed head disks are more as compared to the moving head disks. In a fixed head disk, the track or the cylinder is selected by electronically switching to the appropriate read/write head rather than by mechanically moving it. This makes it faster.

However, as the cost of the additional read/write heads is quite high, so the fixed head disks are not commonly used.



CHECK YOUR PROGRESS

Q.1: Fill In the blanks:

i) Magnetic disk has a head which consists of a ..... and a .....

ii) The data on a disk is read or written by the ..... heads located between the disks.

iii) During a read operation, a magnetic field moving relative to the coil produces an ..... in the coil.

iv) The head consists of a partially shielded ..... sensor.

v) The data on the magnetic disk is organized in a concentric set of circles called .....

vi) Each track on a disk can store the same amount of data even though the tracks get ..... toward the center of the disk.

vii) Tracks are divided into .....

viii) The gap in between the tracks in a magnetic disk is called the ..... and the gap between the sectors in the disks is called .....

11.10 SEEK TIME

The time required to position the read/write head from the current track or cylinder to the desired track or cylinder is called seek time. The seek time varies depending upon the position of the head when a data request is made. It is zero when the head is already positioned on the specified cylinder.

The seek time is maximum when the read/write has to move from the outermost track to the innermost track or vice-versa.

11.11 ACCESS TIME

Access time is the time taken to read a record from the magnetic disk. In other words, it is the time elapsed between the initiation of a particular request for data and the receipt of the data.

The sum of the seek time and the rotational latency is the access time. The average disk access time is equal to seek time + average rotational delay+ transfer time + controller overhead.

11.12 TRANSFER TIME

Once the read/write head is in position, the read or write operation can be performed as the sector moves under the head. This is the data

transfer operation of the disk. The transfer time is the time taken to transfer a block of bits. It is a function of the sector size, rotation speed, and the recording density of a track.

The transfer time to or from the disk depends on the rotation speed of the disk. The transfer time, T can be calculated as

$$T = \frac{b}{rN}$$

Where, T = transfer time

b = number of bytes to be transferred

N = number of bytes on a track

r = rotation speed

Thus, the Total average access time can be expressed as

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

here, T_S is the average seek time.

11.11 ROTATIONAL DELAY

Seek time as we have seen is the time taken to position the head at the track. After the track is selected the disk controller waits until the appropriate sector rotates to line up with the head. This time which is taken for the beginning of the sector to reach the head is called rotational delay or rotational latency.

11.12 RAID

RAID is ***Redundant Array of Independent Disks***. Originally it was known as Redundant Array of Inexpensive Disks. It is a way of storing data in different places on multiple disks to protect the data in case of a drive failure. It is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both. The data recording is distributed on the different drives which appear as a single drive to the application program.

The access speed of the magnetic disks is considerably slow as compared to the speed of the processor and the main memory. By improving

the disk performance the overall performance of the computer can be increased. This led to the development of arrays of disks that operate independently and in parallel. The same data is kept in multiple disks. With the multiple disks, separate I/O requests can be handled in parallel provided the data that is required is on the separate disks. Also, a single I/O request can be handled in parallel as long as the data to be accessed is in the multiple disks.

In a RAID, a single large file is stored in several different disks by breaking up the file into a number of smaller pieces and storing the pieces on the separate disks. This is called ***data stripping.*** Thus the file modules can be accessed simultaneously. This increases the access speed and the data transfer speed. Thus the transfer time would be the time that would be required in a single disk divided by the number of disks used in the array. A data stripe can be any size between 512 bytes up to several megabytes.

There are a number of ways in which the data can be organized with the use of multiple disks. These are called as the RAID levels. These levels do not imply a hierarchical relationship. They are different design architectures. The RAID levels use these stripes and the multiple disks in different ways.

RAID 0: The RAID level 0 uses the data stripping as shown in the figure above. A number of disks are connected together to form a single disk.

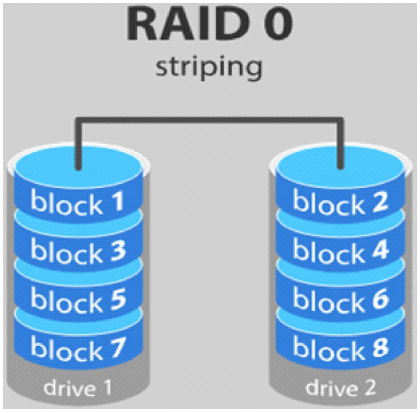


Figure 11.10

In this level, the data are split up into blocks that are written across all the drives in the array. The multiple disks here offer better I/O performance and also a disk system with higher capacity is realized.

RAID 0 does not provide fault handling capacity because if one disk fails, the entire disk system fails.

RAID 1: RAID 1 uses mirroring. It uses pairs of stripped disks and the data is written to both the disk in the pair at once.

If one disk controller is used for both the disk in each pair, the process is called ***mirroring***, and if a separate disk controller is used for each disk in the pair it is called ***duplexing***.

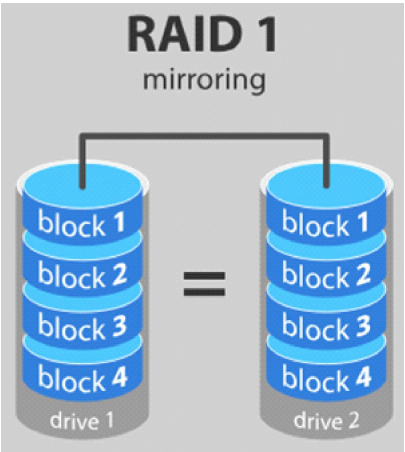


Figure 11.11

RAID 1 is a highly reliable system. If one of the drive fails, then the controller uses either the data drive or the mirror drive for data recovery.

RAID 2: In RAID 2, the data striping is used as in RAID 0 and RAID 1. The data is stripped across two or more disks and one or more extra disk are used to hold error correcting code for the stored data. Thus, if one of the disk crashes, the data of that disk can be regenerated using the appropriate error correcting code held on another disk.

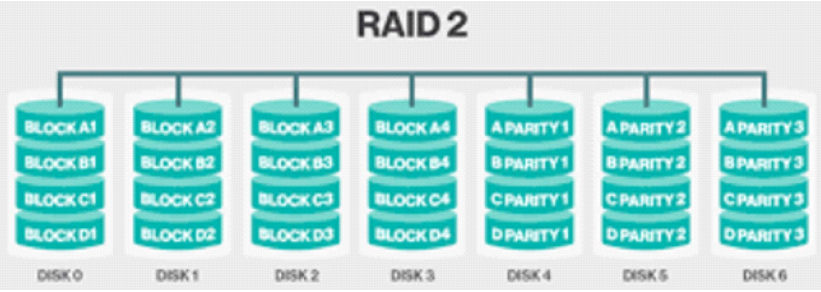


Figure 11.12

These systems are expensive mainly because of the error correcting code that has to be stored. They are rarely used.

RAID 3: This is similar to RAID 2. The data is stripped across a number of disks and an extra disk is used to hold the parity information. Instead of an error correcting code, a simple parity bit is computed for the set of individual bits in the same position. In the event of a disk failure, the data on the failed disk can be revived from the surviving data and parity information.

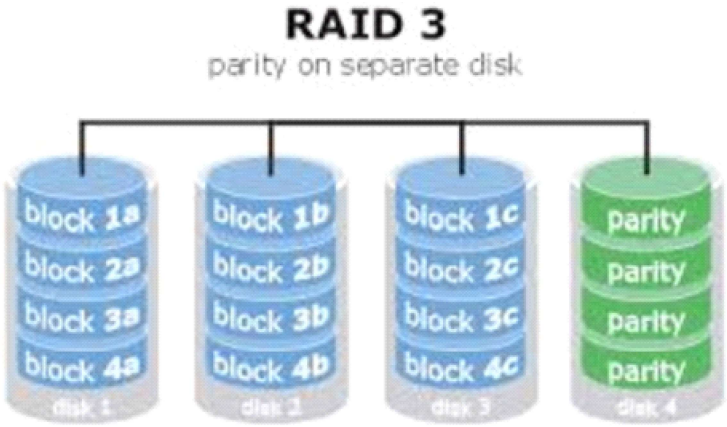


Figure 11.13

RAID 4: RAID 4 consists of a data disks and a parity disk. This level is similar to RAID level 3 because similar stripping method is used. But it employs stripped data in much larger blocks or segments.

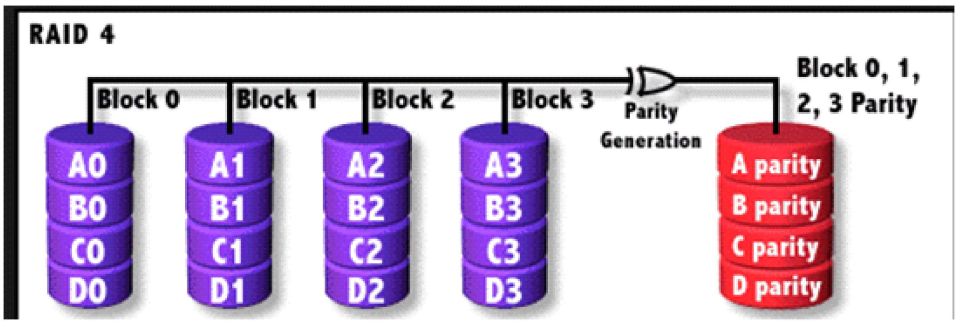


Figure 11.14

As all the parity data is written in to one single disk, so RAID level 4 suffers from a write bottleneck and is not eneally used.

RAID 5: RAID 5 also uses disk stripping with parity. In this configuration, the data and the parity are stripped evenly across all of the disks. So no single disk is a bottleneck. Striping also allows users to reconstruct data in case of a disk failure.

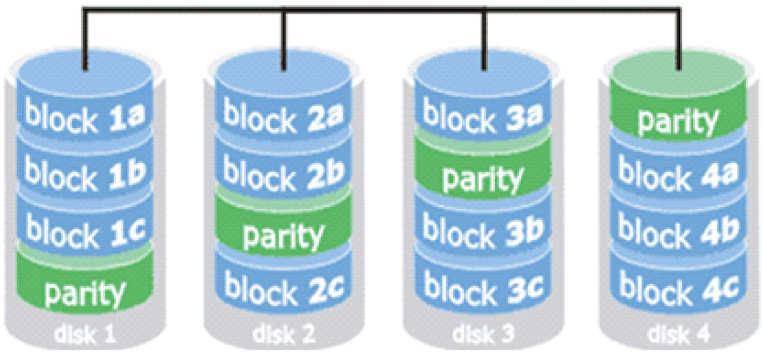


Figure 11.15

RAID 5 used parity instead of mirroring for data redundancy. When data is written to a RAID 5 drive, the system calculates parity and writes that parity into the drive. While mirroring maintains multiple copies of data in each volume to use in case of failure, RAID 5 can rebuild a failed drive using the parity data, which is not kept on a fixed single drive.

RAID 6: As like RAID 5, the RAID level 6 does block level striping. However, it uses dual parity. It creates two parity blocks for each data block. With RAID 6, the RAID array will even survive that second failure.

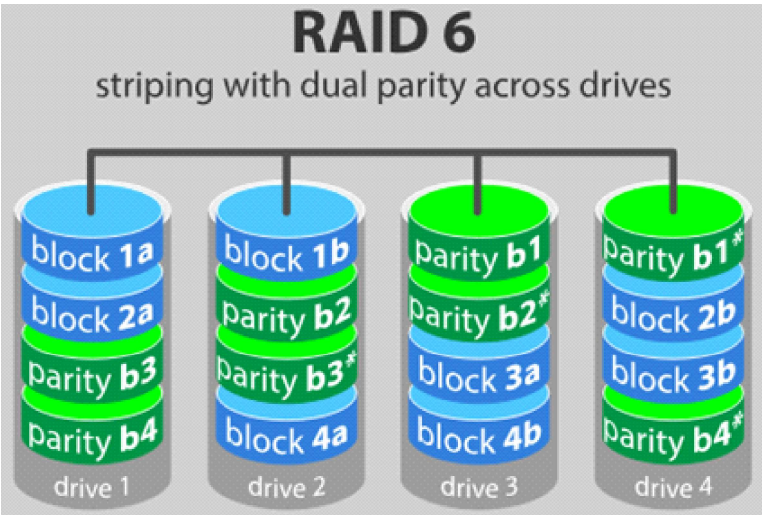


Figure 11.16

If two drives fail, the data can still be accessed, even while the failed drives are being replaced. So RAID 6 is more secure than RAID 5. RAID 6 is a good all-round system that combines efficient storage with excellent security and decent performance. It is preferable over RAID 5 in file and application servers that use many large drives for data storage.

11.11 OPTICAL MEMORY

In 1983, the first optical memory device CD (Compact Disc) was successfully launched. The huge success of CD encouraged the development of optical storage technology.

CD (Compact Disk) and DVD (Digital Versatile Disk) are common examples of optical memory.

Compact Disk: Information is digitally recorded on the compact disk. The recording is done in the form of microscopic dots on the surface of the polycarbonate. It is done by focusing a laser beam on the surface of the disk. The pitted surface is then coated with a highly reflexive surface usually aluminum or gold. The shiny surface protects against dust and scratches by a top coat of clear acrylic.

To record data in a compact disk, the high-intensity laser beam is turned On and Off according to a digital signal. This causes tiny holes or pits are burnt into the disk. This is shown in the figure below:

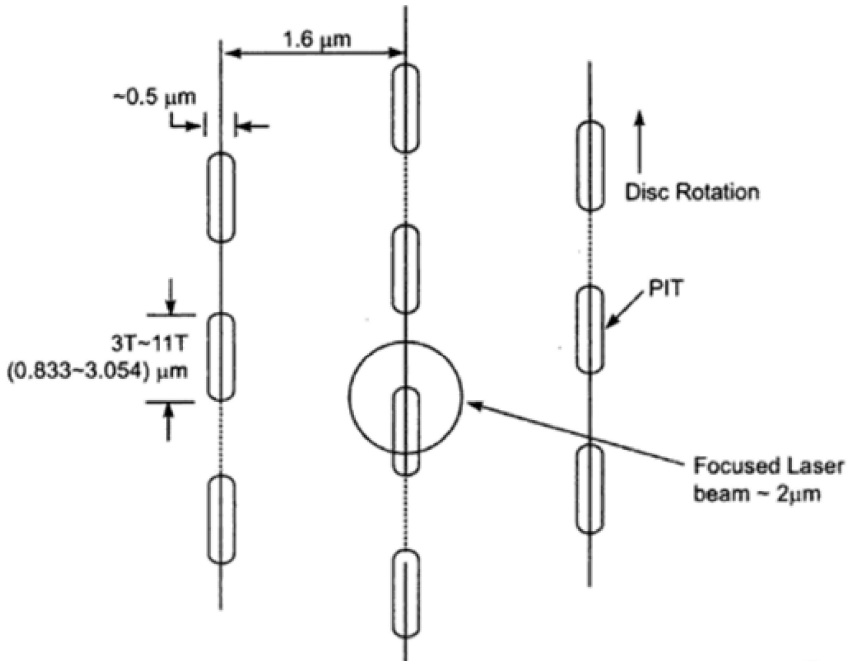


Figure 11.17: Recorded Portion of CD Showing Pits along its Tracks

The recorded data is recovered from the optical disk with an optical pickup, which moves across the surface of the rotating disk. The pickup system is shown in the figure below:

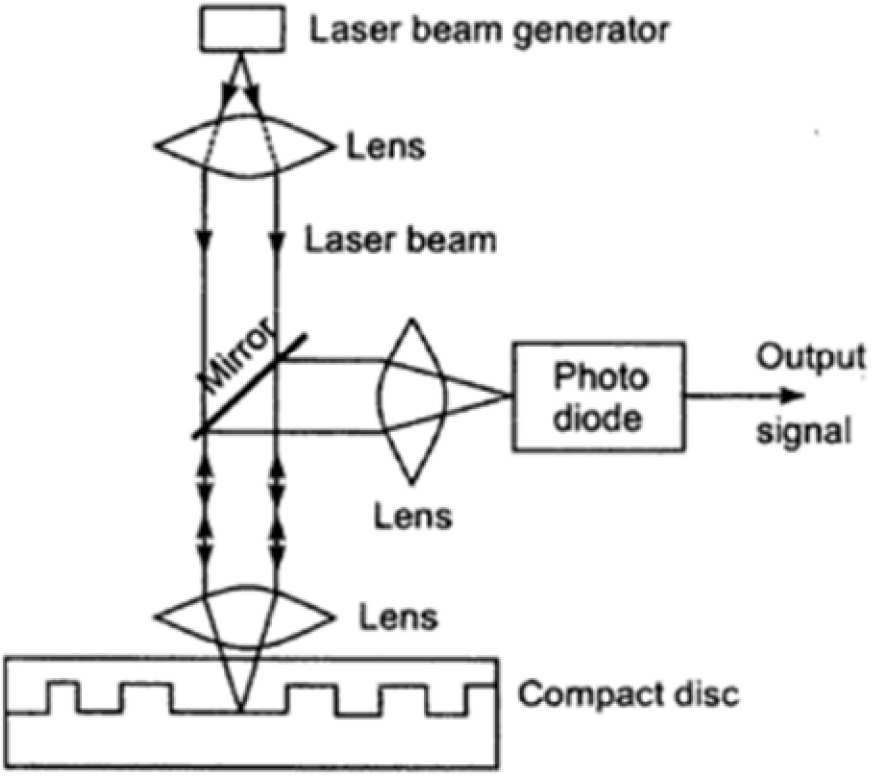


Figure 11.18: Optical Pickup System of a Compact Disk

The area in between the pits is called ***lands***. A land is a smooth surface that reflects back at a higher intensity. A photosensor detects the change between the change between the pits and the lands and converts them to a digital signal.

