

Logic Design & Computer Organization

(Code : 214442)

Semester III – Information Technology

(Savitribai Phule Pune University)

Strictly as per the New Choice Based Credit System Syllabus (2019 Course)

Savitribai Phule Pune University w.e.f. academic year 2020-2021

J. S. Katre

M.E. (Electronics and Telecommunication)
Formerly, Assistant Professor
Department of Electronics Engineering
Vishwakarma Institute of Technology (V.I.T.), Pune.
Maharashtra, India

Harish G. Narula

Formerly Assistant Professor (Senior)
Department of Computer Engineering
D. J. Sanghvi College of Engineering, Mumbai.
Maharashtra, India.

Prof. Nilesh N. Thorat

PhD (Pursuing), M. Tech in CS, B.E. in CE
Assistant Professor in IT Department
JSPM's BSIOTR, Wagholi, Pune
Maharashtra, India.



PO130A Price ₹ 365/-



Logic Design & Computer Organization (Code : 214442)

(Semester III, Information Technology, Savitribai Phule Pune University)

J. S. Katre, Harish G. Narula, Nilesh N. Thorat

Copyright © by Authors. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Printed in India : January 2002

First Edition : August 2020 (**TechKnowledge Publications**)

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

ISBN : 978-93-89889-45-1

Published by :

TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner,

Pune - 411 009. Maharashtra State, India

Ph : 91-20-24221234, 91-20-24225678.

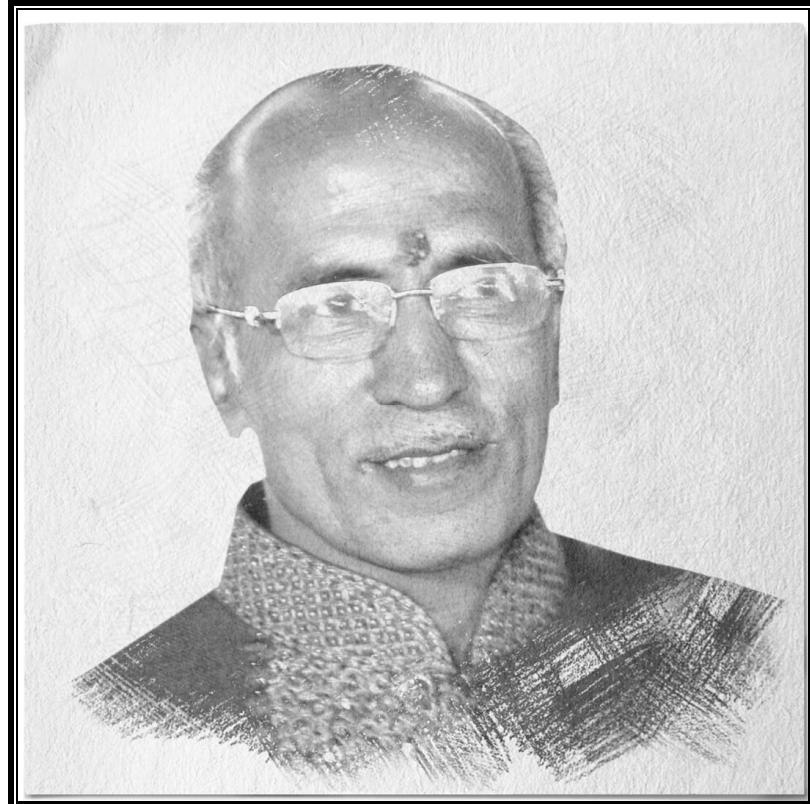
Email : info@techknowledgebooks.com,

Website : www.techknowledgebooks.com

[214442] (FID : PO130) (Book Code : PO130A)

(Book Code : PO130A)

*We dedicate this Publication soulfully and wholeheartedly,
in loving memory of our beloved founder director,
Late Shri. Pradeepji Lalchandji Lunawat,
who will always be an inspiration, a positive force and strong support
behind us.*



“My work is my prayer to God”

- Lt. Shri. Pradeepji L. Lunawat

*Soulful Tribute and Gratitude for all Your
Sacrifices, Hardwork and 40 years of Strong Vision...*

Syllabus...

Logic Design & Computer Organization : Sem. III, Information Technology (SPPU)

Teaching Scheme : Theory (TH) : 03 Hrs. /Week	Credit Scheme : 03	Examination Scheme : Mid-Semester : 30 Marks End-Semester : 70 Marks
--	-------------------------------------	---

Prerequisite Courses : if any: Basics of electronics engineering.

Companion Course : if any :

Course Objectives :

1. To make undergraduates, aware of different levels of abstraction of computer systems from hardware perspective.
2. To make undergraduates, understand the functions, characteristics of various components of Computer& in particular processor & memory.

Course Outcomes : On completion of the course, students will be able to :

CO1 : Perform basic binary arithmetic and simplify logic expressions.

CO2 : Grasp the operations of logic ICs and Implement combinational logic functions using ICs.

CO3 : Comprehend the operations of basic memory cell types and Implement sequential logic functions using ICs.

CO4 : Elucidate the functions and organization of various blocks of CPU.

CO5 : Understand CPU instruction characteristics, enhancement features of CPU.

CO6 : Describe an assortment of memory types (with their characteristics) used in computer systems and basic principle of interfacing input, output devices.

Course Contents

Unit 1

Introduction To Digital Electronics :

Digital Logic families: Digital IC Characteristics; **TTL** : Standard TTL characteristics, Operation of TTL NAND gate; **CMOS** : Standard CMOS characteristics, operation of CMOS NAND gate; Comparison of TTL and CMOS. Signed Binary number representation and Arithmetic: Sign Magnitude, 1's complement and 2's complement representation, unsigned Binary arithmetic (addition, subtraction, multiplication, and division), subtraction using 2's complement; IEEE Standard 754 Floating point number representations. Codes: Binary, BCD, octal, Hexadecimal , Excess-3 , Gray code and their conversions Logic minimization: Representation of logic functions: logic statement, truth table, SOP form, POS form; Simplification of logical functions using K-Maps up to 4 variables.

Case Study : 1) CMOS 4000 series ICs 2) practical applications of various codes in computers 3) four basic arithmetic operations using floating point numbers in a calculator. **(Refer chapters 1, 2, 3 and 4)**

Unit 2

Combinational Logic Design :

Design using SSI chips: Code converters, Half- Adder, Full Adder, Half Subtractor, Full Subtractor, n bit Binary adder. Introduction to MSI chips: Multiplexer (IC 74153), Demultiplexer (IC 74138), Decoder (74238) Encoder (IC 74147), Binary adder (IC 7483) Design using MSI chips: BCD adder and subtractor using IC 7483, Implementation of logic functions using IC 74153 & 74138.

Case Study : Use of combinational logic design in 7 segment display interface

(Refer chapters 5)

Unit 3

Sequential Logic Design :

Introduction to sequential circuits: Difference between combinational circuits and sequential circuits; Memory element-latch and Flip-Flop. Flip- Flops: Logic diagram, truth table and excitation table of SR, JK, D, T flip flops; Conversion from one FF to another, Study of flip flops with regard to asynchronous and synchronous, Preset and Clear, Master Slave configuration ; Study of 7474, 7476 flip flop ICs. Application of flip-flops: Counters- asynchronous, synchronous and modulo n counters, study of 7490 modulus n counter ICs and their applications to implement mod counters; Registers- shift register types (SISO, SIPO, PISO and PIPO) and applications.

Case Study : Use of sequential logic design in a simple traffic light controller

(Refer chapters 6, 7 and 8)

Unit 4

Computer Organization and Processor :

Computer organization and computer architecture, organization, functions and types of computer units- CPU(typical organization, Functions, Types), Memory (Types and their uses in computer), IO(types and functions) and system bus (Address, data and control, Typical control lines, Multiple-Bus Hierarchies); Von Neumann and Harvard architecture; Instruction cycle Processor: Single bus organization of CPU; ALU(ALU signals, Functions and types); Register (types and functions of user visible, control and status registers such as general purpose, address registers, data registers, flags, PC, MAR, MBR, IR) and control unit (control signals and typical organization of hard wired and microprogrammed CU). Micro Operations (fetch, Indirect, Execute, interrupt) and control signals for these micro operations. Case Study : 8086 processor , PCI bus

(Refer chapter 9)

Unit 5

Processor Instructions and Processor Enhancements :

Instruction : elements of machine instruction ; instruction representation (Opcode and mnemonics, Assembly language elements) ; Instruction Format and 0-1-2-3 address formats, Types of operands Addressing modes; Instruction types based on operations (functions and examples of each); key characteristics of RISC and CISC; Interrupt: its purpose, types, Classes and interrupt handling (ISR, multiple interrupts), Exceptions; Instruction pipelining (operation and speed up) Multiprocessor systems: Taxonomy of Parallel Processor Architectures, two types of MIMD clusters and SMP (organization and benefits) and multicore processor (various Alternatives and advantages of multicores), Typical features of multicore intel core i7.

Case Study : 8086 Assembly language programming

(Refer chapter 10)

Unit 6

Memory & Input / Output Systems :

Memory Systems: Characteristics of Memory Systems, Memory Hierarchy, signals to connect memory to processor, memory read and write cycle, Characteristics of semiconductor memory: SRAM, DRAM and ROM, Cache Memory – Principle of Locality, Organization, Mapping functions, write policies, Replacement policies, Multilevel Caches, Cache Coherence, Input / Output Systems : I/O Module, Programmed I/O, Interrupt Driven I/O, Direct Memory Access (DMA). Case Study : USB flash drive

(Refer chapter 11)

**Case Studies may be assigned as self-study to students
and to be excluded from theory examinations**



**Unit 1****Chapter 1 : Digital Logic Families 1-1 to 1-22**

Syllabus : Digital IC characteristics; TTL : Standard TTL characteristics, Operation of TTL NAND gate; CMOS : Standard CMOS characteristics, Operation of CMOS NAND gate; Comparison of TTL & CMOS.

Case Study : CMOS 4000 series ICs.

1.1	Logic Families	1-2
1.1.1	Classification Based on Circuit Complexity	1-2
1.2	Classification of Logic Families	1-2
1.2.1	Classification Based on Devices Used ...	1-2
1.3	Characteristics of Digital ICs	1-3
1.3.1	Voltage and Current Parameters	1-3
1.3.2	Fan-in and Fan-out	1-4
1.3.3	Noise Margin	1-5
1.3.4	Propagation Delay (Speed of Operation)	1-5
1.3.5	Power Dissipation	1-6
1.3.6	Operating Temperature	1-6
1.3.7	Figure of Merit (Speed Power Product (SPP)).....	1-7
1.3.8	Invalid Voltage Levels	1-7
1.3.9	Current Sourcing and Current Sinking ...	1-7
1.3.10	Power Supply Requirements	1-7
1.4	Transistor-Transistor Logic (TTL)	1-7
1.4.1	The Multiple Emitter Transistor	1-7
1.4.2	Two Input TTL-NAND Gate (Totempole Output)	1-8
1.4.3	Totem-pole (Active Pull up) Output Stage	1-10
1.4.4	Unconnected Inputs	1-11
1.4.5	Clamping Diodes	1-11
1.4.6	5400 Series	1-11
1.4.7	Three Input TTL NAND Gate	1-11
1.5	Open Collector Outputs (TTL)	1-12

1.5.1	Disadvantages of Open Collector Output	1-13
1.5.2	Advantage of Open Collector Output ...	1-13
1.5.3	Wired ANDing	1-13
1.5.4	Comparison of Totem-pole and Open Collector Outputs	1-14
1.6	Standard TTL Characteristics	1-15
1.6.1	Advantages of TTL	1-16
1.6.2	Disadvantages of TTL	1-16
1.7	MOS - Logic Family	1-16
1.8	CMOS Logic	1-16
1.8.1	CMOS NAND Gate	1-16
1.8.2	CMOS Series	1-18
1.9	Standard CMOS Characteristics	1-18
1.9.1	Power Supply Voltage	1-18
1.9.2	Logic Voltage Levels	1-18
1.9.3	Noise Margins	1-18
1.9.4	Power Dissipation	1-19
1.9.5	Fan Out	1-19
1.9.6	Switching Speed	1-19
1.9.7	Unconnected Inputs	1-19
1.9.8	Advantages of CMOS	1-20
1.9.9	Disadvantages of CMOS	1-20
1.10	Comparison of CMOS and TTL	1-20
1.11	Case Study CMOS 4000 series ICs	1-21
• Review Questions		1-21

Unit 1**Chapter 2 : Number Systems and Codes 2-1 to 2-28**

Syllabus : Binary, BCD, Octal, Hexadecimal, Excess-3, Gray code and their conversions.

Case study : Practical applications of various codes in computers.