The information on the compact disks is not stored in tracks. Rather, the disk contains a single spiral track, beginning near the center and spiraling out to the outer edge of the disk. The sectors inside and the outside of the spirals are of the same length. So the information is evenly spread across the disk. The magnetic spots or pits are read by the laser at a *Constant Linear Velocity*. The disk rotates more slowly to access the data near the outer edge.

When data is read from a CD, a laser is directed to the surface of the disk. The laser follows the stream of pits and lands from the inside of the disk to the outer in a spiral direction. As the laser shines on the data track, it reflects one pattern of light off of a pit and a different pattern off of a land area. The resulting reflections equate to a series of ones and zeros by a photo sensor. It is shown in the figure below.

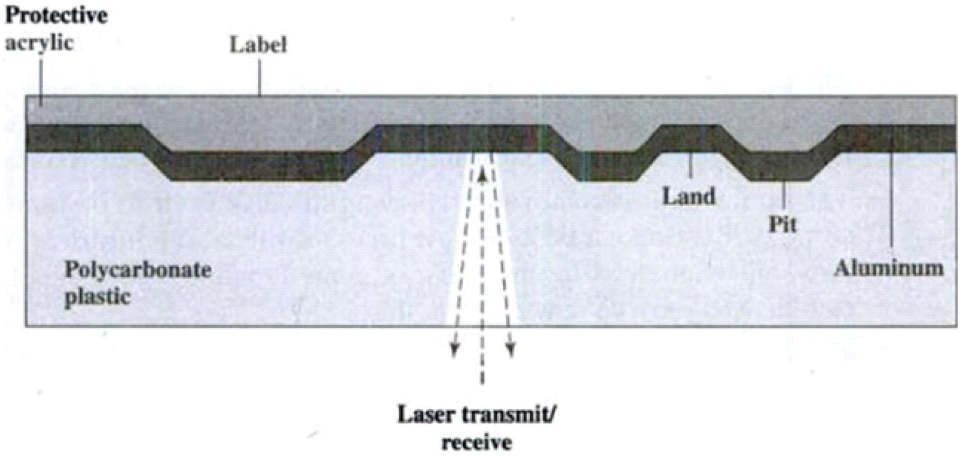


Figure 11.19: CD Operations

Data in CD ROM is stored as a sequence of blocks. It consists of the following fields.

Sync– it identifies the beginning of a block. It has a byte of all 0s, 10 bytes of all 1s and then a byte of all 0s.

Header– it contains the block address and mode byte.

Mode 0 = Blank data field

Mode 1 = 2048 data bytes + error correcting code

Mode 2 = 2336 data bytes

The block format is shown in the figure below.

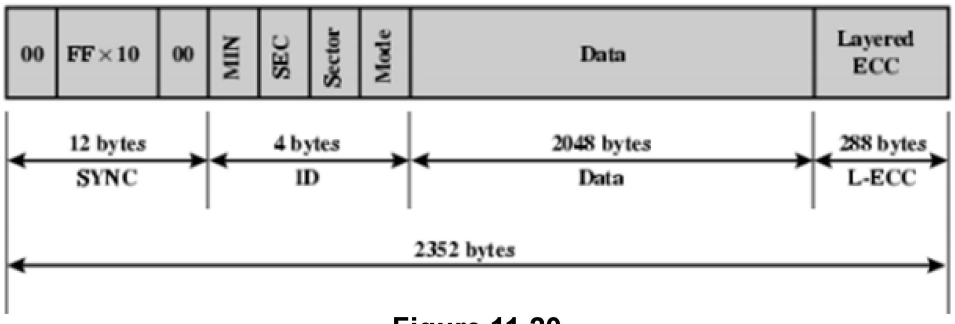


Figure 11.20

The advantages of a compact disk are mainly two. These are

1) The disk together with its information can be replicated inexpensively.
2) It is removable. Thus, the disk can be used for archival storage.

The disadvantages are:

1) The CDs are Read only and cannot be updated.
2) The access time is more than that of a magnetic disk.

DVD (Digital Versatile Disk): The DVDs took the video to the digital age. It is based on the CD technology with increased storage. It has much larger storage capacity. They can be single or double sided. A two layered DVD can hold up to 17 GB of data, video or audio or other information. The DVD has a multiple layers of plastic pieces with a total thickness of 1.2 millimeters. Polycarbonate plastic is basically used for making these layers.

The greater capacity of the DVDs than that of the CDs are due to three reasons as given in the figure below.

1) The bits in a DVD are more closely packed as that of a CD.
2) The DVD has a second layer of pits and lands on top of the first layer. A dual layer DVD has a semi-reflexive layer on top of the reflexive layer. Each layer of the DVD can be read by the lasers.
3) DVDs can be two-sided, whereas the data can be recorded on only one side of a CD. This increases the capacity of the DVD.

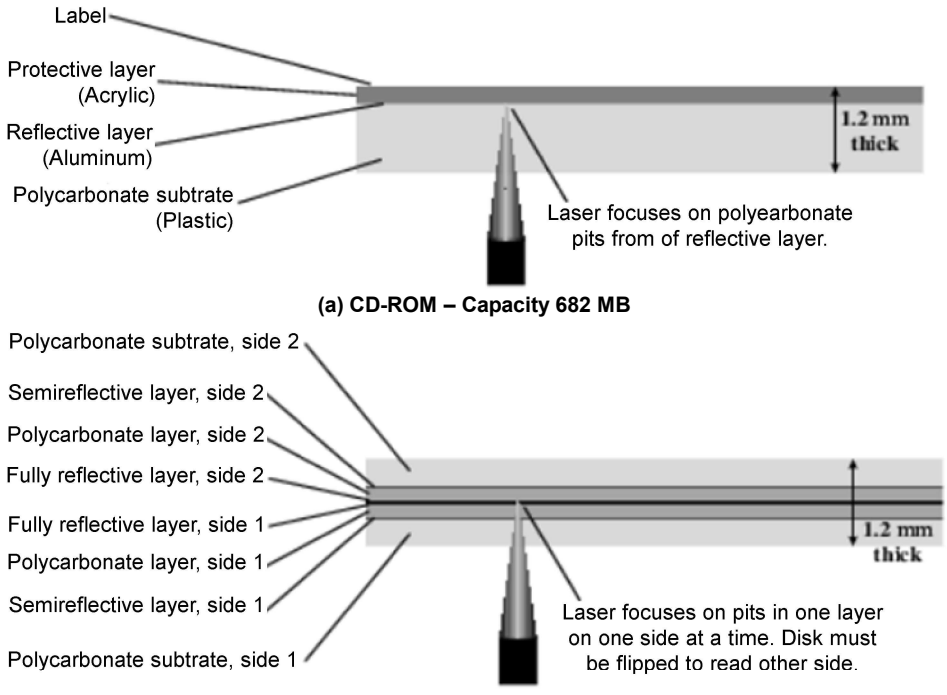


Figure 11.21

11.16 MAGNETIC TAP

Magnetic tape is one of the most popular storage medium for large data that are sequentially accessed and processed. It is a flexible polyester

tape coated with a magnetizable material. Data in the magnetic tape is stored in the form of magnetized and de-magnetized portion on the surface of the tape. The magnetized portion on the surface refers to '1' bit value and the demagnetized portion refers to '0' bit value. The amount of data that can be stored on a linear inch of tape is known as the tape's recording density.

The data in the magnetic tapes are stored in the tracks that run parallel lengthwise. The recording of the data in the parallel tracks is called parallel recording. If the data is laid out as a sequence of bits along each track, the recording is called serial recording. Data are written in blocks which are called the physical records on the tape. The blocks are separated by gaps called the inter-record gaps.

The magnetic tape drive is an example of a sequential storage device. The tape must be rewind or fast forwarded to get the correct block under the read/write head. The magnetic disks, on the other hand, are direct access devices.

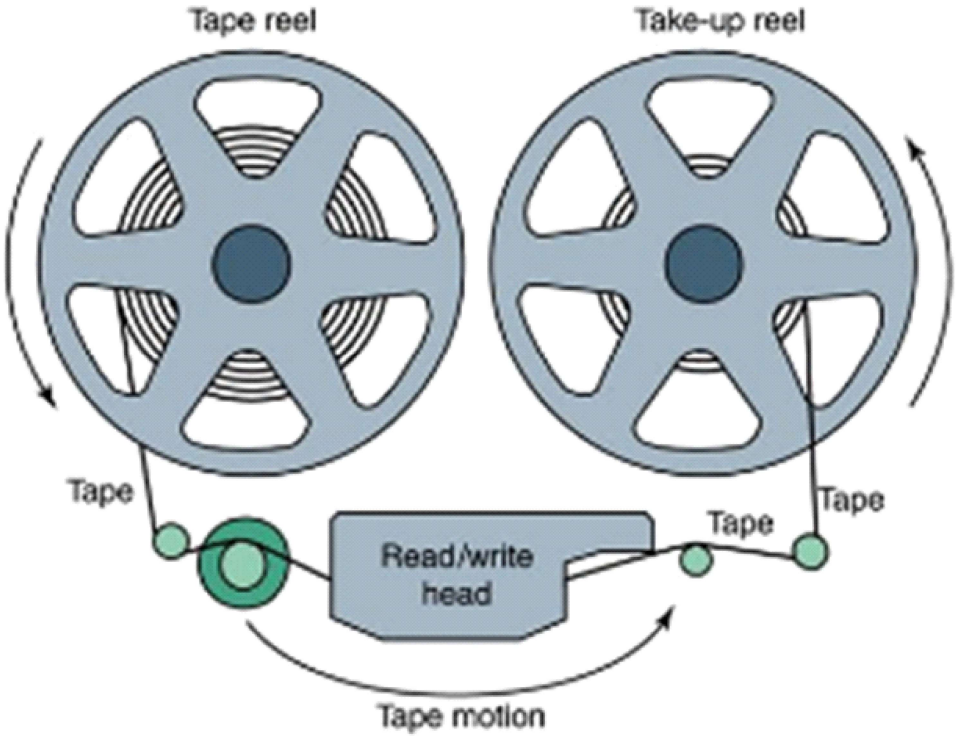


Figure 11.22: A Magnetic Tape Storage Medium



CHECK YOUR PROGRESS

Q.2: Fill In the blanks:

i) The time required to position the read/write head from the current track or cylinder to the desired track or cylinder is called .....

ii) ..... is the time elapsed between the initiation of a particular request for data and the receipt of the data.

iii) The transfer time to or from the disk depends on the ..... of the disk.

iv) The time which is taken for the beginning of the sector to reach the head is called .....

v) RAID is .....

vi) The access speed of the magnetic disks is considerably ..... as compared to the speed of the processor and the main memory.

vii) ..... is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both.

viii) Examples of optical memory are ..... and .....

ix) A ..... detects the change between the change between the pits and the lands and converts them to a digital signal.

x) The magnetic spots or pits are read by the laser at a .....



11.11 LET US SUM UP

- A magnetic disk is a thin, circular metal plate the surface is coated with a magnetic oxide layer usually on both the sides.

- Information is stored on the disks which are represented as magnetic spots. Presence of a magnetic spot represents a 1 bit and the absence of a magnetic spot represents a 0 bit.
- A disk pack may contain anywhere from five to one hundred disks.
- A read/write mechanism is used to achieve the access to read or write on the magnetic disk.
- The data on the magnetic disk is organized in a concentric set of circles called track
- Each of the tracks has the same width as the head and the adjacent tracks are separated by gaps.
- The tracks are numbered from the outermost to the innermost circle starting with zero.
- Each track is divided into sectors.
- The gap in between the tracks in a magnetic disk is called the Inter-track gap and the gap between the sectors in the disks is called inter-sector gap.
- A cylinder is made up of all the tracks of the same number from all the metal disks. Thus cylinder is an imaginary view of all the matching tracks on top of each other.
- There are two types of disks depending upon the two methods of accessing data. One is moving head and the other is fixed head.
- A fixed head disk has many read/write heads on each disk surface as the number of tracks which are mounted on the access arm
- The moving head disk systems have one read/write head for each disk surface which is mounted on the access arm.
- The time required to position the read/write head from the current track or cylinder to the desired track or cylinder is called seek time.
- Access time is the time taken to read a record from the magnetic disk.
- The transfer time is the time taken to transfer a block of bits.
- After the track is selected the disk controller waits until the appropriate sector rotates to line up with the head. This time which is taken for the beginning of the sector to reach the head is called rotational delay or rotational latency.

- RAID is a way of storing data in different places on multiple disks to protect the data in case of a drive failure
- In a RAID, a single large file is stored in several different disks by breaking up the file into a number of smaller pieces and storing the pieces on the separate disks. This is called ***data stripping.***
- CD (Compact Disk) and DVD (Digital Versatile Disk) are common examples of optical memory.



11.18 FURTHER READING

1) Chaudhuri, P. Pal. *Computer Organization and Design*. PHI.
2) Godse, A. P., Godse, D. A. *Computer Architecture*. Technical Publications Pune.
3) Stallings, William. *Computer Organization and Architecture: Designing for Performance*. PHI.



11.19 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) Magnetizing coil, magnetic yoke; ii) read/write; iii) electrical current; iv) magnetoresistive; v) tracks; vi) smaller; vii) sectors; viii) Inter-track gap, inter-sector gap

Ans. to Q. No. 2: i) seek time; ii) access time; iii) rotation speed; iv) rotational delay; v) Redundant Array of Independent Disks; vi) Slow; vii) RAID; viii) CD, DVD; ix) Photosensor; x) Constant Linear Velocity



11.1 MODEL QUESTIONS

Q.1: Explain the organization of the magnetic disk.

Q.2: Compare magnetic disk with that of magnetic tape

Q.3: What is RAID technology? Explain the different levels of RAID.

Q.4: What is optical memory? Explain with examples.

*** ***** ***

UNIT 12: PROCESSOR STRUCTURE AND FUNCTION

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Processor Organization

12.4 Registrar Organization

12.4.1 User Visible Registers

12.4.2 Control and Status Registrar

12.5 Steps for Executing a Program

12.6 Instruction Cycle

12.7 Instruction Pipelining

12.8 Let Us Sum Up

12.9 Further Reading

12.10 Answers to Check Your Progress

12.11 Model Question

12.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about processor organization and functioning
- learn about types of user visible registers and their roles
- describe role of control and status registers in processor operations
- describe various steps of instruction cycle
- learn about basics of instruction pipeline.

12.2 INTRODUCTION

This unit discusses internal structure of processor organization. Register which forms the internal memory of a processor are analyzed in this unit. Various functions of registers including user visible register and control and status register are discussed in detail. Instruction cycle and the phases of instruction cycle are also described. The concept of instruction

pipeline which increases the performance of a system are also discussed at the end of this unit.

12.3 PROCESSOR ORGANIZATION

A processor consists of three major components namely Arithmetic Logic Unit (ALU), Control Unit (CU) and Registers. The figure 12.1 gives a simplified view of a processor. All the components within the processor are interconnected through buses. There are two types of buses: internal bus and external bus. The internal bus is responsible for transferring information within the components in the processor. On the other hand the external bus is responsible for transferring data to and from the processor to the memory unit and the input-output devices. The processors internal data bus is connected to external data bus by logic called bi-directional bus or transreceiver.

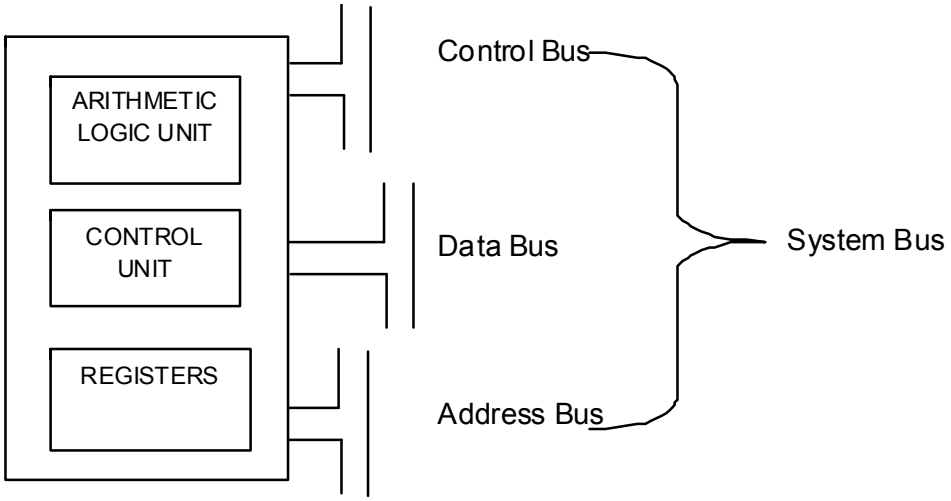


Figure 12.1: Simplified View of Processor Organization

The detailed diagram of a general processor organization is given in figure 12.2. The processor needs a small temporary memory to store data and instruction. It reads the instruction from memory, decodes it, fetches data, executes instruction and finally writes the result back to memory. For doing so it utilizes a set of registers. The ALU does the main processing of the data which are stored in the registers. The Control unit's responsibility is to control the operations of the ALU. The Control unit also checks the movement of data in and out of the ALU.

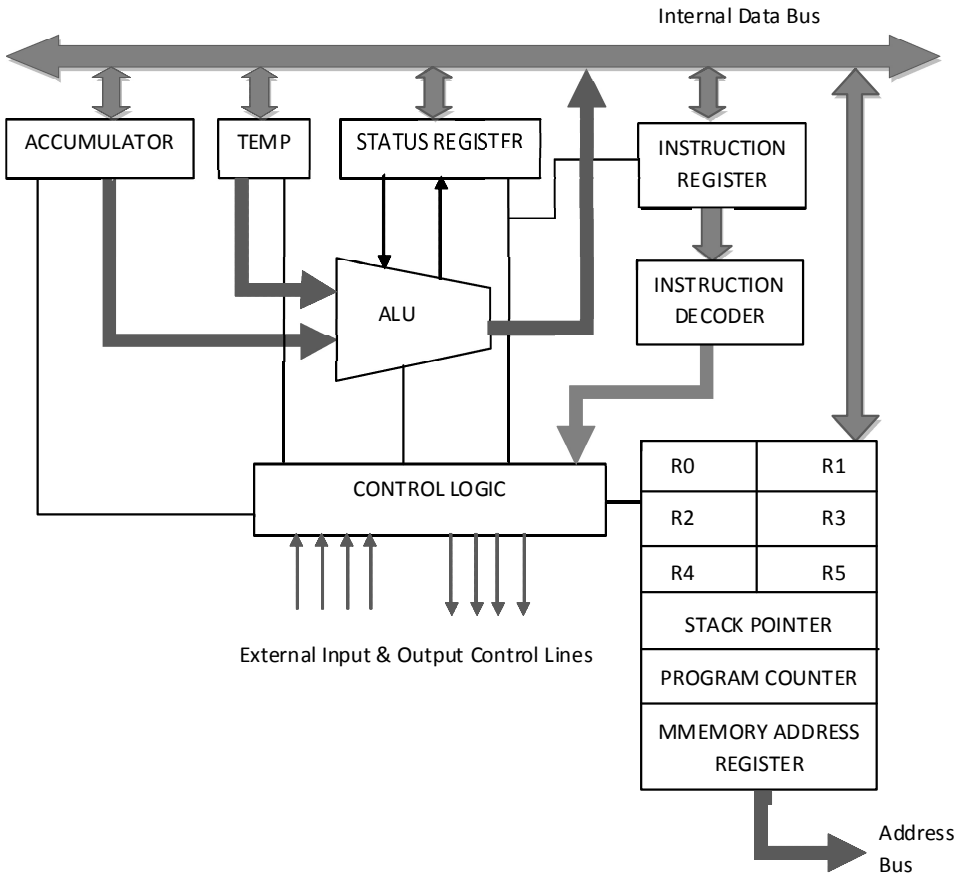


Figure 12.2: General Processor Organization

12.4 REGISTRAR ORGANIZATION

The processor consists of various registers and each register is used for different purposes. Following are the two categories of registers and their functionality.

12.4.1 User Visible Registers

These are the set of registers available to the assembly language programmers so that the main memory references can be minimized. Programmers can make the optimal use of the registers while writing a program. Following are the categories into which user visible register can be divided:

- General Purpose registers
- Accumulator
- Segment Register
- Index Register
- Stack pointer

General Purpose Registers: These are the set of registers ranging from R0 to R(n–1) that are available to the programmers. The programmers can directly make use of these registers in their programs to store the operands and the results. Use of the general purpose registers help minimize memory references therefore making the program execution faster.

Figure 12.3 show general register organization of seven processor registers. It shows how registers are selected by the bit combination of the three select A and select B lines and data is transferred from registers to ALU. Also how the registers are selected using the 3×8 decoders to store the final result. The process involves connecting output of each register to the multiplexer A and multiplexer B. The selection line A & B selects the register outputs and passes them to bus A & B. The two buses form the input to the ALU. The type of ALU operation to be performed on the input is selected by the select operation lines. Finally the result is stored in one of the register. The register selected for storing the result is done using 3×8 decoder.

For example the operation: R1 ← R2 + R3 are executed as

- SELECT A places the content of R2 into bus A
- SELECT B places the content of R3 into bus B
- SELECT OPERATION does the arithmetic addition of the contents of the bus A & B
- SELECT REGISTER is used to select register R1 and transfer the content of the output bus into R1.

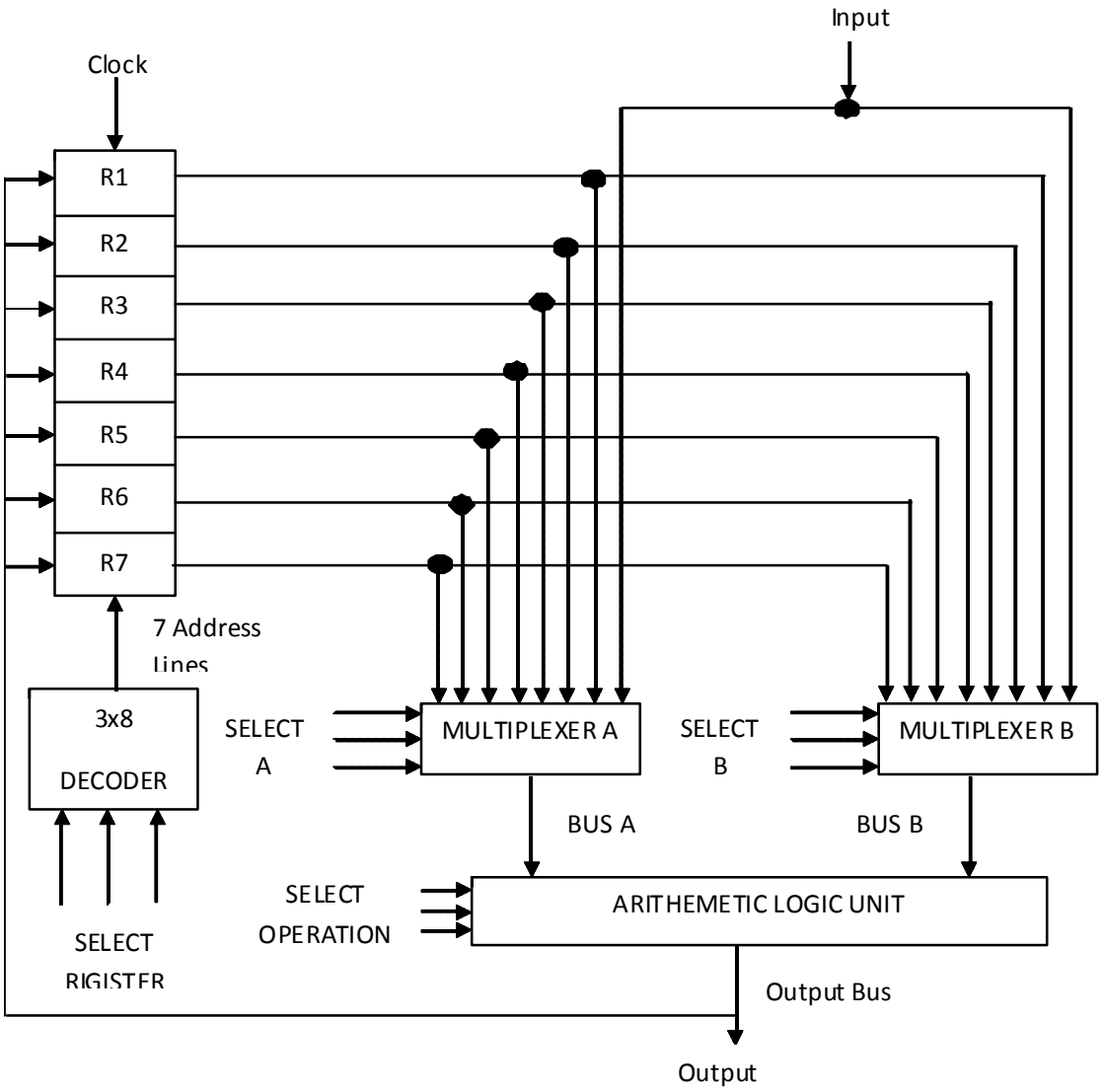


Figure 12.3: General Register Organization

The binary code for selecting register can be understood as:

Binary Code	SELECT A	SELECT B	SELECT REGISTER
000	Input	Input	–
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Accumulator: It is one of the major working register of a processor. For any operation involving two words in ALU, the accumulator holds one of the operand while the other operand is in another register or in memory. For certain operations like complement, the operand must be stored in the accumulator for complementing. Most of the time the result of operation is stored in accumulator which is later transferred to any other register or memory. Also the transfer of data between processor-memory and processor-input output device involves accumulator. It is user accessible and represent by 'A'.

Segment Register: A segment register is generally found in machine that supports segmented addressing. In such cases, the segment register is used to hold the starting address of the segment. There are three main segments namely code segment that contains all instructions of a program to be executed, data segment contains data and constants to be used by the program in the code segment and lastly the stack segment which stores the return address of procedures or functions. In addition to this there is one more segment called extra segment to store data.

Index Register: The machine that supports indexed addressing mode uses index register. For example SI is a source index register for string operations and DI is a destination index register for string operations.

Stack pointer: It is a user visible stack addressing register used to store the address of the top of the stack. It allows implicit stack addressing as the operations PUSH and POP does not need any explicit stack operand.

12.4.2 Control and Status Registers

These are the set of registers that are not visible to users but are required to control the operations in the processor. Following are the categories of these registers:

i) Program Counter

ii) Instruction Register

iii) Memory Address Register

iv) Memory Buffer Register also called Memory Data Register

v) Program Status Word or Status register or condition code register

i) Program Counter (PC): It is an important register used to inform the CPU the address of the next instruction to be executed. A program is a set of instructions which are executed by the CPU in a sequential order. Therefore the program counter helps the CPU by providing the address of the current instruction to be executed and also the address of the instruction to be executed next. The program counter is increment automatically to the next address once the CPU reads the content of program counter. The length of the program counter gives the maximum length of the program in bytes as for example a 16 bit program counter can address to the maximum 2^{16} bytes of memory.

ii) Instruction Register (IR): This register is used to store the instruction fetched from memory specified by the program counter, i.e., the instruction is available in the IR after it is fetched from memory.

iii) Memory Address Register (MAR): This register is used to store the address of the memory location. In a system the MAR connects directly to the address bus. Data are transferred with the help of MAR and MBR. The memory address from which or to which data are to be transferred is specified by the content of the MAR.

iv) Memory Buffer Register (MBR): Contains the data word to be written to memory or read from memory. It works along with MAR. It is also called memory data register. The data is available in the MBR or MDR after reading is completed.

v) Program Status Word (PSW): PSW register contains condition code or status information. When some operation is performed in the ALU, some of the status bits/flags in the PSW is either set/reset. Based on the flags the program can take some logical decisions, for example jump to a specified address when the zero flag is set. Following are the common PSW flags /bits:

Carry: Set to 1 if an operation results in a carry (addition) or borrow (subtraction) out of high order bit.

Zero: Set to 1 when result of an operation is zero.

Sign: Set to 1 if result of the last operation is negative.

Auxiliary Carry: Is set to 1 when addition of first four bits results in carry in the 5^{th} bit.

Equal: Set to 1 if logical comparison operation results in equality.

Overflow: Set to 1 if the result of addition of two signed numbers overflows into sign bit.

Parity: Set to 1 if the operation results in byte that contains even number of one's.

12.5 STEPS FOR EXECUTING PROGRAM

To execute a program/instruction stored in memory the processor goes through the following steps:

- **Fetch Instruction:** The task of fetching the instruction one at a time is achieved with the help of two registers namely Program counter and Instruction Register. The program counter keeps track of which instruction is being executed and what instruction will be executed next. The instruction register is used to store the instruction currently being executed. In addition to this two more registers MAR and MDR are used to transfer data between memory and processor.
- **Interpret Instruction:** After fetching the instruction, the next step is to decode or interpret the instruction so as to know what action needs to be taken. This step also identifies how many bytes the instruction is? This helps to identify how many numbers of bytes of data to read from memory.
- **Fetch Data:** At this step the bytes of data is read from memory or I/O device depending upon the type of instruction; one byte, two byte or three byte. If the instruction is of one byte then data is assumed to be in special register called accumulator. If instruction is of two bytes, first byte represent instruction and second byte represent data. If instruction is of three bytes, first byte represents instruction and second and third

byte represents data. So data of one or two bytes is fetched from memory and stored in processor registers at this step.

- **Process Data:** Once the data and the action to be taken on the data are available, the processor executes the instruction requiring some arithmetic or logical operations.
- **Write Data:** The result obtained from previous step will require the data to be written or stored in memory or I/O device.

12.6 INSTRUCTION CYCLE

It is a process by which a computer goes through a cycle of steps to read instruction and data from memory, execute the instruction and store the result. The figure 12.4 shows an instruction cycle. The instruction cycle is consists of the following four cycles:

- Fetch Cycle
- Indirect Cycle
- Execute Cycle
- Interrupt Cycle

The process starts with fetch cycle. The Interrupt Cycle is always followed by the Fetch Cycle Then Indirect Cycle is followed by the Execute Cycle. In case of fetch and execute cycles, the next cycle is dependent on the current state of the system.