2.1	Introduction	2-2
2.2	System or Circuit	2-2
2.2.1	Digital Systems	2-2
2.3	Binary Logic and Logic Levels	2-2
2.3.1	Positive Logic	2-2
2.3.2	Negative Logic	2-2
2.4	Number Systems	2-3



2.4.1	Important Definitions Related to All Numbering Systems	2-3	2.14.2	Hex to Other Systems	2-18
2.4.2	Various Numbering Systems	2-3	2.15	Concept of Coding	2-21
2.5	The Decimal Number System	2-4	2.16	Classification of Codes	2-21
2.5.1	Characteristics of a Decimal System	2-4	2.16.1	Weighted Binary Codes	2-21
2.6	The Binary Number System	2-4	2.16.2	Non Weighted Codes	2-21
2.6.1	Binary Number Formats	2-5	2.16.3	Alphanumeric Codes	2-21
2.7	Octal Number System	2-5	2.17	Binary Coded Decimal (BCD) Code	2-22
2.8	Hexadecimal Number System	2-5	2.17.1	Comparison with Binary	2-22
2.9	Conversion of Number Systems	2-6	2.17.2	Advantages of BCD Codes	2-23
2.10	Conversions Related to Decimal System	2-6	2.17.3	Disadvantages	2-23
2.10.1	Conversion from any Radix r to Decimal	2-6	2.18	Non – weighted Codes	2-23
2.10.2	Conversion from Decimal to Other Systems	2-7	2.18.1	Excess – 3 Code	2-23
2.10.2.1	Successive Division for Integer Part Conversion	2-7	2.19	Gray Code	2-24
2.10.2.2	Successive Multiplication for Fractional Part Conversion	2-9	2.19.1	Application of Gray Code	2-25
2.10.2.3	Conversion of Mixed Decimal Number to Any Other Radix	2-10	2.19.2	Advantages of Gray Code	2-25
2.11	Conversion from Binary to Other Systems	2-11	2.19.3	Gray-to-Binary Conversion	2-25
2.11.1	Conversion from Binary to Decimal	2-11	2.19.4	Binary to Gray Conversion	2-26
2.11.2	Binary to Octal Conversion	2-11	2.20	Code Conversions	2-26
2.11.3	Binary to Hex Conversion	2-12	2.20.1	Binary to BCD Conversion	2-26
2.12	Conversion from Other Systems to Binary System	2-12	2.20.2	BCD to Binary Conversion	2-26
2.12.1	Conversion from Decimal to Binary	2-12	2.20.3	BCD to Excess – 3	2-26
2.12.2	Octal to Binary Conversion	2-12	2.20.4	Excess – 3 to BCD Conversion	2-27
2.12.3	Hex to Binary Conversion	2-13	•	Review Questions	2-27
2.13	Conversion from Octal to Other Systems	2-14	Unit 1		
2.13.1	Octal to Hex Conversion	2-14	Chapter 3 : Binary Arithmetic		
2.13.2	Conversion from Other Systems to Octal	2-14	3-1 to 3-28		
2.14	Conversions Related to Hexadecimal System	2-16	<p>Syllabus : Signed binary number representation and arithmetic : Sign magnitude, 1's complement and 2's complement representation, Unsigned binary arithmetic (Addition, subtraction, multiplication and division), Subtraction using 2's complement; IEEE standard 754 floating point number representations.</p> <p>Case study : Four basic arithmetic operations using floating point numbers in a calculator.</p>		
2.14.1	Other Systems to Hex	2-16	3.1	Introduction	3-2
			3.2	Unsigned Binary Numbers	3-2



3.2.1	Important Features of Unsigned Numbers	3-2	3.8.3	AND Operator	3-19
3.2.2	Unsigned Binary Arithmetic	3-2	3.8.4	OR Operator	3-19
3.2.3	Binary Addition	3-2	3.8.5	Logic Gates	3-20
3.2.4	Sum and Carry	3-2	3.8.6	Gates, Symbols and Boolean Expression	3-20
3.2.5	Binary Subtraction	3-3	3.8.7	Postulates	3-20
3.2.6	Subtraction and Borrow	3-3	3.9	Definition of Boolean Algebra	3-21
3.2.7	Binary Multiplication	3-4	3.9.1	Boolean Postulates and Laws	3-21
3.2.8	Binary Division	3-4	3.10	Two Valued Boolean Algebra	3-22
3.3	Sign-Magnitude Numbers	3-4	3.11	Basic Theorems and Properties of Boolean Algebra	3-23
	3.3.1 Range of Sign-Magnitude Numbers	3-5	3.11.1	Duality	3-23
3.4	Complements	3-5	3.11.2	Basic Theorems	3-23
	3.4.1 1's Complement	3-5	3.11.3	De-Morgan's Theorems	3-25
	3.4.2 Representation of Signed Numbers using 1's Complement	3-6	3.11.4	Operator Precedence	3-25
	3.4.3 2's Complement	3-6	3.12	Boolean Expression and Boolean Function	3-26
	3.4.4 Representation of Signed Numbers using 2's Complement	3-6	3.12.1	Truth Table Formation	3-26
	3.4.5 Signed Complement Numbers	3-7	3.12.2	Examples on Reducing the Boolean Expression	3-26
	3.4.6 Addition of Signed Magnitude Numbers	3-9	3.12.3	Complement of a Function	3-27
3.5	2's Complement Arithmetic	3-10		• Review Questions	3-28
	3.5.1 Subtraction of Unsigned Binary using 2's Complement	3-10	Unit 1		
	3.5.2 Subtraction of Signed Binary Numbers	3-13			
3.6	Floating Point Representation	3-13			
	3.6.1 Parts of Floating Point Numbers	3-13			
	3.6.2 Binary Floating Point Numbers	3-14			
	3.6.3 Single Precision Floating Point Binary Numbers	3-14			
3.7	IEEE-754 Standard for Representing Floating Point Numbers	3-16	4.1	System or Circuit	4-2
3.8	Introduction to Boolean Algebra	3-19	4.1.1	Digital Systems	4-2
	3.8.1 Basic Logical Operations (Logic Variables)	3-19	4.1.2	Types of Digital Systems	4-2
	3.8.2 NOT Operator (Inversion)	3-19	4.1.3	Combinational Circuit Design	4-3
			4.2	Standard Representations for Logical Functions	4-3
			4.2.1	Sum-of-Products (SOP) Form	4-4
			4.2.2	Product of the Sums Form (POS)	4-4
			4.2.3	Standard or Canonical SOP and POS Forms	4-4

Chapter 4 : Logic Minimization**4-1 to 4-28**

Syllabus : Representation of logic function : Logic statement, Truth-table, SOP form, POS form; Simplification of logical functions using K-Maps upto 4 variables.

4.1	System or Circuit	4-2
4.1.1	Digital Systems	4-2
4.1.2	Types of Digital Systems	4-2
4.1.3	Combinational Circuit Design	4-3
4.2	Standard Representations for Logical Functions	4-3
4.2.1	Sum-of-Products (SOP) Form	4-4
4.2.2	Product of the Sums Form (POS)	4-4
4.2.3	Standard or Canonical SOP and POS Forms	4-4



4.2.4 Conversion of a Logic Expression to Standard SOP or POS Form4-5	4.7.2 Minimization of Logic Functions not Specified in Standard SOP Form4-20
4.3 Concepts of Minterm and Maxterm4-6	4.7.3 Don't Care Conditions4-22
4.3.1 Representation of Logical Expressions using Minterms and Maxterms4-7	4.8 Product of Sum (POS) Simplification4-23
4.3.2 Writing SOP and POS Forms for a Given Truth Table4-7	4.8.1 K-map Representation of POS Form ... 4-23
4.3.3 To Write Standard SOP Expression for a Given Truth Table4-7	4.8.2 Representation of Standard POS form on K-map4-24
4.3.4 To Write a Standard POS Expression for a Given Truth Table4-8	4.8.3 Simplification of Standard POS Form using K-map4-25
4.3.5 Conversion from SOP to POS and Vice Versa4-8	• Review Questions4-28
4.4 Methods to Simplify the Boolean Functions4-9	Unit 2
4.4.1 Algebraic Simplification4-9	
4.5 Karnaugh-Map Simplification (The Map Method)4-10	Chapter 5 : Combinational Logic Design 5-1 to 5-58
4.5.1 K-map Structure4-10	Syllabus : Design using SSI chips : Code converters, Half-adder, Full adder, Half subtractor, Full subtractor, n bit binary adder.
4.5.2 K-map Boxes and Associated Product Terms4-11	Introduction to MSI chips : Multiplexer (IC 74153), Demultiplexer (IC 74138), Decoder (74238), Encoder (IC 74147), Binary adder (IC 7483).
4.5.3 Alternative Way to Label the K-map4-12	Design using MSI chips : BCD adder & subtractor using IC 7483, Implementation of logic functions using IC 74153 & 74138.
4.5.4 Truth Table to K-map4-12	Case study : Use of combinational logic design in 7 segment display interface.
4.5.5 Representation of Standard SOP Form on K-map4-13	
4.6 Simplification of Boolean Expressions using K-map4-13	5.1 Introduction to Combinational Circuits5-2
4.6.1 How does Simplification Takes Place ? 4-14	5.1.1 Analysis of a Combinational Circuit5-2
4.6.2 Way of Grouping (Pairs, Quads and Octets)4-14	5.1.2 Design of Combinational Logic using SSI Chips5-3
4.6.3 Grouping Two Adjacent One's (Pairs) ..4-14	5.2 Design of Combinational Logic using SSI Chips5-4
4.6.4 Grouping Four Adjacent Ones (Quad) ..4-15	5.2.1 Code Converters5-4
4.6.5 Grouping Eight Adjacent Ones (Octet)4-17	5.2.1.1 BCD to Excess 3 Converter5-4
4.6.6 Summary of Rules Followed for K-Map Simplification4-17	5.2.2 BCD to Gray Code Converter5-6
4.7 Minimization of SOP Expressions (K Map Simplification)4-17	5.2.3 Binary to Gray Code Converter5-7
4.7.1 Elimination of a Redundant Group4-20	5.2.4 Gray to BCD Converter5-8
	5.2.5 Excess 3 to BCD Converter5-10
	5.3 Binary Adders and Subtractors5-10
	5.3.1 Types of Binary Adders5-11



5.3.2	Half Adder	5-11	5.8.1	A 2-Bit Comparator	5-27
5.3.3	Full Adder	5-12	5.9	Multiplexer (Data Selector)	5-29
5.3.4	Full Adder using Half Adder	5-13	5.9.1	Necessity of Multiplexers	5-29
5.3.5	Applications of Full Adder	5-13	5.9.2	Advantages of Multiplexers	5-29
5.3.6	Binary Subtractors	5-13	5.10	Types of Multiplexers	5-30
5.3.7	Half Subtractor	5-13	5.10.1	2 : 1 Multiplexer	5-30
5.3.8	Full Subtractor	5-14	5.10.2	A 4 : 1 Multiplexer	5-30
5.3.9	Full Subtractor using Half Subtractors	5-15	5.10.3	8 : 1 Multiplexer	5-31
5.4	The n-Bit Parallel Adder	5-16	5.10.4	Applications of a Multiplexer	5-31
5.4.1	A Four Bit Parallel Adder Using Full Adders	5-16	5.11	Study of Different Multiplexer ICs	5-32
5.4.2	Propagation Delay in Parallel Adder	5-16	5.11.1	54LS 153/DM 54LS 153/DM 74LS 153 (Dual 4 : 1 Multiplexer)	5-32
5.4.3	Look Ahead – Carry Adder	5-17	5.12	Multiplexer Tree/Cascading of Multiplexer	5-33
5.4.4	Four Bit Fast Adder with Look-Ahead Carry	5-18	5.13	Use of Multiplexers in Combinational Logic Design	5-35
5.4.5	MSI Binary Adder IC 74 LS 83 / 74 LS 283	5-19	5.13.1	Implementation of a Logical Expression in the Standard SOP Form	5-35
5.4.6	Four Bit Binary Adder using IC 7483	5-19	5.13.2	Use of 4 : 1 MUX to Realize a 4 Variable Function	5-36
5.4.7	Cascading of Adders	5-19	5.13.3	Use of 8 : 1 MUX to Realize a 4 Variable Function	5-37
5.5	n-bit Parallel Subtractor	5-20	5.13.4	Implementation of a Logical Expression in the Non-standard SOP Form	5-38
5.5.1	4 Bit Parallel Subtractor using IC7483	5-20	5.13.5	Implementing a Standard POS Expression using Multiplexer	5-38
5.5.2	4-Bit Binary Parallel Adder / Subtractor Using IC 7483	5-20	5.13.6	Implementation of Boolean SOP Expression with Don't Care Conditions	5-39
5.6	BCD Addition	5-21	5.14	Demultiplexers	5-41
5.6.1	BCD Adder using MSI IC 7483	5-22	5.14.1	Demultiplexer Principle	5-41
5.7	BCD Subtractor using MSI IC 7483	5-24	5.15	Types of Demultiplexers	5-42
5.7.1	BCD Subtraction using 9's Complement	5-24	5.15.1	1 : 2 Demultiplexer	5-42
5.7.2	4-Bit BCD Subtractor using 9's Complement Method	5-24	5.15.2	1 : 4 Demultiplexer	5-42
5.7.3	BCD Subtraction using 10's Complement	5-25	5.15.3	1 : 8 Demultiplexer	5-43
5.7.4	4-bit BCD Subtraction using 10's Complement Method	5-26	5.15.4	IC 74138 as 1 : 8 DE-MUX	5-43
5.8	Magnitude Comparators	5-27	5.16	Demultiplexer Tree	5-44
			5.16.1	Use of DEMUX in Combinational Logic Design	5-44



5.17	Encoders	5-47
	5.17.1 Types of Encoders	5-47
5.18	Priority Encoder	5-47
	5.18.1 Priority Encoders in the IC Form	5-48
	5.18.2 Decimal to BCD Encoder	5-48
	5.18.3 Decimal to BCD Encoder MSI IC 74147	5-48
5.19	Decoder	5-49
	5.19.1 2 to 4 Line Decoder	5-49
	5.19.2 Demultiplexer as Decoder	5-49
	5.19.3 3 to 8 Line Decoder	5-49
	5.19.4 1:8 DEMUX as 38 Decoder	5-50
	5.19.5 IC 74138 / IC 74238 as 3:8 Decoder	5-50
	5.19.6 Combinational Logic Design Using Decoders	5-51
	5.19.7 Advantage of Decoder Realization	5-54
5.20	Case Study Combinational Logic Design of BCD to 7 Segment Display Controller	5-54
	5.20.1 Seven Segment LED Display	5-54
	5.20.2 Types of Seven Segment Displays	5-54
	5.20.3 Common Anode Display	5-54
	5.20.4 Common Cathode Display	5-54
	5.20.5 Use of a Decoder for Driving the Seven Segment Display	5-55
	5.20.6 BCD to Seven Segment Display Driver (Common Anode Display)	5-55

Unit 3

Chapter 6 : Flip Flops

6-1 to 6-38

Syllabus : Introduction to sequential circuits : Difference between combinational circuits and sequential circuits; Memory element-latch & Flip-Flop.