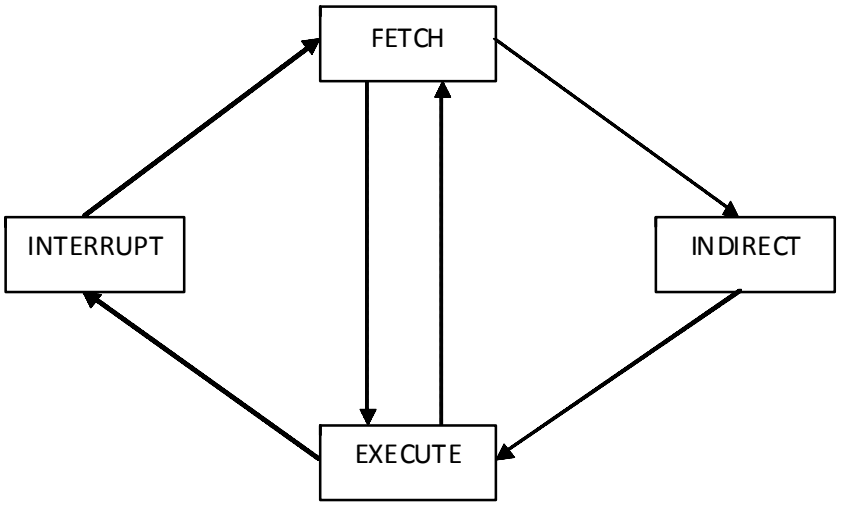


Figure 12.4: Instruction Cycle

The instruction cycle involves the following registers in the process of instruction execution.

- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Program Counter (PC)
- Instruction Register (IR)

Fetch cycle: In the beginning of any cycle, the instruction to be executed next is available in the Program Counter (PC). And following steps are taken:

- The content (address) of program counter is placed in the memory address register (MAR). The MAR is connected to the address bus of the system bus.
- The control unit then issues a READ command and the data from the memory specified by MAR is copied to data bus.
- Next the data bus content is copied to memory buffer register (MBR) and program counter (PC) is incremented by one, so that address of the next instruction is ready.
- Finally the MBR content is moved to instruction register (IR).

Indirect cycle: After fetching the instruction and storing it in instruction register, the next step is to fetch the operands by indirect addressing. If the operands are in register then no need to fetch the operands. Otherwise following steps are taken:

- Rightmost N bits of the Memory buffer register (MBR) containing the reference operand address is transferred to the MAR.
- Control unit issues a memory read instruction, and the desired operand address is moved to the MBR.

Execute Cycle: This cycle depends on the type of instruction in the IR. The instruction may be a data transfer instruction, a branch instruction, a memory read or write instruction, and an I/O read or write instruction or some arithmetic or logical operation in the ALU.

Interrupt Cycle: On completion of the execute cycle the processor checks if the any interrupt has occurred or not. If the interrupt is enabled and any interrupt occurs then the interrupt cycle takes place.

- Contents of program counter are transferred to memory buffer register and the contents are saved so that on completion of the interrupt cycle it can return to the point where it transferred the control to interrupt.
- Contents of the PC are saved to the address specified by MAR.
- Next PC is loaded with the starting address of the interrupt service routine.

On completion of the interrupt cycle, previously saved return address is used to fetch the next instruction.

12.7 INSTRUCTION PIPELINING

A pipeline is similar to an assembly line in a production factory. A product has to go through multiple stages in the assembly line before the final product is manufactured. At a time all the stages work simultaneously but on different stages of the product. This process is referred to as pipelining. In a computer system the technique of executing multiple instructions in an overlapped fashion is known as ***pipelining***. A pipeline consists of many stages and these stages are connected to one another in a pipe like structure. An instruction enters one end of the pipeline, goes through several stages before exiting from another end. Pipelining improves the overall throughput of the system.

In a pipeline system, each stage uses register to hold the output of that stage. Output of one stage is applied as input to the next stage.



Figure 12.5: Five Stage Instruction Pipeline

Figure 12.5 shows an example of five stage instruction pipeline consisting of fetch, decode, operand fetch, execute and write stages. Here streams of instructions are executed in overlapped fashion thereby increasing the throughput of the computer system.

Figure 12.6 shows the timing diagram of an instruction pipeline. While the instruction pipeline reads one instruction from the memory,

previous instructions is executed in other stage of the pipeline. Thus multiple instructions are executed simultaneously. From the figure 12.6 it can be observed that while the first instruction started at time period one, the second instruction started at time period two and so on. Up to time period four not all stages were working simultaneously, but from time period five onwards all the five stages are working simultaneously. Therefore from instruction number five onwards each stage is working on a different instruction as:

Instruction 1: Write

Instruction 2: Execute

Instruction 3: Operand Fetch

Instruction 4: Decode

Instruction 5: Fetch

Time →

	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	F	D	OF	E	W						
Instruction 2		F	D	OF	E	W					
Instruction 3			F	D	OF	E	W				
Instruction 4				F	D	OF	E	W			
Instruction 5					F	D	OF	E	W		
Instruction 6						F	D	OF	E	W	
Instruction 7							F	D	OF	E	W

Figure 12.6: Timing Diagram for Instruction Pipeline Operation

If there are ***k*** number of stages and ***n*** number of instructions, then total time ***T*** taken to execute n instructions can be given as:

$$T = k + (n–1)$$

The pipeline performs to its maximum if all the stages take same amount of time to process and there is no branch instruction in the program. In case of a branch instruction, figure 12.7 shows the timing diagram of the branch instruction pipeline assuming instruction four is a branch instruction. Here branch is not taken until the end of time period 7. At this point the pipeline clears all those instruction which are not useful i.e. instruction 5, 6 and 7. Also no instruction complete in time period 9, 10 and 11. This is known

as branch penalty as the processor could not anticipate the branch. So the instruction 17 assumed to be the instruction to be executed on the branch starts at time period eight.

Time → |←Branch Penalty→|

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	F	D	OF	E	W									
Instruction 2		F	D	OF	E	W								
Instruction 3			F	D	OF	E	W							
Instruction 4				F	D	OF	E	W						
Instruction 5					F	D	OF							
Instruction 6						F	D							
Instruction 7							F							
Instruction 17								F	D	OF	E	W		
Instruction 18									F	D	OF	E	W	

Figure 12.7: Timing Diagram for Instruction Pipeline Operation



CHECK YOUR PROGRESS

Q.1: Choose the correct option:

i) A processor consists of–

a) Registers b) ALU, Control Unit

c) ALU, Control Unit, Registers

d) ALU, Control Unit, Registers, Memory

ii) Which of the following is not a user visible register–

a) Accumulator b) Segment Register

c) Index Register d) Program counter

iii) A program consists of the following segment registers–

a) Code, Data b) Code, Data, Stack

c) Code. Data, Stack, Extra

d) Code. Data, Stack, Segment

iv) The register that points to the top of the stack is called–

a) Stack register b) Stack pointer

c) Push Pop pointer d) All of the above

v) The register that is used to keep track of next instruction to be executed is–

a) Program Counter b) Instruction Register

c) Program Status Word d) Stack Pointer

vi) Consider subtraction of two floating point numbers 10101010 and 10101011. Which bit will be set?

a) Sign b) Carry

c) Overflow d) Both (a) and (b)

vii) What will be the auxiliary carry and parity bit if the number 10101011 is complemented

a) 1, 0 b) 0, 1 c) 1, 1 d) 0, 0

viii) Which of the following is not a valid flag?

a) Borrow b) Direction

c) Index d) Overflow

ix) The order in which phases of instruction cycle are executed is

a) Fetch cycle, Indirect cycle, Execute cycle, Interrupt cycle

b) Fetch cycle, Execute cycle, Indirect cycle, Interrupt cycle

c) Fetch cycle, Interrupt cycle, Execute cycle, Indirect cycle

d) Interrupt cycle, Fetch cycle, Execute cycle, Indirect cycle

x) Time taken to execute 10 instruction in a five stage pipeline is–

a) 12 b) 13 c) 14 d) 15



12.8 LET US SUM UP

- A processor consists of three major components namely Arithmetic Logic Unit (ALU), Control Unit (CU) and Registers.

- User Visible Registers are the set of registers available to the assembly language programmers so that the main memory references can be minimized.
- It is one of the major working register in the processor. For any operation in ALU involving two words, the accumulator holds one of the operand while the other operand is in another register or in memory.
- A segment register is generally found in machine that supports segmented addressing. In such cases, the segment register is used to hold the starting address of the segment.
- It is a user visible stack addressing register used to store the address of the top of the stack.
- It is an important register used to inform the CPU the address of the next instruction to be executed.
- Instruction Register is used to store the instruction fetched from memory specified by the program counter.
- Memory Address Register is used to store the address of the memory location.
- Memory Buffer Register Contains the data word to be written to memory or read from memory. It works along with MAR.
- Program Status Word register contains condition code or status information. When some operation is performed in the ALU, some of the status bits/flags in the PSW is either set/reset.
- To execute a program/instruction stored in memory the processor goes through following: fetch instruction, interpret instruction, fetch data, process data, and write data.
- It is a process by which a computer goes through a cycle of steps to read instruction and data from memory, execute the instruction and store the result.
- In a computer system the technique of executing multiple instructions in an overlapped fashion is known pipelining.



12.9 FURTHER READING

1) Hamacher, Carl. (5th Edition). *Computer Organization*. McGraw Hill.

2) Mano, M. Morris; Mall, Rajib. (3rd Edition). *Computer System Architecture*. Pearson.

3) Stallings, William. (10th Edition). *Computer Organization and Architecture: Designing for Performance*. Pearson.



12.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (c), ii) (d), iii) (c), iv) (b), v) (a), vi) (d), vii) (d), viii) (c), ix) (a), x) (c)



12.11 MODEL QUESTION

Q.1: Explain the roles of user visible processor registers?

Q.2: Discuss segment registers role in a user program.

Q.3: What is a program status word? How different flag bits contribute in an ALU operation?

Q.4: What are the various phases of instruction cycle?

Q.5: Explain instruction pipeline using a suitable example.

Q.6: Briefly explain how instruction pipeline can contribute to conditional branch instruction.

Q.7: Explain how memory address register and memory buffer register help in data transfer operation.

Q.8: Using an example show the difference between Carry flag and Overflow flag.

Q.9: Consider addition of two 8-bit numbers 00110111 and 11100010. What will be the value of following flags: (a) Carry, (b) Auxiliary Carry, (c) Sign, (d) Zero, (e) Parity

*** ***** ***

UNIT 13: INTRODUCTION TO PARALLEL PROCESSING

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Pipelining

13.3.1 Hazards of Pipelining

13.3.2 Structural Hazard

13.3.3 Data Hazard

13.3.4 Control Hazard

13.4 Parallel Processing

13.5 Let Us Sum Up

13.6 Answers to Check Your Progress

13.7 Further Reading

13.8 Model Questions

13.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the basic concepts of pipelining and the hazards associated with it
- describe the concepts of superscalar and parallel processing.

13.2 INTRODUCTION

We have already come to know the basic organization of a computer and how a processor executes a program. To execute a sequence of instructions, the processor uses a cycle of fetch-decode-execute operations: an instruction is fetched from memory, it is then decoded and executed; the next instruction in the sequence is fetched, decoded and executed and the cycle is repeated until the end of the program (or sequence of instruction) is reached.

In this unit, we shall consider only the three stages viz. Fetch, Decode

and Execute and discuss how the simple fetch- decode- execute cycle can be optimized so that performance of the system is improved. Pipelined execution of instructions is based on the fact that the different phases of the fetch-decode-execute cycle require completely different hardware units. Pipelining involves reordering instructions so that all hardware units are used optimally to achieve better performance. Parallel processing further involves executing several independent instructions in the program at the same time, to achieve better performance.

13.3 PIPELINING

The basic idea of pipelining is similar to the assembly line operation in manufacturing plants. For example, in a car manufacturing plant, there is a sequence of stations each of which perform different stages of the assembling the car. The first station prepares the chassis, the next station adds the body to the chassis, the next station installs the engine and so on. While one station is preparing the chassis of one car, the second station is adding the body to the chassis of another car, the third station is installing the engine in yet another car and so on. This way, all stations are fully utilized at all times so that an optimal number of cars can be manufactured. Similarly, in pipelining, the execution of a single instruction has at least three different stages (fetch, decode and execute) that are performed by different hardware units. While the system bus is used to fetch an instruction from the memory, the decoder unit of CPU can be used to decode a different instruction and the ALU can be used to execute yet another instruction. Compared to sequential execution of instruction, pipelined execution leads to considerably better performance.

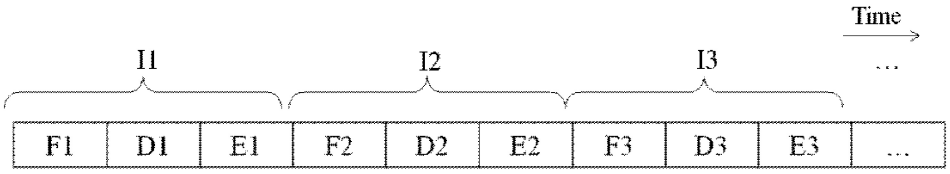


Figure 13.1: (a) Sequential Execution of Instructions I1, I2, I3, where F1, D1, E1 means fetch, decode and execution phase of instruction 1.

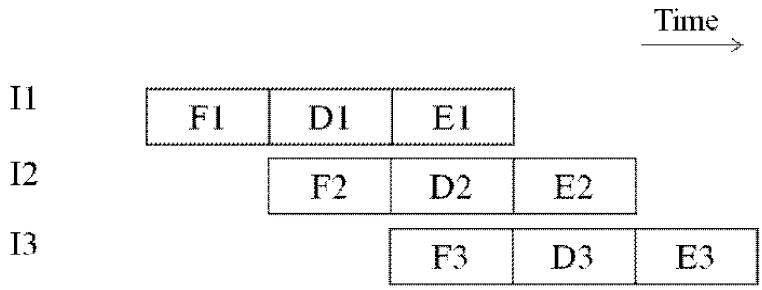


Figure 13.1: (b) Pipelined Execution of Instructions I1, I2, I3

Pipelined execution of sequence of instructions is shown in figure 13.1(b). The time taken to complete any of the fetch, decode or execute phases of an instruction is one clock cycle. While sequential execution (Figure 13.1(a)) completes execution of instructions I1, I2 and I3 in nine clock cycles, pipelined execution completes the same in just five clock cycles. An even greater difference in the number of clock cycles is seen if the total number of instructions to be executed is more. The number of pipeline stages may vary depending on the number of separate hardware units available in the system. The increase in performance, as a result of pipelining, is proportional to the total number of pipeline stages.

13.3.1 Hazards of Pipelining

The increase in performance, due to pipelining, is proportional to the number of pipelining stages only if the pipelined executions (as shown in Figure 13.1 (b)) can be continued throughout the entire program. However, during pipelined execution of a program, there are many situations when one or more stages of executions may be halted or ***stalled*** temporarily. As a result, it is often not possible to achieve an uninterrupted pipelined execution of instructions.

The pipelined execution of instructions may be interrupted due to various reasons– for example, the execution phase of some instruction requires more than one clock cycle. Figure 13.2 shows a situation where the execution phase of I1 (E1) requires two clock cycles instead of one. The execution phase of I2 (E2) has to be ***stalled*** for one clock cycle– instead of being completed in clock

cycle 4, it is completed in clock cycle 5. This is because the execution unit had not completed E1 and can handle execution of only instruction at a time. Any condition that causes the pipeline to stall is called a ***hazard***.

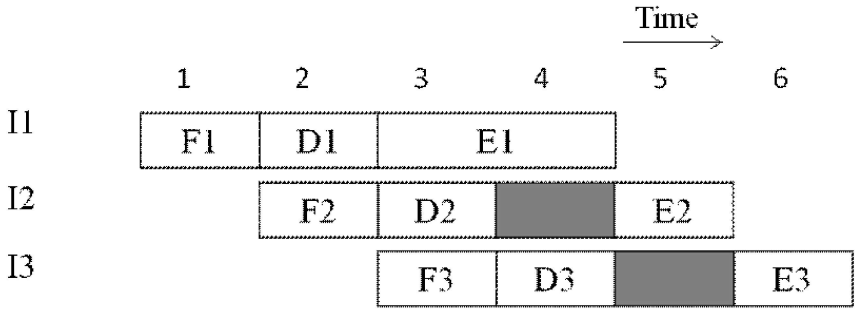


Figure 13.2: Pipelined execution of instructions when interrupted or *stalled*

Based on the type of condition that has caused the pipeline to stall, pipelining hazards can be classified as:

- Structural hazards
- Data hazards
- Control hazards

13.3.2 Structural Hazard

Structural hazard arises when two instructions in the pipeline require the use of the same hardware resource at the same time. One of the most common hardware resources that can lead to structural hazard is memory. Say, there is a sequence of instructions as follows:

I1: Add A, 5

I2: Add B, 5

I3: Load R1, C

Here, A, B, C are references to memory locations and R1 is a register. Each of the arithmetic instructions (I1 and I2) can be split into stages to: fetch instruction (F), decode instruction (D), fetch operand (M), execute arithmetic operation (E) and write result (W). Out of these, F, M and W require memory access. Similarly,

instruction I3 can be split into stages to: fetch instruction (F), decode instruction (D), and to execute the load instruction a memory access operation is require (M). The pipelined execution of these three instructions is shown in Figure 13.3 during clock cycle 5, both instruction I1 and I3 require memory access. As a result, I3 has to be stalled for clock cycle to allow I1 to complete. During clock cycle 6, again I2 and I3 both require memory access. Once again I3 is stalled to allow I2 to complete first.

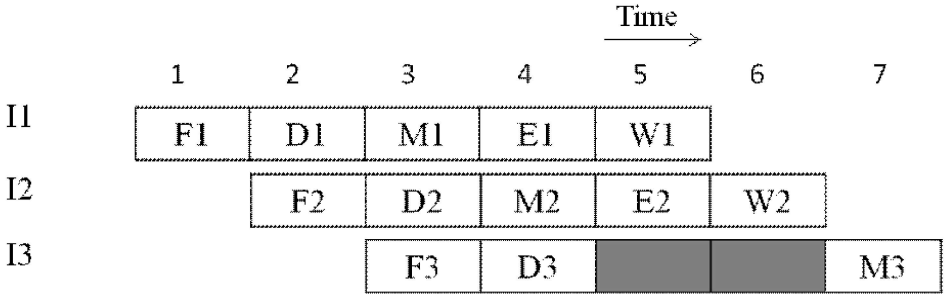


Figure 13.3: Structural Hazard caused by I1 and I3 Requiring Memory Access at the same time

To handle structural hazards, in general the number of hardware resources may be increased.

13.3.3 Data Hazard

Data hazards occur when an instruction operates on a data that is to be produced by some other instruction in the pipeline. Say there is sequence of instructions as follows:

I1: Add A, 5

I2: Load R1, A

I3: Add R2, R1

The pipelined execution of these instructions is shown in figure 13.4. In this sequence, immediately after adding 5 to the memory location A, the value of A has to be loaded to register R1. However, in the pipelined execution, even before the result of the addition operation is written into memory location A, instruction I2 attempts to access the value in A. If the initial value of A is 4, then after execution of I1 the value changes to 9 and the value loaded into

register R1 should be 9. However, due to pipelining, even before the value 9 is stored in memory location A, I2 is executed and the value loaded into register R1 is 4. Such an incorrect execution will affect the execution of instruction I3 and the end result.

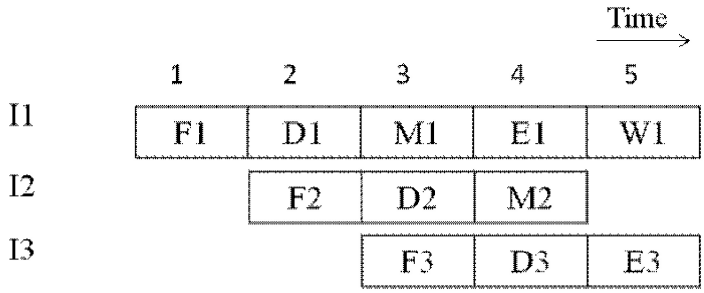


Figure 13.4: Effect of Data Hazard if not handled properly

It is difficult to deal with data hazards because such hazards are not detected easily. In order for the above example to be correctly executed, the system will have to detect that both I1 and I2 are working on the same data. This can be done only at the software level during compilation. Once detected, the execution of the second instruction can be stalled until the previous instruction has completed execution. Another way to handle data hazard would be to reorder the complete sequence of instructions such that by the time I2 is executed, the write results phase of I1 will be completed.

13.3.5 Control Hazards

In pipelining, the instruction fetch unit fetches subsequent instructions in advance so that the decode/execute unit does not have to wait. As a result, the execute unit is used optimally because it always has a steady stream of instructions that are ready for execution. However, this is only true if the sequence of instructions that is to be executed next is well-defined. In case of branching instructions (conditional and unconditional JUMP statements) the sequence of instructions to be executed next depends on the result of the current instruction in the pipeline. Such hazards caused by transfer of control instructions are called control hazards or instruction hazards. Say there is a sequence of instructions as follows:

I1: CMP A, 10

I2: JG label

I3: ADD R1, R2

I4: SUB B, 1

I5: label: SUB A, B

I6: ADD B, 1

The pipelined execution of this sequence of instructions is shown in figure 13.5. Here, whether instruction I3 is executed or I5 is executed depends on the result of the instruction I1 – if A is greater than 10 then I5 is executed, otherwise I3 is executed. However, as shown in the figure, the result of I1 is not obtained until clock cycle 4. But pipelining requires instructions I3 and I4 to be fetched even before that. Therefore in cases where A is greater than 10, redundant work (fetching of I3 and I4) is done by pipelining. This redundant work has to be discarded before the proper execution of the program can be continued.

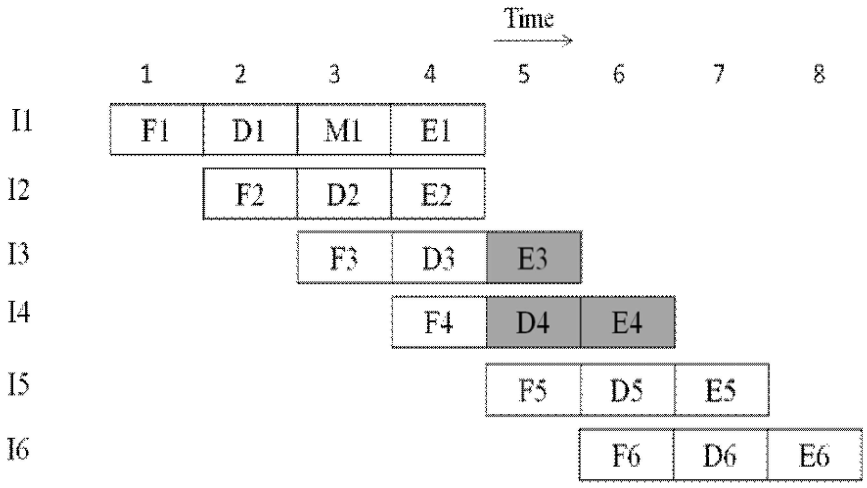


Figure 13.5: Effect of Control Hazard if not handled properly

Similar to data hazards, control hazards can be detected at the software level during compilation. To reduce the amount of redundant work done due to control hazards, the sequence of instructions can be stalled until execution of I1 is complete. Further, to efficiently utilize all clock cycles the sequence of instructions can be reordered. The reordering is done such that the execution of I1 is completed before fetching I3 or I4. It has to be ensured that the

instructions that are fetched immediately after I1 are instructions that have to be executed regardless of the outcome of the comparison operation. If such a reordering of execution is not possible then at the compilation stage itself a prediction is made as to which branch will be taken and the instructions are fetched accordingly. However, there is still a possibility that the branch is incorrectly predicted and any changes made due to the pipelined execution of the incorrect branch may have to be discarded later.



CHECK YOUR PROGRESS

Q.1: What are the different categories of pipelining hazards?

.....

.....

Q.2: When do data hazards occur?

.....

.....

Q.3: Which pipelining hazards may occur in the following sequence of instructions?

```
ADD     A, AH
SUB     AH, 30h
ADD     CH, B
CMP     CH, 10
JG      label1
ADD     AH, B
JMP     label2
label1: SUB     AH, B
label2: MOV     BH, B
```

13.4 PARALLEL PROCESSING

Traditionally, the execution of instructions in a computer is assumed to be strictly sequential in nature. However, with pipelined execution of

instructions, the sequential view of program execution is somewhat inaccurate. During pipelined execution, at any given clock cycle, there are several instructions in the pipeline that are in different stages of their execution. However, only a single instruction is in its execution stage at any given clock cycle. This is taken a step further with ***superscalar processors*** that allow multiple instructions to be executed at the same time.

Superscalar Processors: Superscalar organizations have several execution units within a single processor. This allows multiple instructions from the same program to be executed in parallel. As a result the number instructions executed per cycle is more than one. This is also known as ***instruction-level parallelism***. Figure 9.6 shows an example of instruction level parallelism in a superscalar processor with two execution units.

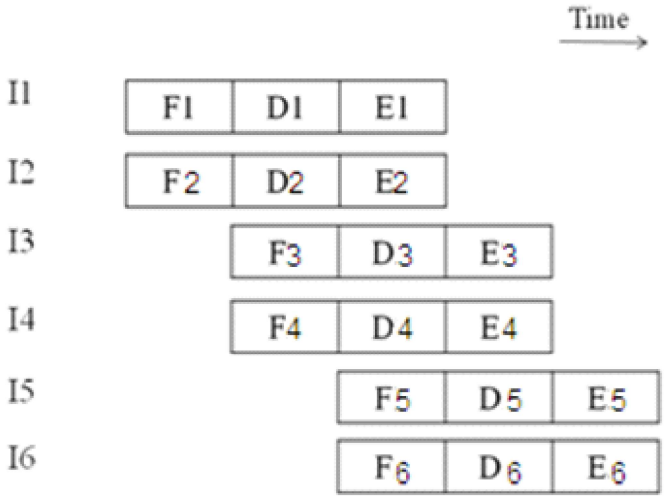


Figure 13.6: Instruction Level Parallelism in a Superscalar Processor with Two Execution Units

However, such an aggressive approach to increase performance of the system also has several issues. The various hazards encountered in pipelining are encountered in superscalar processors as well and are seen to be even more prominent. Further, most programs are written assuming a sequential execution of the instructions, with several dependencies amongst the instructions. In superscalar processors, multiple instructions are executed at the same time. If the dependencies amongst the instructions being executed parallel are not handled, then it may result in incorrect executions. Thus in such systems, in order to achieve high performance

and correct executions, it is important for the compiler to be able to deal with the hazards and the dependencies effectively by proper selection and reordering of the instructions.



13.5 LET US SUM UP

- Pipelining is based on dividing the work done in the instruction execution cycle into stages such that each stage can be executed by completely different hardware units.
- Pipelining involves concurrent execution of multiple instructions even though the instructions in the pipeline are in different stages of their execution. Therefore, pipelining does not follow a strictly sequential execution of instruction.
- Pipelining hazards occur because most programs are written for a sequential execution. Instructions in a program have several dependencies amongst themselves which are properly executed only if executed sequentially.
- There are three most common types of pipelining hazards: data hazards, control hazards, structural hazards.
- Pipelining aims to achieve a performance of one instruction per cycle.
- Superscalar processors have more than one execution units within the same processor. When combined with pipelining the performance of superscalar processors is expected to be more than one instruction per cycle.
- Superscalar processors encounter more aggressive forms of pipelining hazards.



13.6 FURTHER READING

1) Hamacher, V. C., Vranesic, Z. G., Zaky, S. G., Vransic, Z. & Zakay, S. (1984). *Computer Organization* (Vol. 3). New York et al.: McGraw-Hill.

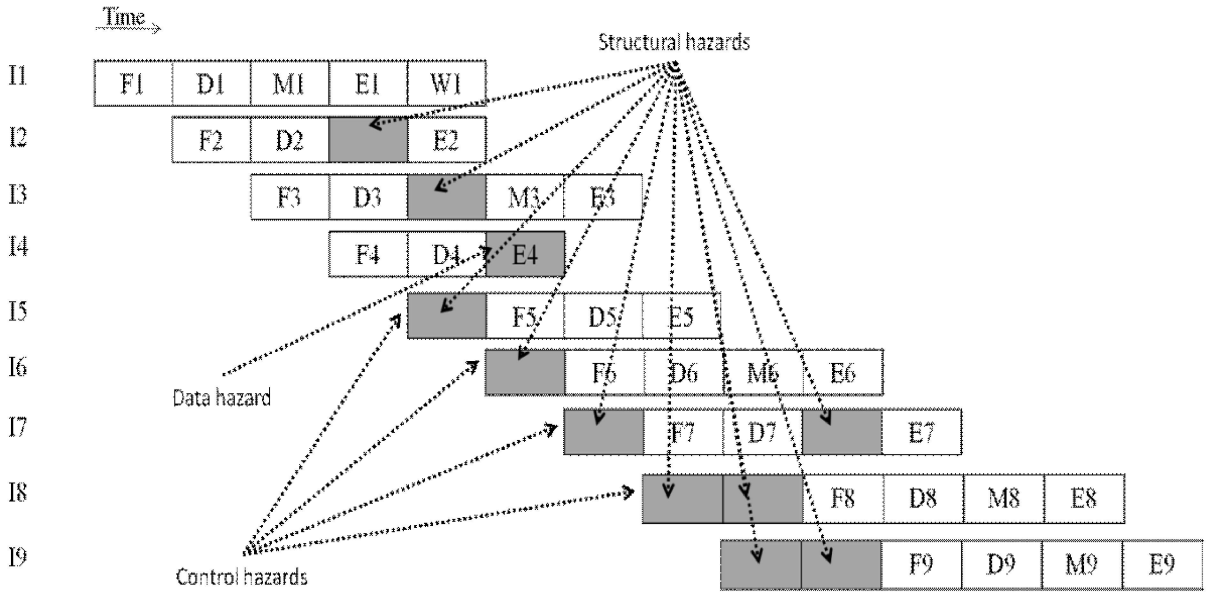
2) Stallings, William. (2006). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.

13.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: The different categories of pipelining hazards are: Structural hazards, Data Hazards and Control hazards.

Ans. to Q. No. 2: Data hazards occur when an instruction operates on a data that is also being operated on some by some other instruction in the pipeline.

Ans. to Q. No. 3:



13.8 MODEL QUESTIONS

Q.1: What is pipelining? Explain the pipelining hazards with example.

Q.2: What are superscalar processors?

Q.3: Differentiate between pipelining and superscalar processors.

Q.4: What is instruction level parallelism?

*** ***** ***

UNIT 14: CISC AND RISC ARCHITECTURE

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 Complex Instruction Set Computers (CISC)

14.3.1 Performance Analysis of CISC

14.4 Reduced Instruction Set Computers (RISC)

14.4.1 Performance Analysis of RISC

14.5 Let Us Sum Up

14.6 Further Reading

14.7 Answers to Check Your Progress

14.8 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- learn the basic concepts CISC architecture and its performance issues
- describe the basic concepts RISC architecture and its performance issues
- differentiate between CISC and RISC architecture.

14.2 INTRODUCTION

With the development and evolution of high level languages has led to a semantic gap, between the kind of operations allowed by these high level languages and the kind of operations allowed by the computer architecture. To meet the requirements of complex high level languages, complex architectures were designed that offered larger number of basic operations. On the other hand a reduced instruction set computer focuses on a simpler set of instructions and fewer basic operations that can be executed in one clock cycle.

In this unit, we shall discuss the problems associated with complex

architectures and how reducing the complexity of the architectures may lead to improved performances.

14.3 COMPLEX INSTRUCTION SET COMPUTERS (CISC)

High level languages (HLL) often include complex operations to make it easier for the programmer to express the algorithms. There is a semantic gap between the complex operations provided by the HLLs and the operations provided by the architecture. The translation of these complex operations into the basic operations that the architecture provides is left to the compiler. As a result the programs generated by the compilers are large and not efficient enough. To close this semantic gap, complex architectures with large instructions sets, several addressing modes, and more number of basic operations were designed to meet the requirements of the high level languages. Computers with such architectures are called ***Complex Instruction Set Computers*** (CISC).