Flip-Flops : Logic diagram, Truth table & excitation table of SR, JK, D, T flip flops; Conversion from one FF to another, Study of flip flops with regard to asynchronous and synchronous, Preset & clear, Master slave configuration; Study of 7474, 7476 flip flop ICs.

Case study : Use of sequential logic design in a simple traffic light controller.

6.1	Introduction	6-2
6.1.1	Clock Signal	6-2
6.1.2	Comparison of Combinational and Sequential Circuits	6-2
6.1.3	1-Bit Memory Cell (Basic Bistable Element)	6-3
6.1.4	Latch	6-3
6.1.5	Symbol and Truth Table of S-R Latch	6-4
6.1.6	Characteristic Equation	6-4
6.1.7	NAND Latch [S-R Latch using NAND Gates]	6-5
6.2	Triggering Methods	6-6
6.2.1	Concept of Level Triggering	6-6
6.2.2	Types of Level Triggered Flip-flops	6-6
6.2.3	Concept of Edge Triggering	6-7
6.2.4	Types of Edge Triggered Flip Flops	6-7
6.3	Gated Latches (Level Triggered SR Flip Flop)	6-7
6.3.1	Types of Level Triggered (Clocked) Flip Flops	6-7
6.4	The Gated S-R Latch (Level Triggered S-R Flip Flop)	6-7
6.4.1	Positive Level Triggered SR Flip-flop	6-7
6.4.2	Negative Level Triggered SR Flip Flop ..	6-8
6.5	The Gated D Latch (Clocked D Flip Flop)	6-9
6.6	Gated JK Latch (Level Triggered JK Flip Flop)	6-10
6.6.1	Race Around Condition in JK Latch	6-10
6.6.2	Difference between Latch and Flip-flop	6-11
6.7	Edge Triggered Flip Flops	6-11
6.7.1	Positive Edge Triggered S-R Flip Flop	6-11
6.7.2	Negative Edge Triggered S - R Flip Flop	6-13
6.8	Edge Triggered D Flip Flop	6-13
6.8.1	Positive Edge Triggered D Flip Flop	6-13
6.8.2	Negative Edge Triggered D Flip Flop ...	6-14



6.8.3 Applications of D Flipflop	6-14
6.9 Edge Triggered J-K Flip Flop	6-14
6.9.1 Positive Edge Triggered JK Flip Flop ...	6-14
6.9.2 Characteristic Equation of JK Flip Flop	6-16
6.9.3 How does an Edge Triggered JK FF Avoid Race Around Condition ?	6-16
6.9.4 Negative Edge Triggered JK Flip-Flop	6-16
6.10 Toggle Flip Flop (T Flip Flop)	6-17
6.10.1 Positive Edge Triggered T-FF	6-17
6.10.2 Negative Edge Triggered T Flip Flop	6-18
6.10.3 Application of T F/F	6-18
6.11 Master Slave (MS) JK Flip Flop	6-18
6.12 Preset and Clear Inputs	6-20
6.12.1 S-R Flip-Flop with Preset and Clear Inputs	6-20
6.12.2 Synchronous Preset and Clear Inputs	6-21
6.12.3 JK Flip Flop with Preset and Clear Inputs	6-21
6.12.4 Applications of JK Flip Flop	6-21
6.13 Various Representations of Flip Flops	6-21
6.13.1 Characteristic Equations	6-22
6.14 Excitation Table of Flip-Flop	6-22
6.14.1 Excitation Table of SR Flip Flops	6-22
6.14.2 Excitation Table of D Flip Flop	6-23
6.14.3 Excitation Table of JK Flip Flop	6-23
6.14.4 Excitation Table of T Flip Flop	6-23
6.15 Conversion of Flip Flops	6-23
6.15.1 Conversion from S-R Flip Flop to D Flip Flop	6-24
6.15.2 Conversion of JK FF to T FF	6-24
6.15.3 SR Flip Flop to T Flip Flop	6-25
6.15.4 SR Flip Flop to JK Flip Flop	6-25
6.15.5 Conversion of D Flip Flop to T Flip Flop	6-26
6.15.6 T Flip Flop to D Flip Flop Conversion	6-26
6.15.7 JK Flip Flop to D Flip Flop Conversion	6-27
6.15.8 JK Flip Flop to SR Flip Flop Conversion	6-27
6.15.9 D FF to SR FF Conversion	6-28
6.15.10 T FF to SR FF Conversion	6-28
6.15.11 Conversion from D FF to JK FF	6-28
6.16 Applications of Flip Flops	6-29
6.17 Study of Flip-Flop ICs	6-29
6.17.1 SN74LS74A Dual D-Type Positive Edge- triggered Flip-Flop Low Power Schottky	6-29
6.17.2 SN74LS76A Dual JK Flip-Flop with Set and Clear Low Power Schottky	6-30
6.18 Analysis of Clocked Sequential Circuits	6-31
6.18.1 State Table	6-31
6.18.2 State Diagram	6-31
6.18.3 State Equation	6-32
6.19 Design of Clocked Synchronous State Machine using State Diagram	6-34
6.20 Case Study Use of Sequential Logic Design in a Simple Traffic Light Controller	6-35
• Review Questions	6-37

Unit 3**Chapter 7 : Counters**

7-1 to 7-34

Syllabus : Application of flip-flops : Counters - Asynchronous, Synchronous and modulo n counters, Study of 7490 modulus n counter ICs & their applications to implement mod counters.

7.1 Introduction	7-2
7.1.1 Types of Counters	7-2
7.1.2 Classification of Counters	7-2
7.2 Asynchronous / Ripple Up Counters	7-2



7.2.1	Two Bit Asynchronous Up Counter using JK Flip-Flops	7-4	7.10.3	Comparison of Synchronous and Asynchronous Counters	7-29
7.2.2	3 Bit Asynchronous Up Counter	7-4	7.11	Lock Out Condition	7-29
7.2.3	4 Bit Asynchronous up Counter	7-5	7.11.1	Bushless Circuit	7-29
7.2.4	State Diagram of a Counter	7-6	7.12	Bush Diagram	7-31
7.3	Asynchronous Down Counters	7-6	7.13	Applications of Counters	7-33
7.3.1	3- Bit Asynchronous Down Counter	7-6		• Review Questions	7-33
7.4	UP / DOWN Counters	7-7			
7.4.1	UP/DOWN Ripple Counters	7-7			
7.5	Modulus of the Counter (MOD-N Counter)	7-8			
7.5.1	Design of Asynchronous MOD Counters	7-8			
7.5.2	Frequency Division Taking Place in Asynchronous Counters	7-10			
7.5.3	Disadvantages of Ripple Counters	7-10			
7.6	Ripple Counter IC 7490 (Decade Counter)	7-10			
7.6.1	The Internal Diagram of IC 7490	7-11	8.1	Introduction	8-2
7.6.2	Other Applications of IC 7490	7-12	8.2	Data Formats	8-2
7.6.3	Symmetrical Bi-quinary Divide by Ten Counter	7-12	8.3	Classification of Registers	8-2
7.7	Problems Faced by Ripple Counters	7-19	8.4	Buffer Registers	8-2
7.8	Synchronous Counters	7-19	8.5	Shift Registers	8-3
7.8.1	2-Bit Synchronous up Counter	7-19	8.5.1	Serial Input Serial Output (Shift Left Mode)	8-4
7.8.2	3-Bit Synchronous Binary up Counter	7-20	8.5.2	Serial In Serial Out (Shift Right Mode)	8-5
7.8.3	Design of the 3 Bit Synchronous Counter	7-21	8.5.3	Applications of Serial Operation	8-6
7.8.4	Four Bit Synchronous Up Counter	7-23	8.6	Serial In Parallel Out (SIPO)	8-7
7.9	Modulo – N Synchronous Counters	7-24	8.7	Parallel In Serial Out Mode (PISO)	8-7
7.9.1	Synchronous Decade Counter	7-24	8.8	Parallel In Parallel Out (PIPO)	8-8
7.10	UP / DOWN Synchronous Counter	7-27	8.9	Bidirectional Shift Register	8-9
7.10.1	3-bit Up/Down Synchronous Counter	7-28	8.9.1	A 3-bit Bidirectional Register using the JK Flip Flops	8-9
7.10.2	Advantages of Synchronous Counter ...	7-28	8.10	Applications of Shift Registers	8-10
			8.10.1	Serial to Parallel Converter	8-10
			8.10.2	Parallel to Serial Converter	8-10
			8.10.3	Ring Counter	8-10
			8.10.4	Johnson's Counter (Twisted / Switch Tail Ring Counter)	8-12
			8.10.5	Sequence Generator	8-16
			8.11	Sequence Detector	8-16



8.11.1 Pseudo Random Binary Sequence (PRBS) Generator	8-16
• Review Questions	8-16

Unit 4**Chapter 9 : Computer Organization & Processor****9-1 to 9-18**

Syllabus : Computer organization and computer architecture, Organization, Functions and types of computer units - CPU (Typical organization, Functions, Types), Memory (Types and their uses in computer), IO (Types and functions) and system bus (Address, data and control, Typical control lines, Multiple-Bus Hierarchies); Von Neumann and Harvard architecture; Instruction cycle.

Processor : Single bus organization of CPU; ALU (ALU signals, functions and types); Register (Types and functions of user visible, Control and status registers such as general purpose, Address registers, Data registers, Flags, PC, MAR, MBR, IR) & control unit (Control signals and typical organization of hard wired and microprogrammed CU), Micro Operations (Fetch, Indirect, Execute, Interrupt) and control signals for these micro operations.

Case study : 8086 processor, PCI bus.

9.1 Introduction	9-2
9.2 Basic Organization of Computer and Block Level Description of Functional Units	9-2
9.2.1 Structural Components of a Computer ...	9-2
9.2.2 Functional View of a Computer	9-3
9.3 Von Neumann and Harvard Architecture	9-3
9.3.1 Von Neumann Architecture	9-3
9.3.2 Harvard Architecture	9-6
9.4 Basic Instruction Cycle	9-6
9.4.1 Interrupt Cycle	9-6
9.5 CPU Architecture and Register Organization	9-8
9.5.1 Interrupt Control	9-9
9.5.2 Timing and Control Unit	9-9
9.6 Instruction, Micro-instructions and Micro-operations Interpretation and Sequencing	9-9
9.6.1 Fetch Cycle	9-10

9.6.2 Execute Cycle	9-10
9.6.3 Interrupt Cycle	9-11
9.6.4 Examples of Microprograms	9-12
9.6.5 Applications of Microprogramming	9-14

9.7 Control Unit Hardwired Control Unit Design Methods	9-15
--	------

9.8 Control Unit Soft Wired (Micro programmed) Control Unit Design Methods	9-16
9.8.1 Wilkie's Microprogrammed Control Unit	9-17
9.8.2 Comparison between Hardwired and Micro-programmed Control	9-18

• Review Questions	9-18
---------------------------------	-------------

Unit 5**Chapter 10 : Processor Instructions & Processor Enhancements****10-1 to 10-40**

Syllabus : Instruction : Elements of machine instruction; Instruction representation (Opcode and mnemonics, Assembly language elements) ; Instruction format and 0-1-2-3 address formats, Types of operands,

Addressing modes; Instruction types based on operations (Functions and examples of each); Key characteristics of RISC and CISC; Interrupt : Its purpose, Types, Classes and interrupt handling (ISR, Multiple interrupts), Exceptions; Instruction pipelining (Operation and speed up),

Multiprocessor systems : Taxonomy of Parallel Processor Architectures, Two types of MIMD clusters and SMP (Organization and benefits) and multicore processor (Various alternatives and advantages of multicores), Typical features of multicore intel core i7.

Case study : 8086 Assembly language programming.