Such complex instruction sets are expected to reduce the complexity of compilers and improve the efficiency of the code generated by the compilers. CISC architectures aim to complete any task in as few number of machine instructions as possible. To accomplish that, a large set of complex instructions are directly incorporated into hardware. Such a complex architecture also provides support for even more sophisticated HLLs.

The compilers in CISC architectures are much simpler and the machine code generated is also much smaller in size. As a result, another advantage to CISC architecture is that programs require lesser memory. Further, smaller programs have the advantage that the number of *instruction fetch* operations during the execution of the program will be much smaller. This is expected to reduce execution time of the program.

14.3.1 Performance Analysis of CISC

CISC architectures were originally designed to simplify compilers for high level language and enable compilers to generate more concise machine code. In an attempt to do so, complex

operations are incorporated into the hardware. However, it has been noted that it is often difficult to exploit such a complex set of machine instructions. In order to use the complex instruction set properly the compiler has to look for the machine language translation that exactly fits a HLL instruction. This becomes difficult and time consuming when the number of choices that the compiler has for translating some HLL instruction is very large.

Further, CISC architecture compilers were expected to reduce the size of machine code generated. However, with the advances in technology, memory has become much cheaper and therefore is no longer a constraint. Additionally, it is not always true that CISC architectures generate smaller size machine code because the length of a single machine instruction is much larger. As the number of opcodes increase with the number basic machine operations available, the number of bits required to specify the opcodes and the various addressing modes also increase.

In CISC architectures instruction execution was expected to be faster. This is because in order to execute a single HLL instruction much fewer number of machine instructions need to be fetched and executed. However, in practice this is not always so. To allow for such a sophisticated set of instructions, either the control unit needs to be much more complex or in case of micro programmed control unit, the number of stored micro programs will be much larger. In both of these cases executing a single machine instruction would require more number of clock cycles. This is true not only for the complex operations but also for the simpler operations that are more frequently used during program execution.

14.4 REDUCED INSTRUCTION SET COMPUTERS (RISC)

While CISC architectures were initially designed to reduce memory requirements, complexity of compilers and memory requirements, researchers have later recognized there are several disadvantages to the

approach. Because of these disadvantages, several groups have decided to pursue a completely opposite approach to CISC architecture. The Reduced Instruction Set Computers (RISC) have characteristics that are completely opposite of CISCs. Instead of having complex hardware to simplify the software (viz. compilers), RISC architecture aims to simplify the hardware at the cost of complex software. At the hardware level only the most basic and most frequently used operations are incorporated, in order to keep the number of basic operations as small as possible. As a result of this, even though the compilers are more complex and the number of machine instructions generated for a HLL program is much larger, the time required to execute a single machine instruction is constant. This enables the processor to execute programs at the rate of *one instruction per cycle.* With such a simple set of machine instructions, it becomes possible to have simple hardwired control unit. As a result, a single machine instruction can be executed much faster.

RISC architectures usually ensure faster execution of operations by ensuring that instructions are executed on operands that are stored in registers. Registers are much faster storage devices and therefore if frequently used operands remain on registers, the execution time becomes much faster. Further, this ensures that control unit is even simpler because the only operations that require memory access are simple LOAD and STORE operations. This characteristic of RISCs necessitates a large number of registers available for the processor in order to keep the operands on the registers for a longer duration and further reduce the number of memory access operations.

Since most instructions in RISC architectures allow only register-register operations, the number of addressing modes required is also significantly reduced. This further simplifies the instruction set and the control unit. A simple instruction set in turn leads to simple instruction formats. Instruction length in RISC architectures are fixed and are usually aligned to word boundaries. The number of bits required for opcode is fixed and relatively small because of the small number of available basic operations. The number of bits required for specifying addressing of operands is also

fixed and small, because all operands are in registers and registers have shorter address lengths. The simple instruction format requires simple hardware for decoding in the control unit and a uniform number of instruction fetch operations. All characteristics of RISC architectures are aimed at simplifying the control unit to improve efficiency of instruction execution.

14.4.1 Performance Analysis of RISC

The characteristics with which RISC architectures were designed allow for several performance benefits. While the complexity of compilers is increased with RISC architecture, it is also true that a lot of power to generate more efficient machine code rests with the compiler. The compilers are better enabled to optimize the machine code because the available machine instructions are more primitive. Further, the compiler is not required to find an exact match of machine instruction for every HLL instruction. Instead, depending on how well the compiler is designed, it may be possible to generate a more effective and efficient machine code for a given HLL program.

Most instructions in any HLL program involve only simple operations. As a result the machine code generated by the compiler would involve a lot of simple operations that can be executed much faster due to the simple control unit.

The instruction format in RISC architecture is relatively simple and uniform. As a result the various stages of execution for all instructions are also more or less uniform. This uniformity allows for pipelining to be implemented more efficiently than compared to CISC architectures.

However, most of the perceived advantages of RISC over CISC are based on a lot circumstantial evidence. It is worth noting that there are certain features of CISC architectures that can be incorporated in a RISC design and vice versa for a more well-rounded system. Table 10.1 lists some of the key differences between RISC and CISC architectures.

Table 14.1: Differences between RISC and CISC

CISC	RISC
➢ Emphasis on hardware	➢ Emphasis on software
➢ Includes multi-clock complex instructions	➢ Single-clock, reduced instruction only
➢ Small code sizes, high cycles per second	➢ Low cycles per second, large code sizes
➢ Transistors used for storing complex instructions	➢ Spends more transistors on storage registers



CHECK YOUR PROGRESS

Q.1: Give the full form of CISC and RISC.

.....

.....

.....

Q.2: Give two differences between CISC and RISC.

.....

.....

.....



14.5 LET US SUM UP

- CISC architectures involve incorporating complex operations at the hardware level to enable simple translation of high level language to machine code.
- CISC architectures aim towards simplifying compiler design, support for very high level languages, and reduction of memory requirements for program storage.
- CISC machine instructions allow a large number of basic operations and addressing modes. This increases the number of bits required to specify a single instruction.

- CISC architectures have complex hardware and require a large number of microprograms. This increases the execution time for a single instruction to several clock cycles.
- RISC architectures are aimed at simplifying hardware at the cost of complex compilers. Compilers in RISC architectures are expected to be able to effectively translate HLL programs.
- RISC architectures incorporate only the most basic and frequently used instructions at the hardware level. This significantly reduces the hardware complexity.
- RISC architecture allows execution of a single machine instruction in one clock cycle.
- RISC architectures have fewer operations and addressing modes. This reduces the length of a single instruction and allows for uniformity in the instruction format.
- RISC architectures incorporate a large number of registers as most operations are performed only on register operands. This increases cost due to the number of registers but significantly reduces execution time.
- RISC architectures allow for a more effective implementation of pipelining.
- An architecture that incorporates characteristics of both RISC and CISC would have a more balanced effective approach towards reducing complexity and execution time.



14.6 FURTHER READING

1) Hamacher, V. C., Vranesic, Z. G., Zaky, S. G., Vransic, Z. & Zakay, S. (1984). Computer organization (Vol. 3). New York et al.: McGraw-Hill.
2) Stallings, William. (2000). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.



14.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: The full forms are:

CISC: Complex Instruction Set Computers

RISC: Reduced Instruction Set Computers

Ans. to Q. No. 2: Two differences between CISC and RISC are:

CISC	RISC
➢ Emphasis on hardware	➢ Emphasis on software
➢ Includes multi-clock complex instructions	➢ Single-clock, reduced instruction only



14.8 MODEL QUESTIONS

Q.1: What do you mean by CISC architectures? What are the reasons behind the development of CISC architectures?

Q.2: What are the basic characteristics of CISC architectures?

Q.3: What are the disadvantages of CISC architectures?

Q.4: What do you mean by RISC architectures?

Q.5: What are the basic characteristics of RISC architectures?

Q.6: What are the advantages of RISC over CISC?

Q.7: What are the differences between RISC and CISC?

*** ***** ***

UNIT 15: PARALLEL ARCHITECTURES

UNIT STRUCTURE

15.1 Learning Objectives

15.2 Introduction

15.3 Flynn's Classification of Computer Architecture

- 15.3.1 SISD
- 15.3.2 SIMD
- 15.3.3 MISD
- 15.3.4 MIMD

15.4 Pipelining

- 15.4.1 Arithmetic Pipeline

15.5 Type of Processors

- 15.5.1 Scalar Processor
- 15.5.2 Superscalar Processor and Pipelined Processor
- 15.5.3 Array Processor

15.6 Vector Processor

- 15.6.1 Vector-Register Processor
- 15.6.2 Memory-Memory Vector Processor
- 15.6.3 Vector and Scalar Register
- 15.6.4 Chaining
- 15.6.5 Scatter Gather Operations

15.7 Vector Computing

15.8 Let Us Sum Up

15.9 Further Reading

15.10 Answers to Check Your Progress

15.11 Model Question

15.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about classification of computer architecture based on the notion of instruction and data stream

- describe pipelining technique used in systems to improve the overall performance of the system
- learn about different types of processors and their functioning
- define vector processor and describe its types and instructions
- learn about process of vector computation.

15.2 INTRODUCTION

Previous unit discusses the main functions, architectures and difference between CISC and RISC. This unit reviews architectural evolution of computers starting from uniprocessor systems to multiprocessor system though Flynn's classification of computer architecture. Various parallel computer architectures like pipelined processors, array processors and vector processors were discussed. Finally the unit discusses various techniques in vector processing like chaining, scatter, and gather instruction which aid in improving the performance of the system.

15.3 FLYNN'S CLASSIFICATION OF COMPUTER ARCHITECTURE

With the increase of multiple processing units where a job is broken into multiple segments and each segment is executed concurrently resulted in classification system. Flynn's classification or Flynn's taxonomy of computer architectures is proposed by Michael J. Flynn in the year 1972. The classification is based on the notion of instruction stream and data stream. A stream refers to sequence of instruction or data operated by the computer system. The flow of instruction from memory to processor is called instruction stream and the flow of data between processor and memory is called data stream.

The figure 15.1 shows four categories in which Flynn has classified computer architecture based on instruction and data stream. A conventional uniprocessor system is called SISD (Single instruction stream Single data stream) computers. A vector /array of processors is called SIMD (Single instruction stream Multiple data stream) computers. In MISD (Multiple instruction stream Single data stream) computers, different instructions work

on the same data. Finally in MIMD (Multiple instruction stream Multiple data stream) computers multiple processors each working on different data increases the overall performance of the system.

Data Stream	Instruction Stream: Single	Instruction Stream: Multiple
Multiple	SISD Traditional Von Neumann Single Processor Architecture	MISD Systolic Arrays
Single	SIMD Vector processor/Array processor fine grained parallel computer	MIMD Multiprocessor Systems, Multi Core systems

Figure 15.1: Flynn's Classification of Computer Architecture

15.3.1 SISD

A Single instruction stream single data stream (SISD) system as shown in figure 15.2 is a uniprocessor system. Such systems works on a single instruction and single data stream at a time. Any instructions in SISD systems are processed in a sequential order and are therefore known as sequential computers. Conventional systems were having SISD architecture. Processing in SISD systems involve storing both instructions and data in primary memory. Processing speed of such systems depends on internal data transfer rate. However performance of such systems can be improved with the help of multiple functional units or pipelining. Example of SISD systems includes CDC 6600, IBM PC.

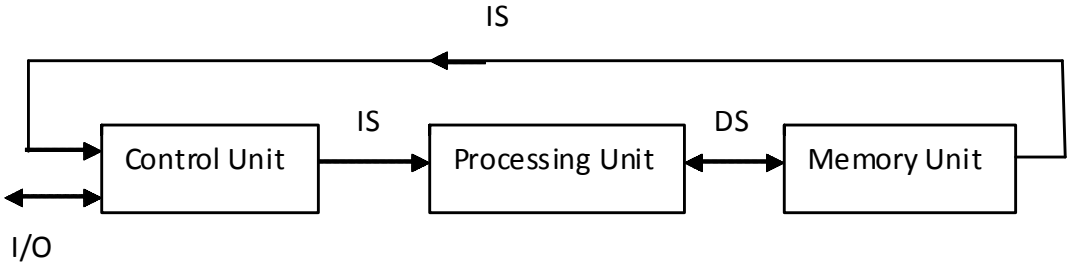


Figure 15.2: SISD Architecture

15.3.2 SIMD

A single instruction stream multiple data stream (SIMD) system as shown in figure 15.3 are multiprocessor systems capable of working on different data streams through a single instruction stream. SIMD systems are used in scientific computation involving vector operations. They are also known as vector processors or array processors. There are n number of processing units each having its own memory and stream of data. All the n processing units receive the same instruction from the control unit. Example of SISD systems includes CRAY vector computers.

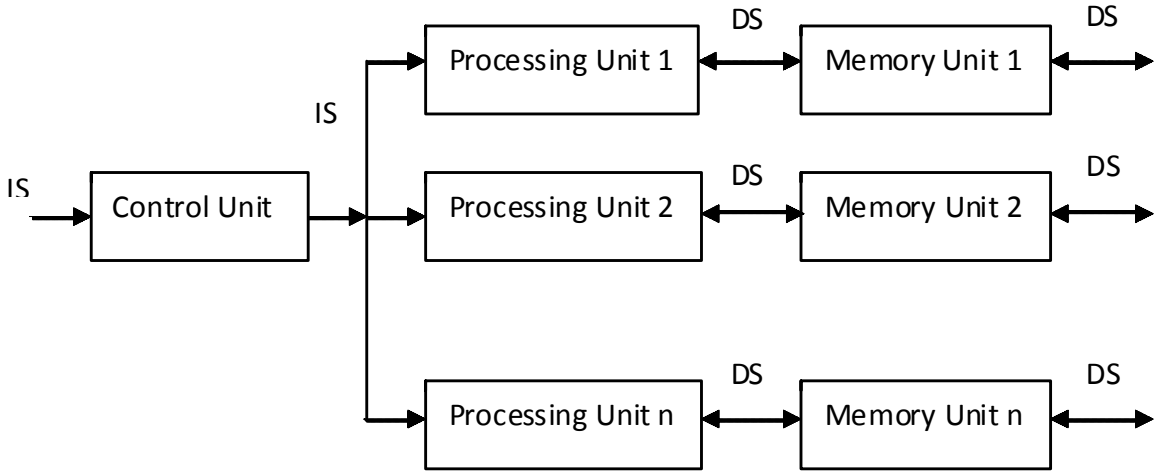


Figure 15.3: SIMD Architecture

15.3.3 MISD

A multiple instruction stream single data stream (MISD) system as shown in figure 15.4 is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions. MISD systems are not practical in the majority of the application and therefore are not available commercially. One such example of MISD system is *systolic array.*

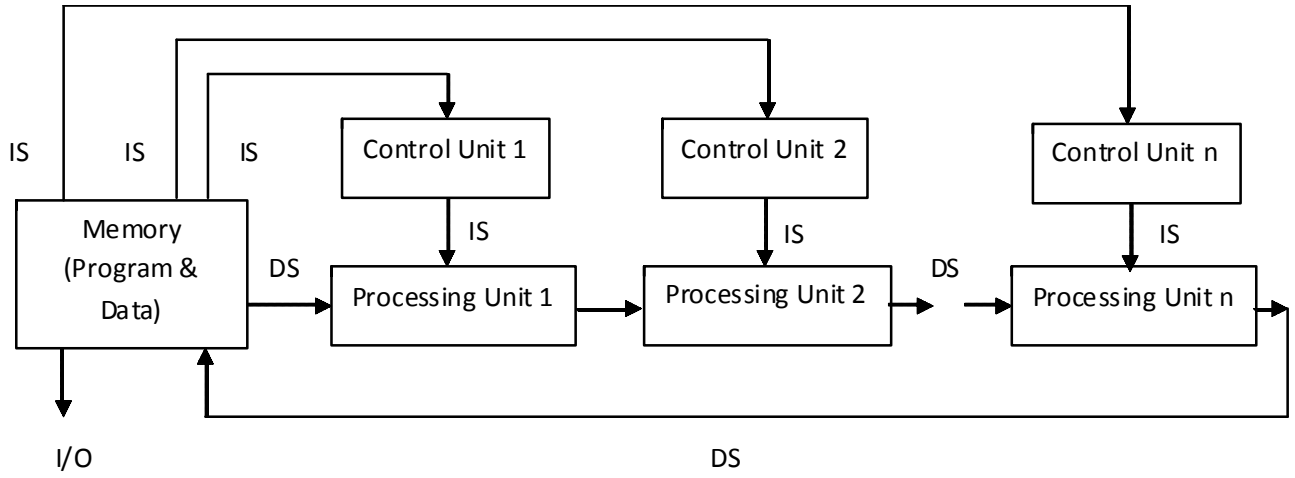


Figure 15.4: SIMD Architecture

15.3.4 MIMD

A multiple instruction stream multiple data stream (MIMD) system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously. The figure 15.5 shows MIMD architecture. Multiple SIMD systems connected together can be viewed as a MIMD system. MIMD systems can be classified into shared-memory MIMD and distributed-memory MIMD based on processing unit-main memory connections.