10.1 Instruction Encoding Format	10-2
10.2 Instruction Format and 0-1-2-3 Address Formats	10-5
10.2.1 Instruction Formats	10-5
10.2.2 Instruction Word Format - Number of Addresses	10-5
10.3 Addressing Modes	10-6



10.3.1 Examples on Addressing Modes	10-8	10.9.2.4 Branch Prediction	10-31
10.4 Instruction Set of 8085	10-9	10.9.2.5 Pipeline Stall (Delayed Branch)	10-31
10.5 Reduced Instruction Set Computer Principles	10-11	10.9.2.6 Loop Unrolling Technique	10-31
10.5.1 RISC Versus CISC	10-11	10.9.2.7 Software Scheduling or Software Pipelining	10-31
10.5.2 RISC Properties	10-11	10.9.2.8 Trace Scheduling	10-32
10.5.3 Register Window	10-12	10.9.2.9 Predicated Execution	10-33
10.5.4 Miscellaneous Features or Advantages of RISC Systems	10-12	10.9.2.10 Speculative Loading	10-33
10.5.5 RISC Shortcomings	10-14	10.9.2.11 Register Tagging	10-33
10.5.6 ON-Chip Register File Versus Cache Evaluation	10-14	10.9.3 Branch Prediction	10-34
10.6 Polling and Interrupts	10-14	10.9.3.1 Misprediction Penalty	10-34
10.7 Pipeline Processing	10-15	10.9.3.2 Static Branch Prediction	10-34
10.7.1 Non-Pipelined System Versus Two Stage Pipelining	10-15	10.9.3.3 Branch-Target Buffer or Branch-Target Address Cache	10-34
10.7.2 Basic Pipelined Datapath and Control for a Six Stage CPU Instruction Pipeline	10-16	10.9.3.4 Dynamic Branch Prediction	10-34
10.7.3 Linear Pipeline Processors	10-17	10.9.3.5 One-bit Dynamic Branch Predictor	10-35
10.7.3.1 Asynchronous and Synchronous Linear Pipelining	10-17	10.9.3.6 Two-bit Prediction	10-35
10.7.3.2 Clocking and Timing Control	10-18	10.10 Multiprocessor Systems and Multicore Processor (Intel Core i7 Processor)	10-35
10.7.3.3 Speedup, Efficiency and Throughput .	10-19	10.10.1 Overlapping the CPU and Memory or I/O Operations	10-35
10.7.4 Non Linear Pipeline Processors	10-20	10.11 Flynn's Classifications	10-35
10.7.4.1 Collision Free Scheduling or Job Sequencing	10-22	10.11.1 Flynn's Classification of Parallel Computing	10-35
10.8 Instruction Pipelining and Pipelining Stages	10-27	10.11.2 i5/i7 Mobile Version	10-38
10.9 Pipeline Hazards	10-28	• Review Questions	10-40
10.9.1 Methods to Resolve the Data Hazards and Advances in Pipelining	10-29		
10.9.1.1 Pipeline Stalls	10-29	Unit 6	
10.9.1.2 Operand Forwarding (or) Bypassing ..	10-29		
10.9.1.3 Dynamic Instruction Scheduling (or) Out-Of-Order (OOO) Execution	10-30		
10.9.2 Handling of Branch Instructions to Resolve Control Hazards	10-30		
10.9.2.1 Pre-Fetch Target Instruction	10-30		
10.9.2.2 Branch Target Buffer (BTB)	10-30		
10.9.2.3 Loop Buffer	10-30		

Chapter 11 : Memory & Input / Output Systems**11-1 to 11-38**

Syllabus : Memory Systems : Characteristics of memory systems, Memory hierarchy, Signals to connect memory to processor, Memory read and write cycle, Characteristics of semiconductor memory : SRAM, DRAM and ROM, Cache memory – Principle of locality, Organization, Mapping functions, Write policies, Replacement policies, Multilevel caches, Cache coherence,

Input / Output systems : I/O module, Programmed I/O, Interrupt driven I/O, Direct Memory Access (DMA).

Case study : USB flash drive.



11.1	Introduction to Memory and Memory Parameters	11-2
11.1.1	Bytes and Bits	11-3
11.2	Memory Hierarchy Classifications of Primary and Secondary Memories	11-3
11.3	Types of RAM	11-4
11.4	ROM (Read Only Memory)	11-5
11.4.1	Types of ROM	11-5
11.4.2	Magnetic Memory	11-6
11.4.3	Optical Memory	11-7
11.5	Allocation Policies	11-9
11.6	Signals to Connect Memory To Processor and Internal Organization of Memory	11-10
11.7	Memory Chip Size and Numbers	11-11
11.8	Cache Memory Concept, Architecture (L1, L2, L3) and Cache Consistency	11-18
11.8.1	Cache Operation	11-18
11.8.2	Principles of Locality of Reference	11-18
11.8.3	Cache Performance	11-19
11.8.4	Cache Architectures	11-19
11.9	Cache Mapping Techniques	11-20
11.9.1	Direct Mapping Technique	11-20
11.9.2	Fully Associative Mapping	11-21
11.9.3	Set Associative Mapping	11-22
11.9.4	Write Policy	11-24
11.9.5	Replacement Algorithms	11-25
11.9.6	Cost and Performance Measurement of Two Level Memory Hierarchy	11-28
11.9.7	Cache Consistency (Also Known as Cache Coherency)	11-28
11.9.8	Bus Master / Cache Interaction for Cache Coherency	11-28
11.10	Pentium Processor Cache Unit	11-29
11.10.1	Memory Reads Initiated by the Pentium Processor	11-29
11.10.2	Memory Writes Initiated by PENTIUM Processor	11-30
11.10.3	Memory Reads Initiated by Another Bus Master	11-31
11.10.4	The MESI Model	11-31
11.11	Input / Output System	11-32
11.11.1	Parallel Versus Serial Interface	11-32
11.11.2	Types of Communication Systems	11-33
11.12	I/O Modules and 8089 IO Processor	11-33
11.12.1	I/O Module	11-34
11.13	Types of Data Transfer Techniques Programmed I/O, Interrupt Driven I/O and DMA	11-34
11.13.1	Programmed I/O	11-34
11.13.2	Interrupt Driven I/O	11-35
11.13.3	DMA	11-36
11.13.4	DMA Transfer Modes	11-36
•	Review Questions	11-37

□□□

Unit 1

Chapter 1

Digital Logic Families

Syllabus

Digital IC characteristics; TTL : Standard TTL characteristics, Operation of TTL NAND gate; CMOS : Standard CMOS characteristics, Operation of CMOS NAND gate; Comparison of TTL & CMOS.
Case study : CMOS 4000 series ICs.

Chapter Contents	
1.1	Logic Families
1.2	Classification of Logic Families
1.3	Characteristics of Digital ICs
1.4	Transistor-Transistor Logic (TTL)
1.5	Open Collector Outputs (TTL)
1.6	Standard TTL Characteristics
1.7	MOS - Logic Family
1.8	CMOS Logic
1.9	Standard CMOS Characteristics
1.10	Comparison of CMOS and TTL
1.11	Case Study : CMOS 4000 series ICs



1.1 Logic Families :

SPPU : Dec. 08, May 10, Dec. 19

University Questions

Q. 1 What is logic family ?

(Dec. 08, May 10, Dec. 19, 2 Marks)

Definition :

- Logic families are defined as the type of logic circuit used in an IC. Various digital ICs available in market belong to various types. Each type is known as a logic family.
- Various digital ICs available in market belong to various types. These types are known as "families".
- Based on the components and devices internally used, the digital IC families are named as RTL (Resistor Transistor Logic), TTL (Transistor Transistor Logic), DTL (Diode Transistor Logic), CMOS etc.

Features of logic families :

- Table 1.1.1 gives the comparison of some of the outstanding features of these logic families.

Table 1.1.1 : Comparison of important features of logic families

Sr. No.	Characteristics	TTL	CMOS	ECL
1.	Power input	Moderate	Low	Moderate to high
2.	Frequency limit	High	Moderate	Very high
3.	Circuit density	Moderate to high	High to very high	Moderate
4.	Circuit types per family	High	High	Moderate

- Thus TTL family has a great versatility. It has high speed as well (delay less than 1 nS for some TTL subfamilies).
- The CMOS family is popular because of its low input power and very high circuit density i.e. more circuits can be placed in the same volume of IC.
- ECL is used for very high speed digital circuits.
- But it needs more input power and less types of logic circuits are available in ECL than those available in TTL and CMOS.

1.1.1 Classification Based on Circuit Complexity :

- The logical circuits of different types are available in the integrated circuits.
- Depending on the level of complexity the integrated circuit are classified into four categories as follows :
 - SSI : Small Scale Integration.
 - MSI : Medium Scale Integration.
 - LSI : Large Scale Integration.
 - VLSI : Very Large Scale Integration.
- The number of components (diodes, transistors, gates etc.) in SSI will be the lowest and that in VLSI will be the highest.
 - SSI < 10 components.
 - MSI < 100 components.
 - LSI > 100 components.
 - VLSI > 1000 components.

1.2 Classification of Logic Families :

1.2.1 Classification Based on Devices Used :

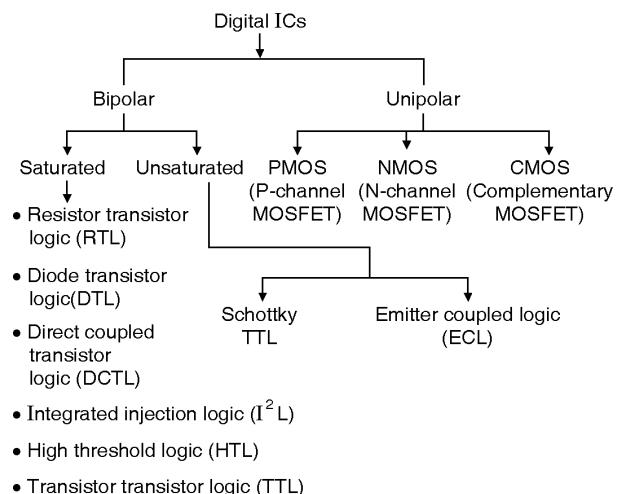
SPPU : Dec. 08, May 10

University Questions

Q. 1 Give the classification of logic families.

(Dec. 08, May 10, 4 Marks)

- The two basic techniques for manufacturing ICs are :
 - Bipolar technology.
 - Unipolar devices-Metal Oxide Semiconductor (MOS) technology.
- The classification of logic families is shown in Fig. 1.2.1.



(C-205) Fig. 1.2.1 : Classification of logic families

**Bipolar families :**

- The bipolar families of logic circuits use the bipolar transistors fabricated on the chip.
- That means all the gates belonging to the bipolar family use the transistorised circuits.
- In the bipolar category there are three basic families called, Diode Transistor Logic (DTL), Transistor Transistor Logic (TTL) and Emitter Coupled Logic (ECL).
- DTL uses diodes and transistors, TTL uses transistors as the main elements.
- TTL has become the most popular family in SSI (Small Scale Integration) and MSI (Medium Scale Integration) chips, while ECL is the fastest logic family which is used for high speed applications.
- In the "Bipolar saturated" logic families, the bipolar transistors are used as the main device.
- It is used as a switch and operated in the saturation or cutoff regions.
- TTL is an example of saturated bipolar logic.
- In the unsaturated bipolar logic, the bipolar transistors are not driven into hard saturation.
- This increases the speed of operation. So the unsaturated bipolar ICs such as Schottky TTL and ECL (Emitter Coupled Logic) are much faster as compared to TTL.
- All these ICs are fabricated on silicon chips using different fabrication technologies.

Unipolar families :

- The MOS family use fabricates MOS Field Effect Transistors (MOSFETs) fabricated on the chip. So all the gates belonging to the MOS family use the MOSFET based circuits.
- In the MOS category there are three logic families namely PMOS (p-channel MOSFETs) family, NMOS (n-channel MOSFETs) family and CMOS (Complementary MOSFETs) family.
- PMOS is the oldest and slowest type. NMOS is used for the LSI (large scale integration) field i.e. for the microprocessors and memories.
- CMOS which uses a push pull arrangement of n-channel and p-channel MOSFETs, is extensively used where low power consumption is needed such as in pocket calculators.

1.3 Characteristics of Digital ICs :

- Eventhough there are various logic families, the general characteristics, their definitions, and terminologies used for all of them have been standardized.
- So let us discuss some of the most important general characteristics that are applicable to all the digital ICs first and then discuss them for some particular families.
- The important characteristics of all the digital IC families are as follows :
 1. Voltage and current parameters.
 2. Fan-in and fan-out.
 3. Noise margin.
 4. Propagation delay (speed).
 5. Power dissipation.
 6. Operating temperature.
 7. Invalid voltage levels.
 8. Figure of merit (SPP).

1.3.1 Voltage and Current Parameters :**SPPU : Dec. 08, May 17****University Questions****Q. 1 Explain any four characteristics of digital ICs.****(Dec. 08, 2 Marks)****Q. 2 Explain any three characteristics of digital ICs.****(May 17, 6 Marks)****Voltage parameters (Threshold levels) :**

- Ideally the input voltage levels of 0 V and + 5 V (for TTL) are called as logic 0 and 1 levels respectively.
- However practically we won't always observe or obtain the voltage levels matching exactly to these values.
- Therefore it is necessary to define the worst case input voltages.

1. $V_{IL(max)}$ - Worst case low level input voltage :

- This is the maximum value of input voltage which is to be considered as a logic 0 level.
- If the input voltage is higher than $V_{IL(max)}$, then it will not be treated as a low (0) input level.

2. $V_{IH(min)}$ - Worst case high level input voltage :

- This is the minimum value of the input voltage which is to be considered as a logic 1 level.
- If the input voltage is lower than $V_{IH(min)}$, then it will not be treated as a High (1) input.

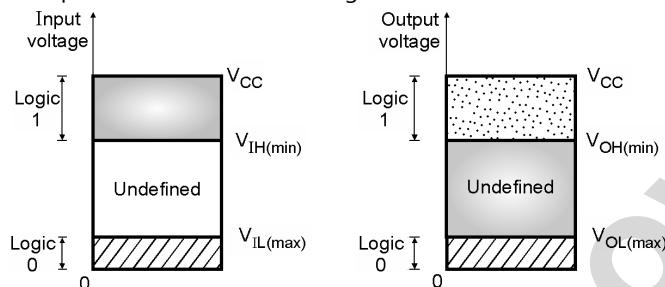


3. $V_{OH(min)}$ - Worst case high level output voltage :

- This is the minimum value of the output voltage which will be considered as a logic HIGH (1) level.
- If the output voltage is lower than this level then it won't be treated as a HIGH (1) output.