The shared memory MIMD system also known as tightly coupled multiprocessor system, are the one where all the processing units are connected to a single shared memory. Any form of communication between processing unit takes place with the help of shared memory. Changes done to data in shared memory by one processing unit is visible to all other processing units. In distributed memory MIMD systems or loosely coupled multiprocessor systems all processing units have their own local memory. The communication between processing units takes place through the interconnection.

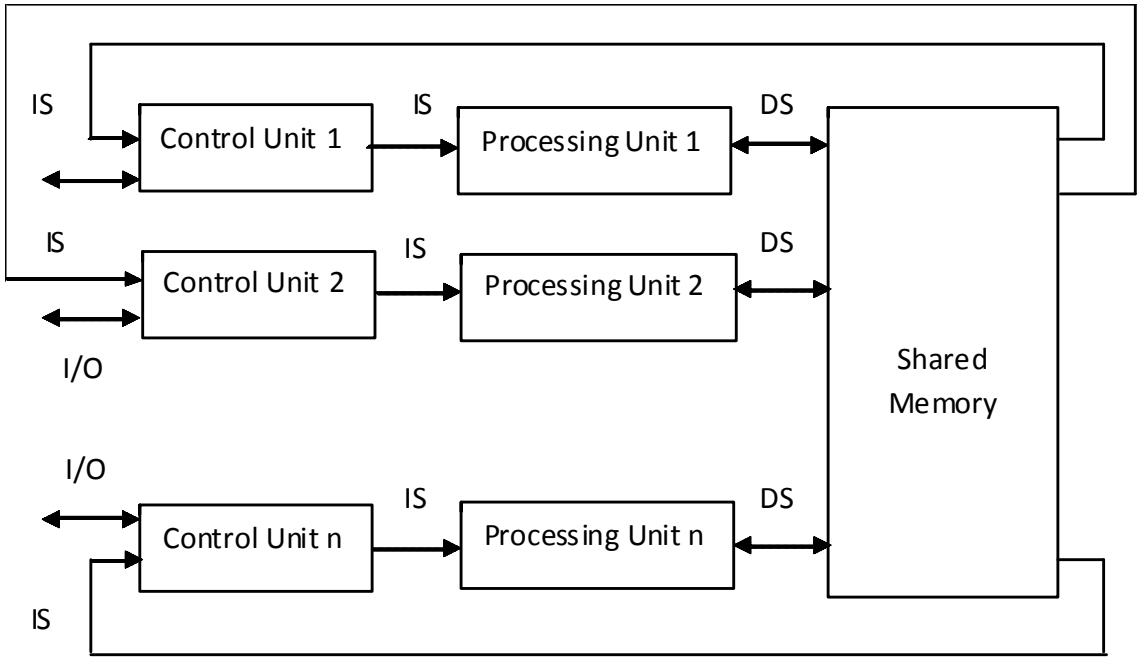


Figure 15.5: MIMD Architecture

15.4 PIPELINING

The technique of executing multiple instructions in an overlapped manner is called pipelining. In pipelining an instruction passes through several stages before completing execution. Each stage is connected to the next stage and so on to form a pipeline structure. An instruction enters from one end of the pipeline and goes through several stages before exiting from other end of the pipeline.

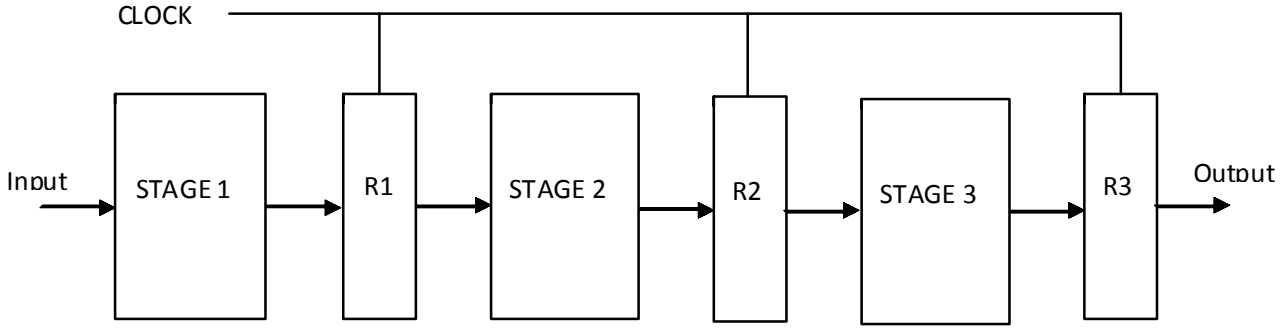


Figure 15.6: Pipeline Structure

Figure 15.6 shows a pipeline system with 3 stages and 3 registers. Each stage consists of an input register and a combinational circuit. The

register is used to hold the data and the combinational circuit to operate on the data. The output of each stage is stored in the register and with each clock pulse the output is move to the next stage. Overall the throughput of the instruction is improved by pipelining.

15.4.1 Arithmetic Pipeline

Arithmetic pipelines are generally found in the majority of the computers. They are used for operations involving floating point and fixed point numbers like addition, subtraction multiplication etc. For example figure 15.7 shows an arithmetic pipeline for floating point addition and subtraction.

The floating point addition and subtraction are completed in four stages:

- Compare exponents.
- Choose exponent and align mantissa
- Add or subtract mantissas
- Adjust exponent and normalize the result.

Registers are used for storing the intermediate results between stages.

In the first stage the exponents x and y are subtracted and the difference is used to determine how many times the mantissa of the smaller exponent must be shifted. In the second stage the larger exponent is chosen as the final result exponent and the mantissa is aligned. In the third stage the two aligned mantissa are added or subtracted. Finally in the fourth stage the result is normalized i.e. in case of overflow the result is shifted right and exponent incremented by number of right shifts, in case of underflow result is shifted left and exponent is decremented by number of left shifts.

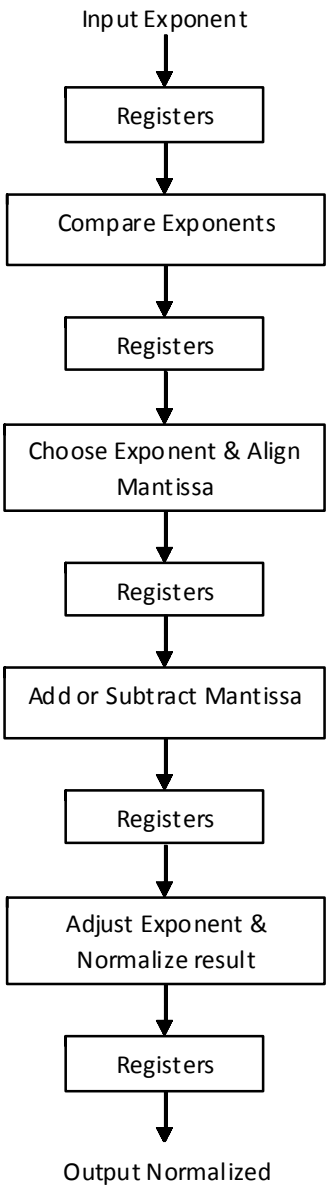


Figure 15.7: Arithmetic Pipeline for Floating Point Addition and Subtraction

The example below gives a clear idea of the process:

$P = 0.1235 \times 10^5$

$Q = 0.432 \times 10^3$

Stage 1: Exponent subtraction: 5 – 3 = 2

Stage 2: $P = 0.1235 \times 10^5$

$Q = 0.00432 \times 10^5$

Stage 3: $R = 0.1235 + 0.00432 \times 10^5$

Stage 4: $Z = 0.12782 \times 10^5$

15.5 TYPE OF PROCESSORS

15.5.1 Scalar Processor

A scalar processor also known as SISD (Single Instruction Stream Single Data Stream) can process a single data item at a time. Scalar processor can process either integer or floating point operands. The simplest scalar processor uses floating point unit to process integer operands. However scalar processor may have separate integer and floating point units for handling integer and floating point operands. AMD 2900, Motorola 68040, Intel 386, Intel 486, M88100 are some examples of scalar processor.

15.5.2 Superscalar Processor and Pipelined Processor

Superscalar Processor: Superscalar processors are found in parallel computing architecture to improve the performance of the system by executing multiple instructions in parallel. A superscalar processor manages multiple instruction pipelines to execute multiple instructions concurrently in a clock cycle. The performance of superscalar processor is highly dependent on the instruction dependency quotient. If the instructions to be executed are independent, then high performance is achieved.

The figure 15.8 shows a superscalar pipeline of degree 2 (i.e., Two instructions can be executed in parallel). There are five stages in the pipeline namely fetch, decode, operand fetch, execute and write. It can be observed from the figure that the superscalar pipeline has two units each of fetch, decode, operand fetch, execute and write, therefore two instructions can be simultaneously executed. In the first clock cycle two instructions are fetched, in the second clock cycle next two instructions are fetched and the process continues. Pentium, DEC Alpha, PowerPC are some of the example of superscalar processor computers.

	1	2	3	4	5	6	7	8
Instruction 1	F	D	OF	E	W			
Instruction 2	F	D	OF	E	W			
Instruction 3		F	D	OF	E	W		
Instruction 4		F	D	OF	E	W		
Instruction 5			F	D	OF	E	W	
Instruction 6			F	D	OF	E	W	
Instruction 7				F	D	OF	E	W
Instruction 8				F	D	OF	E	W

Figure 15.8: A superscalar Pipeline of Degree 2

Pipelined Processor: There are four types of pipelined processors depending upon the following criterions. It is assumed that all the pipelined processors are of k stages.

- Machine Pipeline Cycle (MPC): Time taken by each stage to process an instruction.
- Instruction Issue Rate (ISR): Number of instruction that can be issued simultaneously.
- Instruction Issue Latency (ISL): Time interval between issue of two instructions.
- Instruction Level Parallelism (ILP): Number of instructions that can be executed simultaneously in the pipeline.

Machine Type	Scalar Pipeline	Superscalar Pipeline	Superpipeline	Superpipeline Superscalar
MPC	1	1	1/n	1/n
ISR	1	m	1	m
ISL	1	1	1/n	1/n
ILP	1	m	n	mn

Figure 15.9: Parameters of Pipelined Processor

15.5.3 Array Processor

An array processor or vector processor are multiprocessor systems that performs computation on large arrays of data. A scalar

processor operates on one data item at a time whereas an array processor operates on multiple sets of data item at a time. The instruction however remains same for all the processors executing different data sets.

As for example if a system wants to execute hundred instructions each associated with thousand data items then in order to perform a task a scalar processor executes one by one all the hundred instruction. Whereas an array processors take the first instruction (from hundred) and executes all thousand data items and then progress to the next instruction.

There are mainly two types of array processors:

- Attached Array Processors
- SIMD Array Processors

Attached Array Processors: An attached array processor shown in figure 15.10 is used to improve the overall performance of a general purpose computer. It is designed as a peripheral device for supporting a general purpose computer in solving complex applications by means of vector processing with multiple functional units. It consists of more than one functional unit like floating point adders, multipliers etc. Such processor can also be programmed by the user to support different types of complex applications.

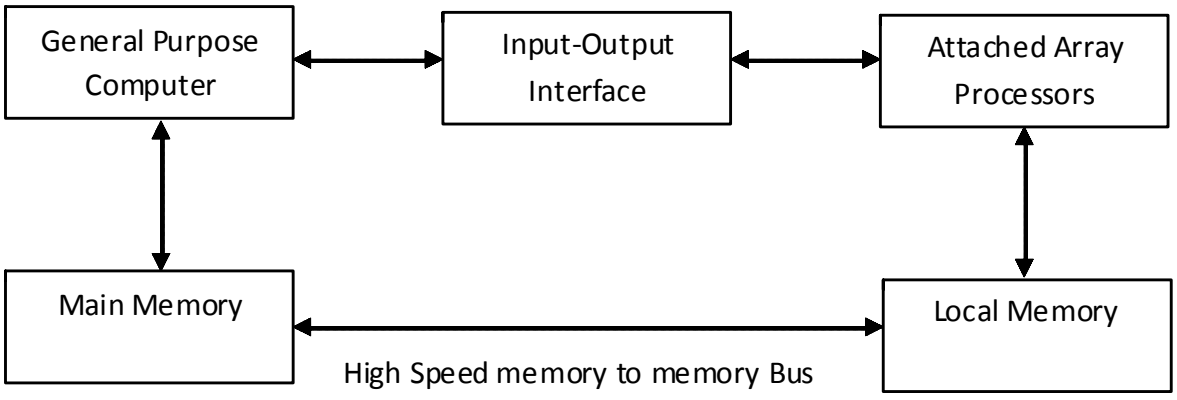


Figure 15.10: Attached Array Processor

SIMD Array Processors: Refer to section 15.3.2

15.6 VECTOR PROCESSOR

Vector processors are found mainly in supercomputers combining pipelining and interleaved memory unit. It is used mainly in scientific and multimedia applications involving processing of huge volume of data. It is capable of processing entire vector in single instruction. The operands in the instructions are vectors instead of a single element. One of the advantages of vector processors is less number of fetch and decode instructions.

Vector processor uses many optimization schemes to improve performance of the system such as use of memory banks to reduce load/ store latency, use of strip mining technique to adjust the size mismatch between vector operands and vector registers, vector chaining to resolve data dependency between vector instructions etc.

Advantages of Vector processing:

- Programs are smaller in size as the number of instructions is quite less.
- As each data in registers is actually used by the vector processor therefore wastage in memory access is significantly less compared to cache memory.
- Requirement of power is limited to only functional unit and register buses during vector operation.

Based on how operands are fetched in vector processors is categorized into two types:

- Vector-Register Processor
- Memory-Memory Vector Processor

15.6.1 Vector-Register Processor

It requires that all the operations performed in the vector processor use the source operands and destination operands as vector registers. However there is a small disadvantage initially that vector data in memory is must be divided into fixed length segments so that can be placed in vector register. But once the pipelining starts this disadvantage is nullified.

15.6.2 Memory-Memory Vector Processor

Such processors allow source operand and destination operand to be routed directly to the arithmetic logic unit (ALU). Once the processing is completed in the ALU, the result is routed back to memory. However due to memory latency the time between initializing the first instruction and the getting the first output from the pipeline is quite large.

15.6.3 Vector and Scalar Register

A vector consists of fixed length set of data words and a set of vector registers to hold the data. These are specialized registers used in vector processing to hold vector data (multiple data) in a single register. A vector registers is loaded with input vector data from memory and are like FIFO queues that can hold multiple data items at a time. In a vector pipeline, vector data elements are sent one at a time from vector register and the output of the pipeline is sent to the vector register one at a time. There are several vector registers in a vector processor and also have specific instruction to load, perform operation and store register contents to memory. In supercomputers like CRAY, vector registers are of fixed length 64 bits. In Fujitsu VP2000 vector registers are reconfigurable to match the length of the vector operands.

Scalar registers unlike vector register is used to hold single/ scalar data item. There are separate set of instructions that work on scalar register.

15.6.4 Chaining

The concept of chaining or vector chaining in vector processor involves forwarding the result of a pipeline vector operation to the next pipeline without storing the result in vector register on completion. The vector chaining technique was first used in Cray 1 supercomputer in 1976. Chaining technique enables in reducing

the startup time i.e. time to initialize the pipeline with first instruction and obtaining the first result compared to memory–memory systems. It is extension to internal data forwarding technique bypassing paths between functional units. The concept of internal data forwarding is common in microprocessors but chaining extends the concept with vectors of larger length. From a compilers point of view chaining allows vectorization of dependent instructions.