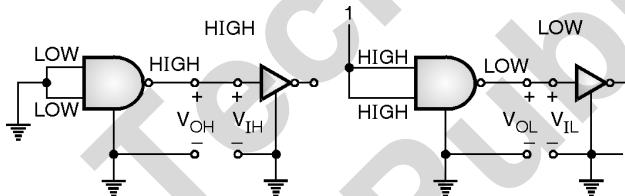
4. $V_{OL(max)}$ - Worst case low level output voltage :

- This is the maximum value of the output voltage which will be considered as a logic LOW (0) level.
- If the output voltage is higher than this value then it won't be treated as a LOW (0) output. All the voltage parameters are shown in Fig. 1.3.1.



(C-7584) Fig. 1.3.1 : Voltage parameters

- The voltage parameters can be shown on the digital circuit consisting of gates as shown in Fig. 1.3.2.
- Note that, the NAND and NOT gates shown, can be of TTL, ECL, CMOS or any other type.



(C-207) Fig. 1.3.2 : Voltage parameters on a logic circuit

Current parameters :

1. I_{IL} - Low level input current :

- It is the current that flows into the input terminals when a low level input voltage in the specified range is applied.

2. I_{IH} - High level input current :

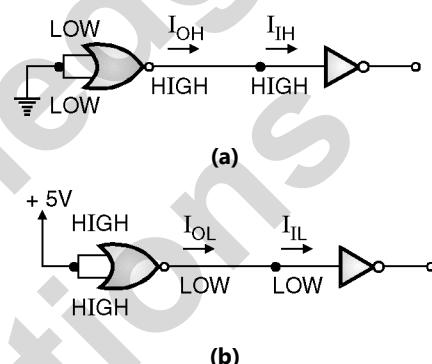
- It is the current that flows into the input terminals when a high level input voltage in the specified range is applied.

3. I_{OL} - Low level output current :

- This is the current that flows out of the output when the output voltage happens to be in the specified low (0) voltage range and a specified load is applied.

4. I_{OH} - High level output current :

- This is the current flowing from the output when the output voltage happens to be in the specified HIGH (1) voltage range and a specified load is applied.
- If the output current flows into the output terminal then it is called as a **sinking current** and if the output current flows out of the output terminal then it is called as a **sourcing current**.
- The current parameters are displayed on the logic circuit shown in Fig. 1.3.3.



(C-208) Fig. 1.3.3 : Current parameters

- Note that the actual current directions can be opposite to those shown in Fig. 1.3.3; depending on the logic family.
- Note that current flowing into a node or device is considered positive and current flowing out of node or device is considered negative.

1.3.2 Fan-in and Fan-out :

SPPU : May 17, May 18, Dec. 19

University Questions

Q. 1 Explain any three characteristics of digital ICs.

(May 17, 6 Marks)

Q. 2 Define following terms related to logic families :

1. Fan-in 2. Fan-out. (May 18, 4 Marks)

Q. 3 What is Logic Family ? Explain the terms :

1. Fan out 2. Propagation Delay.

(Dec. 19, 6 Marks)

Fan-in :

- Fan-in is defined as the number of inputs a gate has. For example a two input gate will have a fan-in equal to 2.

Fan-out :

- Fan-out is defined as the maximum number of inputs of the same IC family that a gate can drive without falling outside the specified output voltage limits.

- Higher fan out indicates higher the current supplying capacity of a gate.
- For example, a fan out of 5 indicates that the gate can drive (supply current to) at the most 5 inputs of the same IC family.

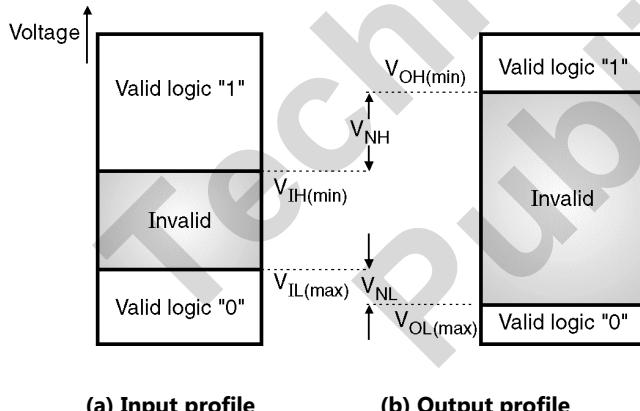
1.3.3 Noise Margin :

SPPU : Dec. 08, Dec. 15, May 17, May 18

University Questions

- Q. 1** Explain noise margin. **(Dec. 08, 2 Marks)**
- Q. 2** Explain the following TTL characteristics :
1. Noise Immunity
 2. High level input voltage (V_{NH})
 3. Figure of Merit **(Dec. 15, 6 Marks)**
- Q. 3** Explain any three characteristics of digital ICs. **(May 17, 6 Marks)**
- Q. 4** Define following terms related to logic families :
Noise margin. **(May 18, 2 Marks)**

- To understand the meaning of the term "Noise Margin" or "Noise Immunity", refer to the input and output voltage profiles shown in Fig. 1.3.4.



(C-210) Fig. 1.3.4

- Noise is an unwanted electrical disturbance which may induce some voltage in the connecting wires used between two gates or from a gate output to load.

Definition :

- **Noise immunity** is defined as the ability of a logic circuit to tolerate the noise without causing the output to change undesirably.
- A quantitative measure of noise immunity of a logic family is known as **noise margin**.

- In order to avoid the effects of noise voltage, the designers adjust the voltage levels $V_{OH}(\min)$ and $V_{IH}(\min)$ to different levels with some difference between them as shown in Fig. 1.3.4.
- The difference between $V_{OH}(\min)$ and $V_{IH}(\min)$ is known as the **high level noise margin V_{NH}** .
- Similarly the difference between $V_{IL}(\max)$ and $V_{OL}(\max)$ is called as the **low level noise margin V_{NL}** .
- High level noise margin,
$$V_{NH} = V_{OH}(\min) - V_{IH}(\min)$$
- Low level noise margin,
$$V_{NL} = V_{IL}(\max) - V_{OL}(\max)$$
- When a high logic output is driving a logic circuit input, any negative noise spike greater than V_{NH} can force the voltage to reduce into the invalid range.
- Similarly when a low logic output is driving a logic circuit input, any positive noise spikes greater than V_{NL} can force the voltage to go into the invalid state.

1.3.4 Propagation Delay (Speed of Operation) :

SPPU : Dec. 08, May 18, Dec. 19

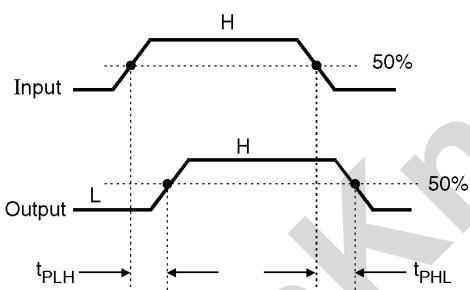
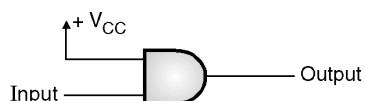
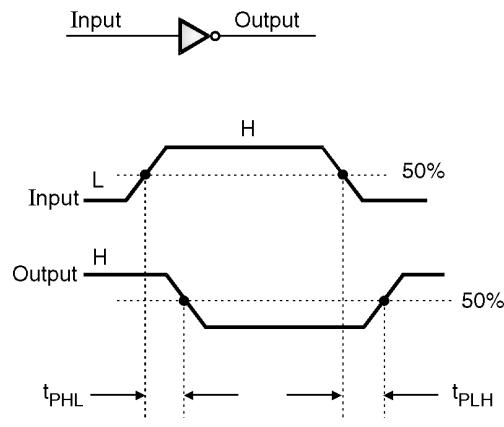
University Questions

- Q. 1** Explain propagation delay. **(Dec. 08, 2 Marks)**
- Q. 2** Define following terms related to logic families : Propagation delay. **(May 18, 2 Marks)**
- Q. 3** What is Logic Family ? Explain the terms :
1. Fan out 2. Propagation Delay.

(Dec. 19, 6 Marks)

Definition :

- The output of a logic gate does not change its state instantaneously when the state of its input is changed.
- There is a time delay between these two time instants, which is called as the propagation delay.
- Thus **propagation delay** is defined as time delay between the instant of application of an input pulse and the instant of occurrence of the corresponding output pulse. This is shown in Fig. 1.3.5.



(b) Propagation delays for an AND gate
(C-211) Fig. 1.3.5

- From Fig. 1.3.5 it is observed that there are two propagation delays.
- 1. **t_{PLH}** : It is the propagation delay measured when the output is making a transition from HIGH (1) to LOW (0) state.
- 2. **t_{PHL}** : This is the propagation delay measured when the output makes a transition from LOW (0) to HIGH (1) state.
- The values of t_{PLH} and t_{PHL} are not always equal. If they are not equal then the one which is higher is considered as the propagation delay time of the gate.
- The propagation delays are measured between the points corresponding to 50% levels as shown in Fig. 1.3.5.
- Ideally propagation delay should be zero and practically it should be as short as possible.
- The values of propagation delays are used to measure how fast a logic circuits is.

- For example, a logic circuit with a propagation time of 5 nS will be a faster logic circuit than the one with 10 nS propagation time, under the specified load conditions.

1.3.5 Power Dissipation :

SPPU : Dec. 08, May 18

University Questions

- Q. 1** Explain the power dissipation. (Dec. 08, 2 Marks)
Q. 2 Define following term related to logic families : Power dissipation. (May 18, 2 Marks)

Definition :

- The power dissipated in a logic IC due to applied voltage and currents flowing through it, is known as its power dissipation.
 - This power is dissipated in it, in the form of heat.
 - The power drawn by an IC from the power supply is given by,
- $$P = V_{CC} \times I_{CC}$$
- where I_{CC} is the current drawn from the power supply.
- This power is in milliwatts. Care should be taken to reduce the power dissipation taking place in the logic IC in order to protect the IC against damage due to excessive temperature, to reduce the loading on power supplies etc.
 - Another importance of power dissipation is that the product of power dissipation and propagation time is always constant.
 - Therefore if we reduce the power dissipation may lead then the propagation delay will increase in order to keep their product constant.
 - Usually there is only one power supply terminal on any IC. It is denoted by V_{CC} for the TTL ICs and V_{DD} for the CMOS ICs.

1.3.6 Operating Temperature :

- The temperature range acceptable for the consumer and industrial applications is 0° to 70° C and that for the military applications is – 55° C to 125° C.
- The performance of gates will be in the specified limits only if the temperature is in these ranges.



1.3.7 Figure of Merit (Speed Power Product (SPP)) :

SPPU : May 18

University Questions

- Q. 1** Define following terms related to logic families :
Figure of merit. **(May 18, 2 Marks)**

Definition :

- The figure of merit of a logical family is the product of power dissipation and propagation delay.
- It is called as the speed power product. The speed is specified in seconds and power is specified in W.
- ∴ Figure of merit = Propagation delay time × Power dissipation.
- Ideally the value of figure of merit is zero and practically, it should be as low as possible.
- Figure of merit is always a compromise between speed and power dissipation.
- That means if we try to reduce the propagation delay then the power dissipation will increase and vice-versa.

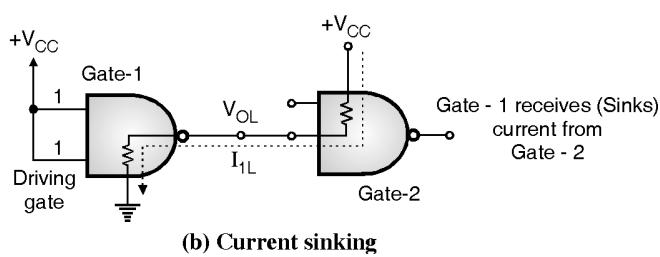
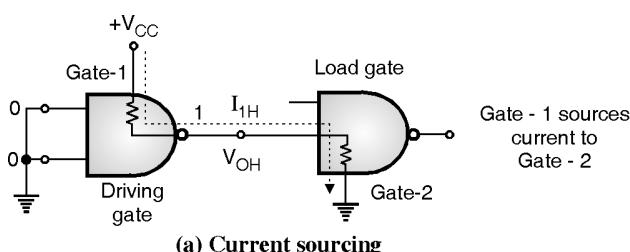
1.3.8 Invalid Voltage Levels :

- The operation of a logic circuit will be proper if and only if its input voltage levels are kept outside the invalid voltage range.
- That means the input voltage should be either lower than $V_{IL(\max)}$ or higher than $V_{IH(\min)}$.
- The invalid input voltage will produce an unpredictable output response. Therefore it should be avoided.
- When the output is overloaded, there is a possibility of output voltage going into the invalid range.

1.3.9 Current Sourcing and Current Sinking :

Current sourcing :

- The current sourcing action is illustrated in Fig. 1.3.6(a). Gate-1 acts as a source and Gate-2 acts as load.
- The output of Gate-1 is high. It supplies a current I_{IH} to the input of Gate-2. This is called as the current sourcing action.



Current sinking :

- The current sinking action has been demonstrated in Fig. 1.3.6(b).
- Gate-2 which is the load gate acts as a load.
- As soon as the output of Gate-1 goes low, the current starts flowing into the output terminal of Gate-1 as shown in Fig. 1.3.6(b).
- Thus when gate-1 is accepting the current through its output terminal, it is said that the current sinking is taking place.

1.3.10 Power Supply Requirements :

- The supply voltage(s) and the amount of power required by an IC are important characteristics required to choose the proper power supply.