15.6.5 Scatter Gather Operations

In vector processing there are two instructions Scatter and Gather which are useful in handling sparse matrices or sparse vectors. In gather instruction non zero elements of sparse matrix are stored in vector register using indices which are itself indexed. The figure 15.11 shows a clear picture of Gather Operation.

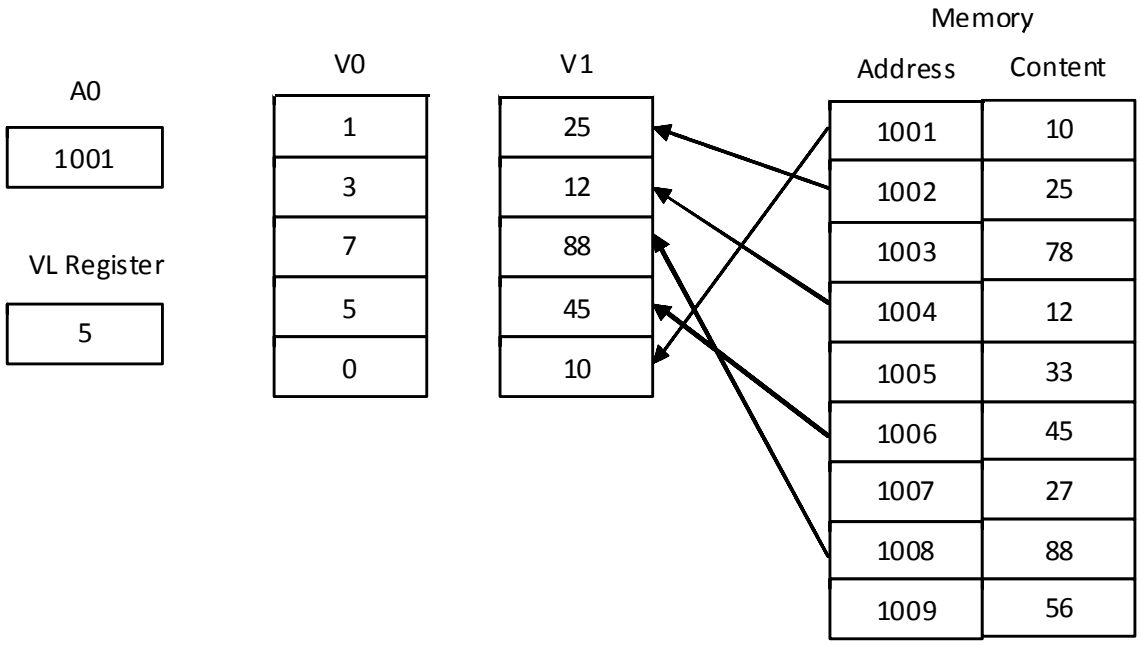


Figure 15.11: Gather Operation

Here the gather instruction the moves non sequential memory contents 25, 12, 88, 45, 10 to register V1 of length (length=5) specified by register VL. The contents are moved based on the index specified in the V0 register. The register A0 is used as the base register.

The scatter instruction on the other hand stores the elements of the vector register to memory specified by the index available in another vector register. It does the opposite mapping operation as to gather. The figure 15.12 shows a scatter operation. Here contents of vector register V1, 59, 62, 08, 29, 17 are stored in non sequential memory location specified by the index value in register V0. The register A0 is used as the base register.

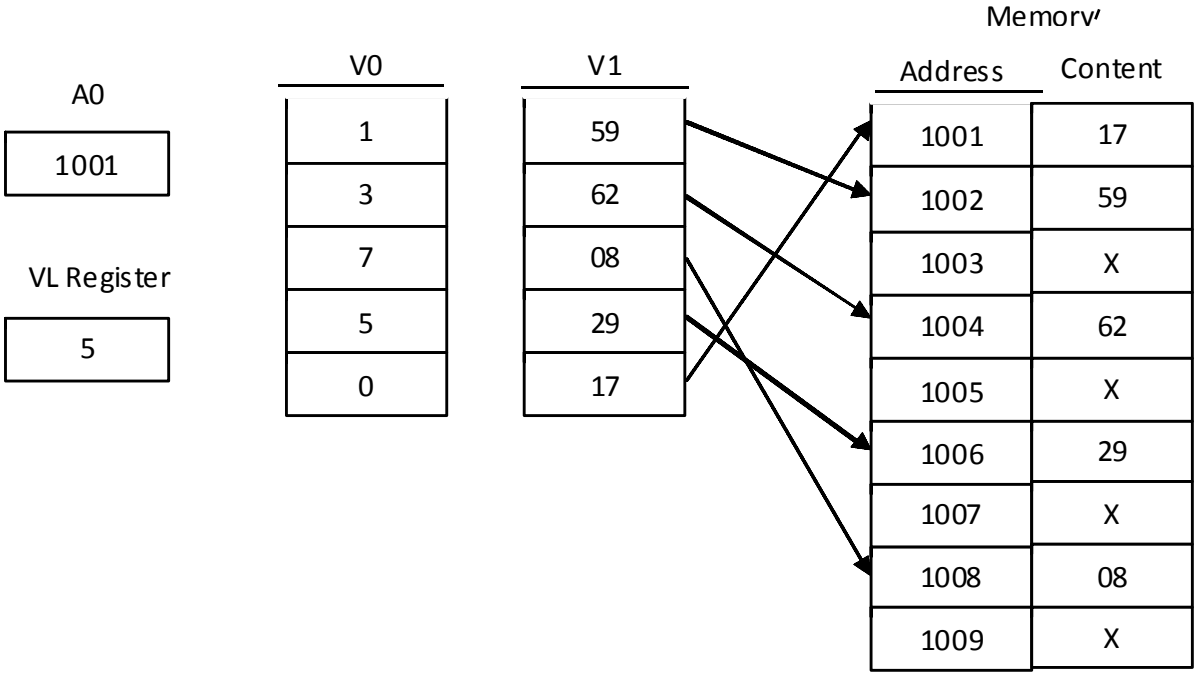


Figure 15.12: Scatter Operation

15.7 VECTOR COMPUTING

The main task in design of a supercomputer is to perform arithmetic operations on vector of floating point numbers. In case of general purpose computer there will be n number of iterations required for a vector of length n to perform any arithmetic operation. Considering multiplication of two vectors A, B of size N X N and storing the result back to vector C, a vector computation will help in parallelizing the entire operation.

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \, X \, b_{k,j}$$

Where $a_{i,j}$, $b_{i,j}$, $c_{i,j}$ are respectively elements of the vector A, B, C.

In scalar processing the code can be written as:

```
for (i = 1; i < N; i++)
    {
        for (j = 1; j < N; j++)
        {
            c[i][j] = 0;
            for (k = 1; k < N; k++)
            c[i][j] = c[i][j] + a[i][k]*b[k][j];
        }
    }
```

In vector processing the code can be written as:

```
for (i = 1; i < N; i++)
{
    c[i][j] = 0;  (for all j=1,N)
    for (k = 1; k < N; k++)
    c[i][j] = c[i][j] + a[i][k]*b[k][j]; (for all j=1,N)
}
```

The vector computation specifies that the statement (for all j=1,N) will be carried out as a single operation on all indices of j. And all the elements will be computed in parallel for each index of i. The code requires N^2 vector multiplication compared to N^3 scalar multiplications.



CHECK YOUR PROGRESS

Q.1: CRAY systems are an example of .....

a) SISD b) SIMD

c) MISD d) MIMD

Q.2: In an arithmetic pipeline addition of two numbers 0.31511×10^4 and 0.01972×10^3 will result in .....

a) 3.17082×10^4 b) 3.17082×10^3

c) 3.17082×10^4 d) 0.317082×10^4

Q.3: Registers are used in a pipeline to .....

a) Read data b) Write data

c) Hold data d) Process data

Q.4: Motorola 68040 is an example of .....

a) Scalar processor b) Superscalar processor

c) Superpipeline processor d) Pipelined processor

Q.5: A superscalar pipeline (5 stages) of degree 3 will need ..... cycles to complete 9 instructions.

a) 6 b) 7 c) 8 d) 9

Q.6: Instruction Issue Latency (ISL) in a pipelined processor means–

a) Time interval between issuing of first and last instruction.

b) Time interval between completion of first and second instruction.

c) Time taken to complete execution of first instruction.

d) Time interval between issuing of two instructions.

Q.7: Instruction Level Parallelism in a pipelined processor means–

a) Number of instructions in the pipeline.

b) Number of instructions that can be completed simultaneously in the pipeline.

c) Number of instructions that can be executed simultaneously in the pipeline.

d) None of the above

Q.8: Array processors are ..... systems

a) SISD b) MISD c) SIMD d) MIMD

Q.9: Attached array processor is a .....

a) Pipelined processor b) Powerful processor

c) Peripheral device d) All of the above

Q.10: Advantages of Vector processing is .....

a) Very fast processing

b) Parallel processing

c) Number of instructions is quite less

d) All of the above

Q.11: Vector register processor uses source and destination operands as–

a) Scalar b) Vector

c) Scalar and vector d) Any type

Q.12: The concept of internal data forwarding in vector processing is known as–

a) Chaining b) Scatter c) Gather d) Pipeline

Q.13: To handle sparse matrix ..... instruction is used.

a) Scatter b) Gather

c) Both (a) and (b) d) Sparse



15.8 LET US SUM UP

- Flynn's taxonomy of computer architectures is proposed by Michael J. Flynn in the year 1972. The classification is based on the notion of instruction and data stream.
- A Single instruction stream single data stream (SISD) system is a uniprocessor system.
- A single instruction stream multiple data stream (SIMD) system are multiprocessor systems capable of working on different data streams through a single instruction stream.
- A multiple instruction stream single data stream (MISD) system is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions.
- A multiple instruction stream multiple data stream (MIMD) system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously.
- The technique of executing multiple instructions in an overlapped manner is called pipelining.
- Arithmetic pipelines are used for operations involving floating point and fixed point numbers like addition, subtraction multiplication etc.
- Superscalar processors are found in parallel computing architecture to

improve the performance of the system by executing multiple instructions in parallel.

- There are four types of pipelined processors namely scalar, superscalar, superpipeline and superscalar superpipeline.
- An array processor or vector processor are multiprocessor systems that performs computation on large arrays of data.
- An attached array processor is designed as a peripheral device for supporting a general purpose computer in solving complex applications by means of vector processing with multiple functional units.
- Vector processors are found mainly in supercomputers combining pipelining and interleaved memory unit.
- The concept of chaining or vector chaining in vector processor involves forwarding the result of a pipeline vector operation to the next pipeline without storing the result in vector register on completion.
- In vector processing there are two instructions Scatter and Gather which are useful in handling sparse matrices or sparse vectors.



15.9 FURTHER READING

1) Hamacher, Carl. (5th Edition). *Computer Organization*. McGraw Hill.

2) Mano, M. Morris; Mall, Rajib. (3rd Edition). *Computer System Architecture*. Pearson

3) Stallings, William. (2000). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.



15.10 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: (b) **Ans. to Q. No. 2:** (b)

Ans. to Q. No. 3: (c) **Ans. to Q. No. 4:** (a)

Ans. to Q. No. 5: (b) **Ans. to Q. No. 6:** (d)

Ans. to Q. No. 7: (c) **Ans. to Q. No. 8:** (c)

Ans. to Q. No. 9: (c) **Ans. to Q. No. 10:** (d)

Ans. to Q. No. 11: (c) **Ans. to Q. No. 12:** (a)

Ans. to Q. No. 13: (c)



15.11 MODEL QUESTION

Q.1: Discuss Flynn's classification of computer architecture.

Q.2: Differentiate between shared memory and distributed memory MIMD systems.

Q.3: Explain how pipelining can increase the performance of a system compared to a single processor system.

Q.4: Consider the two floating point numbers given below; discuss in detail the steps to be taken by an arithmetic pipeline to add these numbers.

$P = 0.3521 \times 10^4$

$Q = 0.01920 \times 10^2$

Q.5: Differentiate between superscalar processor and superpipeline processor.

Q.6: Briefly describe the parameters on which different pipelined processors are measured in terms of their performance.

Q.7: How an attached array processor helps in improving the overall performance of a general purpose computer?

Q.8: Differentiate between vector register processor and memory-memory vector processor.

Q.9: The concept of internal data forwarding in case of vector processing is chaining. Explain?

Q.10: Explain how sparse matrix is handled by vector processor?

Q.11: Time complexity of a program can be reduced by vector computing. How?

*** ***** ***



REFERENCES

1) Chaudhuri, P. Pal. (2nd Edition, 2003). *Compter Organization and Design*. PHI Learning Pvt. Ltd.
2) Floyd, Thomas L. *Digital Fundamentals*. Pearson Education.
3) Gill, D. N. S. & Dixit, J. B. (2008). *Digital Design and Computer Organisation*. Firewall Media.
4) Godse, A. P., Godse, D. A. *Computer Architecture*. Technical Publications Pune.
5) Leach, D. P. & Malvino, A. P. (1994). *Digital Principles and Applications*. Glencoe/McGraw-Hill.
6) Hammacher, Carl. (5th Edition) (International Edition, 2002). *Computer Organization*. McGrawHill.
7) Hamacher, V. Carl, Vranesic, Zvonko G., Zaky, Safwat G. *Computer Organization*. McGraw-Hill International Edition.
8) Lee, S. C. (1976). Digital Circuits and Logic Design.
9) Mano, M. M. (2006). *Computer Systems Architecture*. Pearson Education.
10) Mano, M. (1979). *Digital Logic, Computer Design*. Englewood Cliffs Prentice-Hall.
11) Mano, M. Morris; Mall, Rajib. (3rd Edition). *Computer System Architecture*. Pearson.
12) Null, L. & Lobur, J. (2014). *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers.
13) Rajaraman, V. (4th Edition, 2008). *Fundamentals of Computers*. PHI Publication.
14) Stallings, William. (2004). *Computer Organization and Architecture Designing for Performance*. Pearson Education India.
15) Talukdar, P. H. *Digital Techniques*. N. L. Publications.
16) Zaky, S. G., Hamacher, V. C. & Vranesic, Z. G. (1996). *Computer Organization*. New York: McGraw-Hill.

Name of the Paper: .....
Course Code: .....

Centre for Internal Quality Assurance (CIQA)
Krishna Kanta Handiqui State Open University
City Office: Housefed Complex, Guwahati-781006

Learner's Feedback on Course

Dear Learner,

Regarding the course as mentioned above, we would like to know your opinions and comments so as to improve the quality of self learning materials in future. Please respond to the following statements by ticking the number you feel most reflect your opinion. After completion of the additional comments, please detach the page and send/mail the same to us at the address given below.

The Director, Centre for Internal Quality Assurance, KKHandiqui State Open University
Housefed Complex, Dispur, Guwahati-781006
(E-mail id: ciqa@kkhsou.in)

1) Approximately how many hours did you spend for studying the units in the course? ☐☐☐

2) Please give your opinions (by ✓ mark) to the following items based on your reading of the block:

Sl. No.	Statements	Strongly Agree	Agree	Neither Agree nor Disagree	Disagree	Strongly Disagree
I)	The SLMs of the course fulfil the learning objectives set out in the beginning of each unit					
II)	The units of the SLM could satisfy my academic needs and expectations					
III)	The Unit writers have excellent knowledge about the course contents					
IV)	Language and contents of the units were presented at a level which I could readily understand					
V)	Ample opportunity for participation in the activities provided in the units					
VI)	Used enough Illustrations (Diagrams, tables etc.) for conceptual clarity					
VII)	Quality of content is engaging, relevant, and up-to-date					
VIII)	The self check questions are very helpful					
IX)	The Possible/model questions and the answers to check my progress have benefited me a lot					

Additional Comments: *(Please feel free to provide your open comments)*

1) Which aspects of the SLM, according to you, worked well?

.....

.....

2) What sort of changes/improvements do you feel KKHSOU could implement to improve the overall quality of the SLM?

.....

.....

Thank you for taking the time to complete this form.