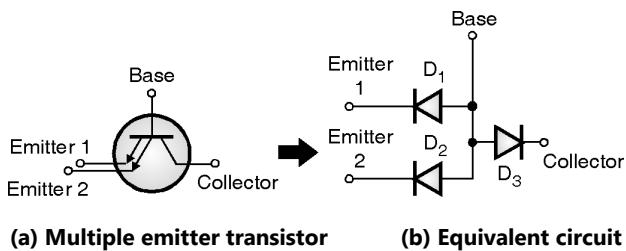
1.4 Transistor-Transistor Logic (TTL) :

- The long form of TTL is transistor logic. The digital ICs in the TTL family use only transistors as their basic building block.
- TTL ICs were first developed in 1965 and they were known as "standard TTL".
- This version of TTL circuits is not practically used now due to availability of advanced versions.
- The standard TTL has been improved to a great extent over the years.
- TTL devices are still used as "glue" logic which connects more complex devices in digital systems.
- The bipolar TTL family as a whole is now becoming obsolete, but we can study it in order to understand basic important concepts in general about the logic families.

1.4.1 The Multiple Emitter Transistor :

- In the TTL circuits that we are going to discuss, use a very special type of transistor.

- The normal transistor has only three terminals namely collector, base and emitter.
- But this special transistor has more than one emitters, as shown in Fig. 1.4.1(a) and its equivalent circuit is shown in Fig. 1.4.1(b). The number of emitters is equal to the number of inputs of the gate.



(C-1011) Fig. 1.4.1

- The multiple emitter input transistor can have upto eight emitters, for an eight input NAND gate.
- In the equivalent circuit of Fig. 1.4.1(b), diodes D_1 and D_2 represent the two base to emitter junctions whereas D_3 represents the collector to base junction.
- In the forthcoming discussions, we are going to replace the multiple emitter transistor by its equivalent circuit shown in Fig. 1.4.1(b).
- This transistor can be turned ON by forward biasing either (or both) the diodes D_1 and D_2 .
- This transistor will be in the OFF state if and only if both the base-emitter junctions (D_1 and D_2) are reverse biased.

Standard TTL :

- The 7400 TTL series is known as the standard TTL series. The TTL gates that we are going to discuss belong to this family.

1.4.2 Two Input TTL-NAND Gate (Totempole Output) :

SPPU : Dec. 08, Dec. 09, May 10, May 12,
Dec. 12, Dec. 13

University Questions

- Q. 1** Draw three input standard TTL NAND gate circuit and explain its operation. **(Dec. 08, 8 Marks)**
- Q. 2** Draw 2-input standard TTL NAND gate circuit and explain operation of transistor (ON/OFF) with suitable conditions and truth table. **(Dec. 09, May 10, 10 Marks)**

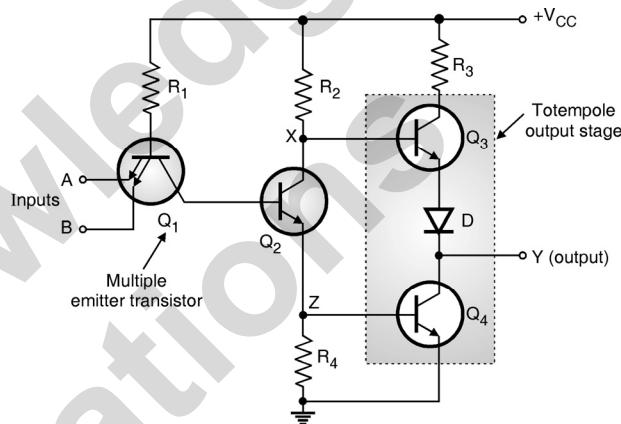
Q. 3 Explain the working of two input TTL NAND gate with active pull up. Consider various input, output states for explanation.

(May 12, Dec. 12, 8 Marks)

Q. 4 Draw and explain 2 inputs TTL NAND gate with totem pole output. **(Dec. 13, 6 Marks)**

Circuit diagram :

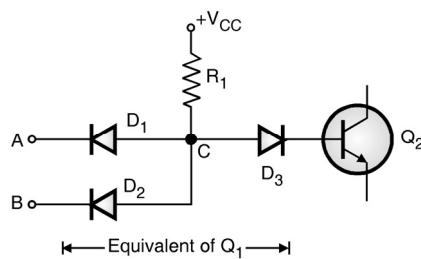
- A two input TTL-NAND gate is shown in Fig. 1.4.2.
- A and B are the two inputs while Y is the output terminal of this NAND gate.



(C-1012) Fig. 1.4.2 : Two input TTL NAND gate

Operation :

- In order to understand the operation of this circuit, let us replace transistor Q_1 by its equivalent circuit as shown in Fig. 1.4.3.

(C-1012) Fig. 1.4.3 : Transistor Q_1 is replaced by its equivalent

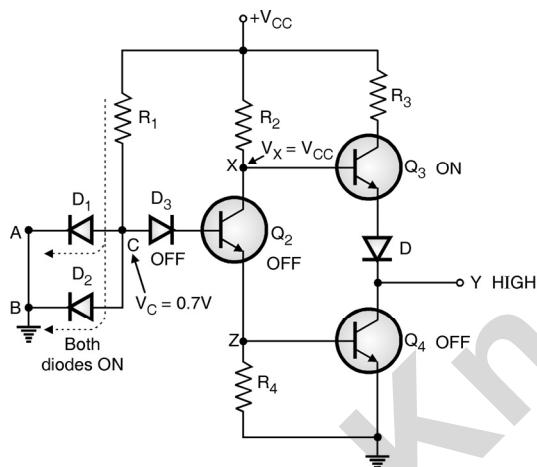
- A and B are the input terminals. The input voltages A and B can be either LOW (zero Volt ideally) or HIGH ($+V_{CC}$ ideally).
- A and B both LOW ($A = B = 0$) :**
 - If A and B both are connected to ground, then both the B-E junctions of transistor Q_1 are forward biased.
 - Hence diodes D_1 and D_2 in Fig. 1.4.3 will conduct to force the voltage at point C in Fig. 1.4.3 to 0.7 V.



- This voltage is insufficient to forward bias base-emitter junction of Q_2 due to the presence of D_3 . Hence Q_2 will remain OFF.
- Therefore its collector voltage V_X rises to V_{CC} .
- As transistor Q_3 is operating in the emitter follower mode, output Y will be pulled up to high voltage.

$$\therefore Y = 1 \text{ (HIGH)} \quad \dots \text{For } A = B = 0 \text{ (LOW)}$$

- The equivalent circuit for this input condition is shown in Fig. 1.4.4(a).

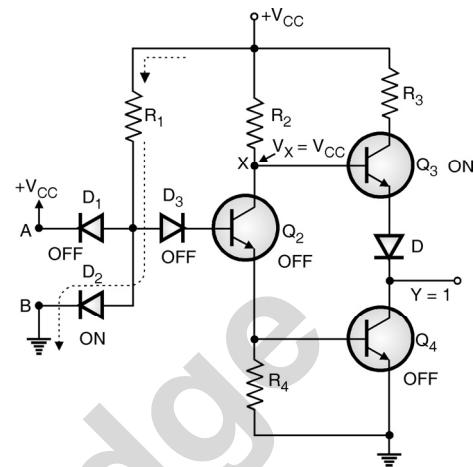
(C-1013) Fig. 1.4.4(a) : Equivalent circuit for $A = B = 0$

3. Either A or B LOW ($A = 0$ or $B = 0$) :

- If any one input (A or B) is connected to ground with the other terminal left open or connected to $+V_{CC}$, then the corresponding diode (D_1 or D_2) will conduct.
- This will pull down the voltage at "C" to 0.7 V. (Fig. 1.4.3)
- This voltage is insufficient to turn ON D_3 and Q_2 . So it remains OFF.
- So collector voltage V_X of Q_2 will be equal to V_{CC} . This voltage acts as base voltage for Q_3 .
- As Q_3 acts as an emitter follower, output Y will be pulled to V_{CC} .

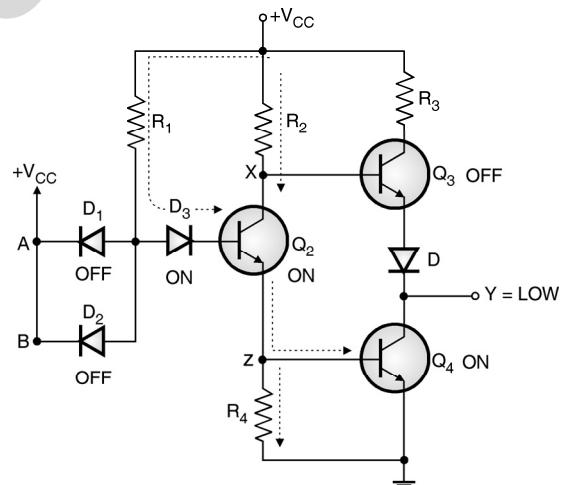
$$\therefore Y = 1 \begin{cases} \text{if } A = 0 \text{ and } B = 1 \\ \text{if } A = 1 \text{ and } B = 0 \end{cases} \quad (\text{C-6373})$$

- The equivalent circuit for this mode is shown in Fig. 1.4.4(b).

(C-1013) Fig. 1.4.4(b) : Equivalent circuit for $A = 1, B = 0$

4. A and B both HIGH ($A = B = 1$) :

- If A and B both are connected to $+V_{CC}$, then both the diodes D_1 and D_2 will be reverse biased and do not conduct.
- Therefore voltage at point "C" i.e. at the anode of D_3 increases to a sufficiently high value.
- Therefore diode D_3 is forward biased and base current is supplied to transistor Q_2 via R_1 and D_3 as shown in Fig. 1.4.4(c).

(C-1014) Fig. 1.4.4(c) : Equivalent circuit for $A = B = 1$

- As Q_2 conducts, the voltage at X will drop down and Q_3 will be OFF, whereas voltage at Z (across R_3) will increase to a sufficient level to turn ON Q_4 .
- As Q_4 goes into saturation, the output voltage Y will be pulled down to a low voltage.

$$\therefore Y = 0 \dots \text{For } A = B = 1$$



- The equivalent circuit for this mode of operation is shown in Fig. 1.4.4(c).
- This discussion reveals that the circuit operates as a NAND gate.

Truth Table :

- The Truth Table of Two input standard NAND gate is as follows.

(C-7538) Table 1.4.1 : Truth Table of a 2-input NAND gate

Inputs		Status of various transistors					Output Y
A	B	D ₁	D ₂	Q ₂	Q ₃	Q ₄	
0	0	ON	ON	OFF	ON	OFF	1
0	1	ON	OFF	OFF	ON	OFF	1
1	0	OFF	ON	OFF	ON	OFF	1
1	1	OFF	OFF	ON	OFF	ON	0

1.4.3 Totem-pole (Active Pull up) Output Stage :

SPPU : May 05, Dec. 07

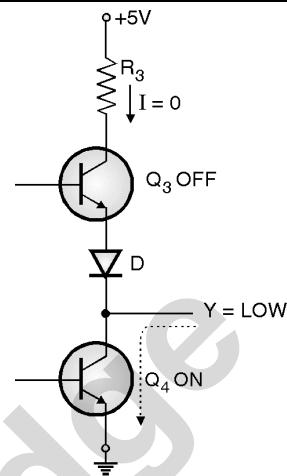
University Questions

- Q. 1** Give advantages and disadvantages of totem-pole output-stage arrangement. (May 05, 8 Marks)
- Q. 2** Will totempole output is suitable for wired-OR logic ? Justify your answer. (Dec. 07, 5 Marks)

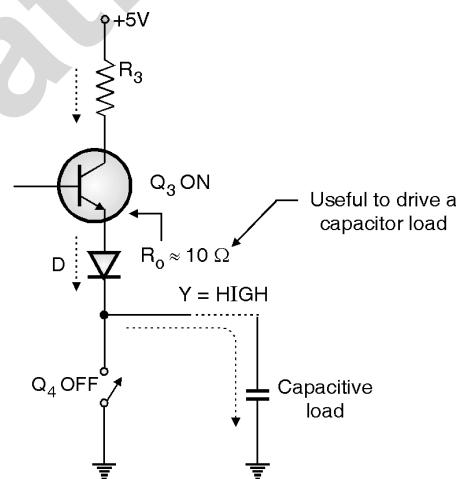
- The arrangement of Q₃ and Q₄ on the output side of a TTL NAND gate is called as the totem-pole arrangement.
- It is possible in TTL gates to speed up the charging of output capacitance without corresponding increase in the power dissipation with help of **Totem-pole** output stage.
- The Totem-pole output is also known as **active pull-up**.

Advantages of totem-pole output stage :

- The **advantages** of using the totem-pole output stage are as follows :
 - With Q₃ in the circuit, the current flowing through R₃ will be equal to zero when the output Y = 0, that means when Q₄ is ON, as shown in Fig. 1.4.5(a). This is important because it reduces the power dissipation taking place in the circuit.

(C-1015) Fig. 1.4.5(a) : No power dissipation in R₃

2. Another advantage of totem-pole arrangement is when the output Y is HIGH. Here Q₃ is ON and acting in the emitter follower mode. It will therefore have a very low output impedance (typically 10 Ω). Therefore the output time constant will be very short for charging up any capacitive load on the output as shown in Fig. 1.4.5(b).



(C-1015) Fig. 1.4.5(b) : Low output resistance

Disadvantages of Totem-pole output :

1. Q₄ in the totem-pole output turns OFF more slowly than Q₃ turns ON.
2. So before Q₄ is completely turned OFF, Q₃ will come into conduction. So for a very short duration of few nanoseconds, both the transistors will be simultaneously ON.
3. This is called as **cross conduction** and it will draw relatively large current (30 to 40 mA) from the 5 V supply.

Function of diode D :

- Diode D is added in the circuit in order to keep Q_3 OFF when Q_4 is already conducting.
- It is important to avoid simultaneous conduction of Q_3 and Q_4 , because it will lead to cross conduction and will increase power dissipation.
- Thus D_3 is used for successfully avoiding the cross conduction.

Function of R_3 :

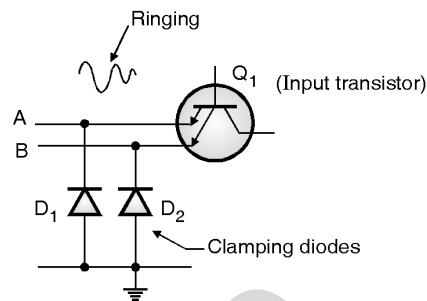
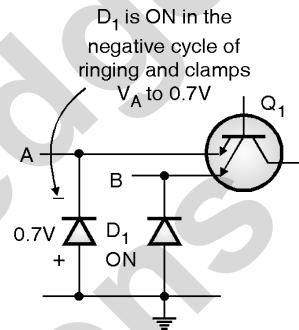
- During the cross conduction, if R_3 is not used then there will be no current limiting element in series with Q_3 and Q_4 and a heavy current will be drawn from the source which can damage the IC.
- This can be avoided by limiting the current by inserting resistor R_3 in series with Q_3 .

1.4.4 Unconnected Inputs :

- If any input of a TTL gate is left open, disconnected or floating, then the corresponding base emitter junction of the input transistor Q_1 is not forward biased as shown in Fig. 1.4.6(a).
- Therefore the open or floating input is equivalent to a logical 1 is applied to that input.
- Hence in TTL ICs all the unconnected inputs are treated as logical 1s.
- However, the unused inputs should either be connected to some used input (s) or returned to V_{CC} through a suitable resistor.

1.4.5 Clamping Diodes :

- The TTL inputs should not be subjected to negative voltages. Under normal operating conditions this gets managed easily.
- But if fast voltage transitions are applied at the input, then there is a possibility of ringing (sinusoidal oscillations with positive and negative half cycles). Due to ringing, the inputs will be subjected to negative voltages.
- To suppress this ringing, clamping diodes are generally connected externally in all the TTL circuits as shown in Fig. 1.4.6(a).

**(a) Clamping diodes****(b) Effect of clamping diodes**

(C-1019) Fig. 1.4.6

- These are fast recovery diodes. They are forward biased during the negative half cycles of the ringing sinusoidal waveform. Hence the negative input voltage will be restricted (clamped) to - 0.7 Volt approximately as shown in Fig. 1.4.6(b).

1.4.6 5400 Series :

- The temperature range for the devices in 7400 series is from 0 to 70°C, over a supply voltage range of 4.75 to 5.25 V.
- So 7400 series is used for the commercial applications.
- But 5400 TTL series is developed specially for the military applications.
- The devices of 5400 series operate over the temperature range of - 55 to 125°C and over the supply voltage range of 4.5 to 5.5 Volts.

1.4.7 Three Input TTL NAND Gate :

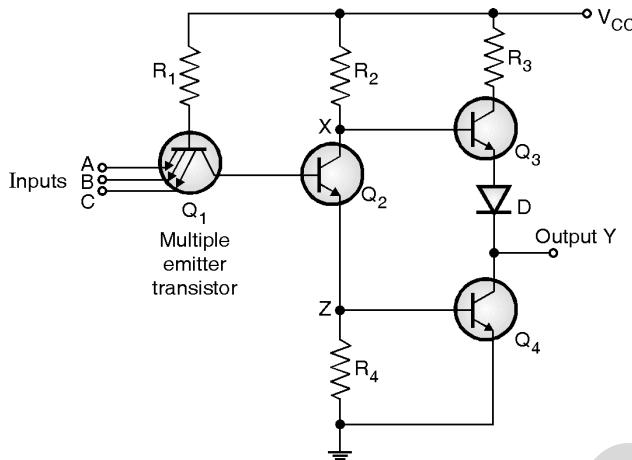
SPPU : Dec. 10, May 11

University Questions

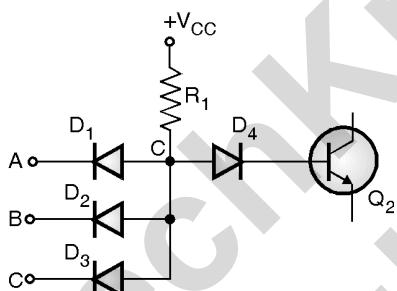
- Q. 1** Draw and explain the design of 3 input TTL NAND gate circuit. Also explain various input, output states and corresponding transistor (ON/OFF) states. **(Dec. 10, May 11, 12 Marks)**

Circuit diagram :

- The circuit diagram of a three input NAND gate is as shown in Fig. 1.4.7.
- Note that the multiple emitter transistor has three emitter terminals which act as the inputs A, B and C to the NAND gate.



(a) Circuit diagram

(b) Equivalent circuit of the multiple emitter transistor Q_1

(C-1685) Fig. 1.4.7 : Three input TTL NAND gate

Operation :

- The principle of operation of this circuit is exactly same as that of the two input NAND gate discussed in section 1.4.7.
- If at least one of the inputs is low (0) then at least one of the diodes D_1 , D_2 , D_3 in Fig. 1.5.2(b) will be conducting. Hence the voltage at point C will be clamped to 0.7 V.
- Hence Q_2 will be off. So Q_3 will conduct and the output $Y = 1$ (HIGH).
- $\therefore Y = 1 \text{ if at least one input is } 0.$
- If $A = B = C = 1$ then D_1 , D_2 , D_3 will be off. So Q_2 will be turned on. So Q_3 will be turned off and Q_4 will be turned on. So the output $Y = 0$ (LOW).

$\therefore Y = 0 \text{ if all the inputs are } 1.$

Truth Table :

- Table 1.4.2 shows the truth table and the status of various transistors in the circuit.

(C-6351) Table 1.4.2 : Truth table of the 3-input NAND gate

Input	Status of various transistors			Output Y	
	Q_2	Q_3	Q_4		
A	B	C			
0	0	0	OFF	ON	1
0	0	1	OFF	ON	0
0	1	0	OFF	ON	1
0	1	1	OFF	ON	0
1	0	0	OFF	ON	1
1	0	1	OFF	ON	0
1	1	0	OFF	ON	1
1	1	1	ON	OFF	0

1.5 Open Collector Outputs (TTL) :

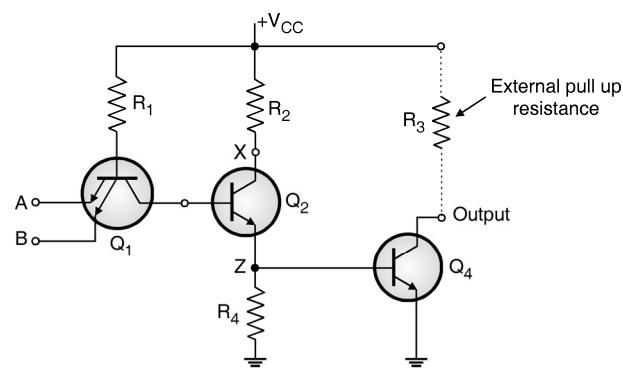
SPPU : May 08, Dec. 11, May 15

University Questions

- Q. 1** What is the advantage of open collector output ? Justify your answer with suitable circuit diagram. **(May 08, 8 Marks)**
- Q. 2** Explain the working of two input TTL NAND gate with open collector output. Consider various input, output states for explanation. **(Dec. 11, 8 Marks)**
- Q. 3** What do you mean by open collector output ? Explain with suitable circuit diagram. What is the advantage of this output ? **(May 15, 6 Marks)**

Circuit diagram :

- We have seen that the gates having totem-pole output cannot be wired ANDed.
- Such a connection becomes possible if another type output stage called open collector output is used.
- The circuit diagram of a 2-input NAND gate is shown in Fig. 1.5.1.
- You will realize that this is the same TTL NAND gate which we have discussed earlier but with R_3 and Q_3 removed.



(C-1027) Fig. 1.5.1 : Open collector 2 input NAND gate



- The collector point of Q_4 is brought out as output as shown in Fig. 1.5.1, therefore it is called as open collector output.
- For proper operation it is necessary to connect an external resistance R_3 between V_{CC} and the open collector output as shown in Fig. 1.5.1.
- This resistance is called as **pull up** resistance.

Operation :

1. With $A = B = 0$:

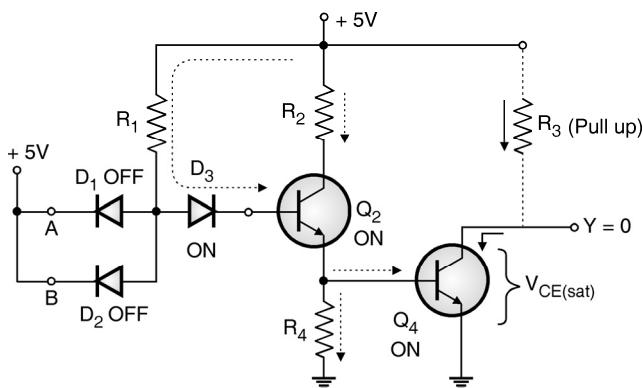
- With $A = B = 0$, both the BE junctions of Q_1 are forward biased. So Q_2 remains OFF.
- Hence no current flows through R_4 . So $V_Z \approx 0 V$.
- Therefore Q_4 is OFF and its collector voltage is equal to V_{CC} . So $Y = 1$ when $A = B = 0$.

2. With $A = 0, B = 1$ OR $A = 1, B = 0$:

- One of the BE junctions is forward biased (the one corresponding to 0 input).
- So Q_2 is OFF and Q_4 also is OFF. So its collector voltage is equal to V_{CC} .
- Therefore output $Y = 1$ when any one input is low.

3. With $A = B = 1$:

- When both the inputs are high, transistor Q_1 is turned OFF.
- So Q_2 will be turned ON. Sufficient voltage is developed across R_4 . Base current is applied to Q_4 and Q_4 goes into saturation.
- So the output voltage is equal to $V_{CE(sat)}$ of Q_4 which is very small. Thus $Y = 0$ when $A = B = 1$. The equivalent circuit for this mode is shown in Fig. 1.5.2.



(C-1028) Fig. 1.5.2 : Equivalent circuit of open collector 2-input NAND gate with $A = B = 1$

1.5.1 Disadvantages of Open Collector Output :

1. The value of pull up resistance is high (few $k\Omega$). Therefore if the load capacitance is large then the RC time constant ($R_3 C$) becomes large. This slows down the switching speed of Q_4 . Therefore the gates having an open collector output will be slow.
2. Second disadvantage is increased power dissipation. When Q_4 is ON, a large current flows through the pull up R_3 . Hence power dissipation is increased. This problem is eliminated if we use totem-pole output arrangement.

1.5.2 Advantage of Open Collector Output :

SPPU : May 15

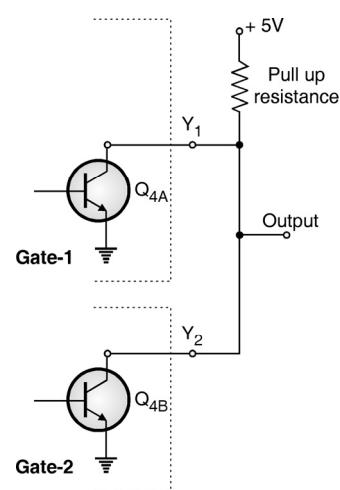
University Questions

- Q. 1** What do you mean by open collector output ? Explain with suitable circuit diagram. What is the advantage of this output ? **(May 15, 6 Marks)**

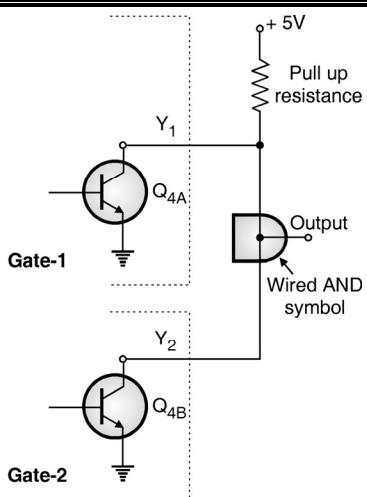
- The main advantage of open collector output is that **Wired ANDing** becomes possible.

1.5.3 Wired ANDing :

- Wired ANDing means connecting the outputs of gates together to obtain the AND function.
- It is possible to connect the outputs of two or more gates together as shown in Fig. 1.5.3(a).
- The wired ANDing is represented schematically as shown in Fig. 1.5.3(b).

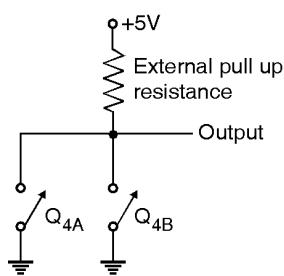


(C-1029) Fig. 1.5.3(a) : Wired ANDing of two open collector TTL gates

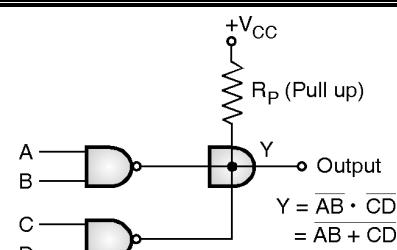


(C-1029) Fig. 1.5.3(b) : Symbol of wired ANDing

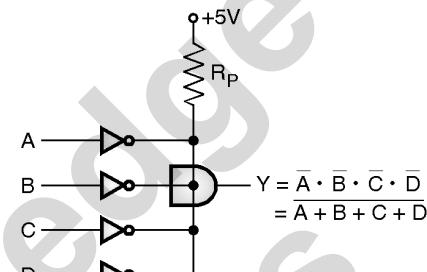
- Q_{4A} and Q_{4B} are the output transistors of Gate-A and Gate-B respectively. A common pull up resistance is used for all the output transistors.
- The transistors Q_{4A} and Q_{4B} are operated as switches. They operate in either saturation or cut off regions.
- Therefore the wired ANDing equivalent circuit is as shown in Fig. 1.5.4(a).
- The equivalent circuit of Fig. 1.5.4(a) indicates that the wired AND output will be "0" if any one or all the output transistor(s) are conducting.
- The output will be high (1) if and only if all the transistors are in their OFF state.
- This is the reason why such a connection is called as wired AND connection.
- Figs. 1.5.4(b), (c) and (d) shows that it is possible to wire AND the outputs of any gate such as NAND, NOR or inverter.
- The corresponding output Boolean equations are given alongwith the diagrams.



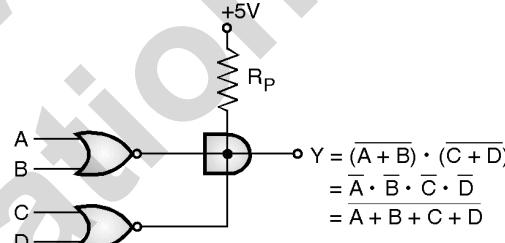
(C-1030) Fig. 1.5.4(a) : Equivalent circuit of wired ANDing



(b) Wired ANDing of NAND gate outputs



(c) Wired ANDing of inverter outputs



(d) Wired ANDing of the NOR outputs

(C-1031) Fig. 1.5.4

1.5.4 Comparison of Totem-pole and Open Collector Outputs :

SPPU : Dec. 05, May 07, Dec. 13

University Questions

- Q. 1** With the help of circuit diagram explain the difference between totempole output and open collector output. (Dec. 05, May 07, 10 Marks)
- Q. 2** Compare totempole and open collector output configurations in TTL. (Dec. 13, 6 Marks)

- Table 1.5.1 gives the comparison of totem-pole and open collector outputs of TTL.

Table 1.5.1 : Comparison of totem-pole and open collector outputs

Sr. No.	Parameter	Totem-pole	Open collector
1.	Circuit components on the output side	Q_3 (pull up transistor), D and Q_4 (pull down transistor) are used.	Only the pull down transistor Q_4 is used.



Sr. No.	Parameter	Totem-pole	Open collector
2.	Wired ANDing	Cannot be done	Easily be done
3.	External pull up resistor	Not required	Required to be connected.
4.	Power Dissipation	Low due to presence of pull up transistor Q_3 .	High due to current flowing through R_3 .
5.	Speed	Operating speed is high.	Operating speed is low.

1.6 Standard TTL Characteristics :

SPPU : May 11, Dec. 11,

May 12, Dec. 14, Dec. 15, Dec. 17

University Questions

Q. 1 Explain various characteristics for TTL logic families. **(May 11, Dec. 14, 4 Marks)**

Q. 2 Define the following terms related to logic families. Mention typical values for standard TTL family :

- 1. Propagation delay 2. Fan-out
- 3. V_{IL} , V_{IH} 4. Noise margin

(Dec. 11, May 12, 8 Marks)

Q. 3 Explain the following TTL characteristics :

- 1. Noise Immunity
- 2. High level input voltage (V_{NH})
- 3. Figure of Merit **(Dec. 15, 6 Marks)**

Q. 4 Explain standard TTL characteristics in brief. **(Dec. 17, 6 Marks)**

- The Texas instrument first introduced two TTL series namely 54 series and 74 series.

Supply voltage and temperature ranges :

- Table 1.6.1 gives the supply voltage and temperature ranges for two TTL families.

Table 1.6.1

TTL series	Supply voltage range	Temperature range
74 series	4.75 V to 5.25 V	0° C to 70° C
54 series	4.5 V to 5.5 V	- 55° C to 125° C

Voltage levels :

- Table 1.6.2 shows the input and output logic voltage levels for the TTL 74 series.

Table 1.6.2 : Voltage levels for TTL 74 series

Voltages	Minimum	Typical	Maximum
V_{OL}	-	0.2 V	0.4 V
V_{OH}	2.4 V	3.4 V	-
V_{IL}	-	-	0.8 V
V_{IH}	2.0 V	-	-

Noise margin :

- We have already defined the noise margins as,
High level noise margin,

$$V_{NH} = V_{OH}(\min) - V_{IH}(\min)$$

and Low level noise margin,

$$V_{NL} = V_{IL}(\max) - V_{OL}(\max)$$

- Substituting the values from Table 1.6.2, the noise margin for the TTL logic families can be calculated.

$$\therefore V_{NH} = 2.4 - 2 = 0.4 \text{ V.}$$

$$V_{NL} = 0.8 - 0.4 = 0.4 \text{ V.}$$

- Thus noise margin for the TTL family is 0.4 V. This means that as long as the induced noise voltage is less than 0.4 V, the operation of the TTL ICs will be unaffected.
- The noise margins have been shown in section 1.3.3.

Power dissipation :

- The average power dissipation for the standard TTL 74 series is approximately 10 mW.
- It is dependent on the parameters such as tolerances, signal level etc.

Propagation delay :

- The propagation delay of a standard TTL gate is approximately 10 nanoseconds (1 nanosecond = 10^{-9} seconds)

Fan out :

- A standard TTL gate is capable of driving at the most 10 other TTL gates simultaneously, maintaining its output voltage within the specified limits. Hence fan out of a TTL gate is 10.
- All the important characteristics of TTL logic family are listed in Table 1.6.3.

**Table 1.6.3 : Important parameters of TTL logic family**

Sr. No.	Parameter	Values
1.	Supply voltage	74 series : (4.75 to 5.25 V) 54 series : (4.5 to 5.5 V)
2.	Temperature range	74 series : 0 to 70° C 54 series : -55 to 125° C
3.	Voltage levels	$V_{IL(max)} = 0.8 \text{ V}$ $V_{OL(max)} = 0.4 \text{ V}$ $V_{IH(min)} = 2 \text{ V}$ $V_{OH(min)} = 2.4 \text{ V}$
4.	Noise margin	0.4 V
5.	Power dissipation	10 mW
6.	Propagation delay	10 nanosec.
7.	Fan out	10
8.	Figure of merit	100

1.6.1 Advantages of TTL :

1. TTL circuits are fast.
2. Low propagation delay.
3. Power dissipation is not dependent on frequency.
4. Compatible to all the other families.
5. Latch ups do not take place.
6. These are not susceptible to the damage due to static charges.
7. Higher current sourcing and sinking capabilities.

1.6.2 Disadvantages of TTL :

1. Large power dissipation.
2. Fan out is lower than that of CMOS.
3. Less component density.
4. Can operate only on + 5 V power supply.
5. Poor noise immunity.

1.7 MOS - Logic Family :

- The MOS - logic family uses MOSFET as the basic device the way TTL uses BJT.
- The MOSFETs are of two types namely the depletion MOSFETs and the enhancement type MOSFETs.
- In logic circuits, the enhancement MOSFETs are used. We use the E-MOSFET as a switch.

1.8 CMOS Logic :

Definition :

- CMOS stands for complementary MOSFET. It is obtained by using a p-channel MOSFET and n-channel MOSFET simultaneously.
- The p and n channel MOSFETs are connected in series, with their drains connected together and output taken from common drain point.
- Input is applied at the common gate terminal formed by connecting two gates together.
- The 74C00 CMOS series is a group of CMOS circuits which are pin-for-pin and function for function compatible with the TTL 7400 devices.
- For example 74C32 is a quad 2-input OR gate in the CMOS family whereas 7402 is a quad 2 input TTL gate in the 7400 TTL family.

1.8.1 CMOS NAND Gate :

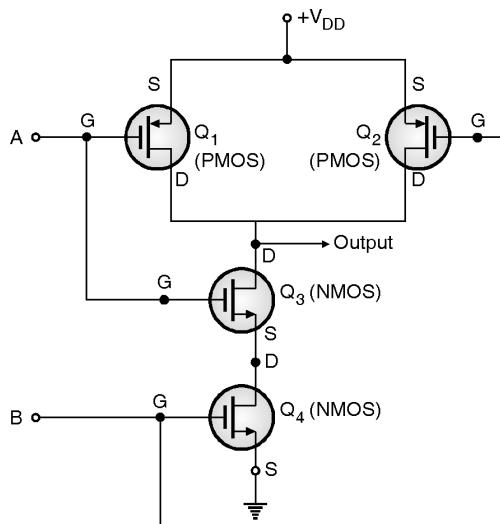
SPPU : Dec. 11, Dec. 12

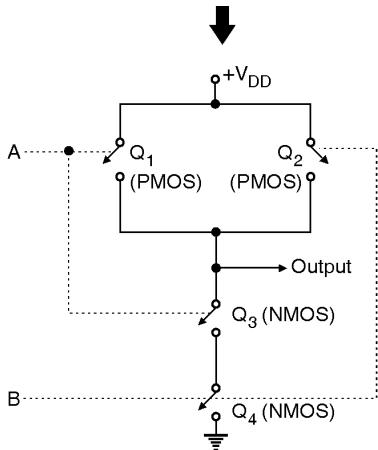
University Questions

- Q. 1** Draw the structure of two input CMOS NAND gate. Explain its working. **(Dec. 11, 4 Marks)**
- Q. 2** Draw and explain the working of a 2-input CMOS NAND gate. **(Dec. 12, 8 Marks)**

Circuit diagram :

- The two input CMOS NAND gate is shown in Fig. 1.8.1(a) and its equivalent circuit by replacing each MOSFET by a switch is shown in Fig. 1.8.1(b).

**(a) 2-input CMOS NAND gate****Fig. 1.8.1(Contd...)**



(b) Equivalent circuit

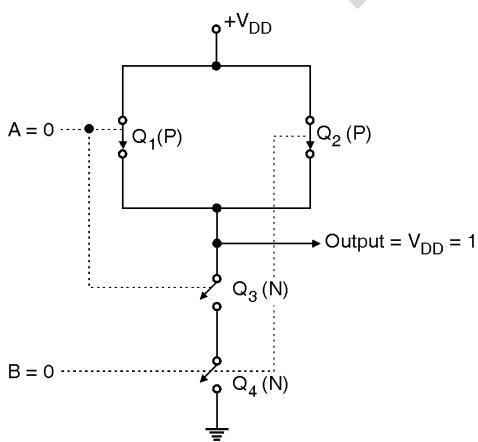
(C-1058) Fig. 1.8.1

- Q_1 and Q_2 are p-channel MOSFETs. They are connected in parallel with each other.
- Q_3 and Q_4 are n-channel MOSFETs. They are connected in series with each other.
- Input A is connected to gates of Q_1 and Q_3 . So A controls the status of MOSFETs Q_1 and Q_3 .
- Input B is connected to gates of Q_2 and Q_4 . So B controls the status of MOSFETs Q_2 and Q_4 .

Operation :

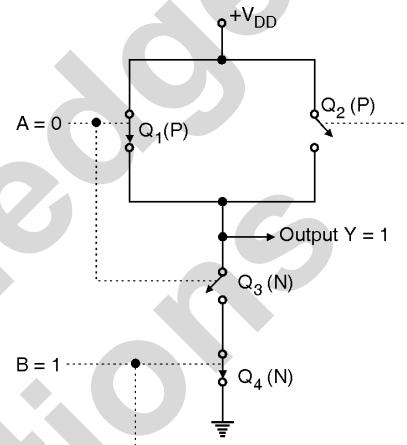
1. $A = B = 0$:

- With $A = 0$ and $B = 0$, both the PMOSFETs i.e. Q_1 and Q_2 will be ON. But both the N-MOSFETs i.e. Q_3 and Q_4 will be OFF.
- As seen from the equivalent circuit of Fig. 1.8.2(a), the output $Y = +V_{DD}$ (logic 1).
- So $Y = 1$ if $A = B = 0$.

(C-1059) Fig. 1.8.2(a) : Equivalent circuit for $A = B = 0$

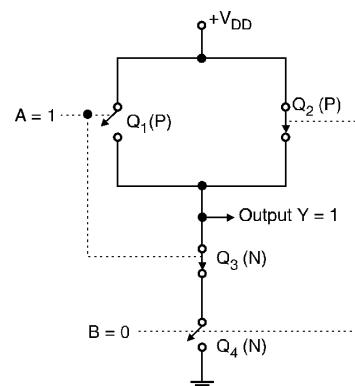
2. With $A = 0$ and $B = 1$:

- With $A = 0$ and $B = 1$, Q_1 will continue to be ON and Q_3 continues to be OFF. But Q_2 will now turn OFF and Q_4 will be turned ON.
 - The equivalent circuit of this mode is shown in Fig. 1.8.2(b) which shows that output $Y = +V_{DD}$ i.e. logic 1.
1. So $Y = 1$ if $A = 0$ and $B = 1$.

(C-1059) Fig. 1.8.2(b) : Equivalent circuit for $A = 0, B = 1$

3. With $A = 1$ and $B = 0$:

- With $A = 1$, Q_1 will be turned OFF and Q_3 will turn ON.
- And with $B = 0$, Q_2 will be turned ON and Q_4 will be turned OFF.
- As seen from the equivalent circuit of Fig. 1.8.2(c), the output $Y = +V_{DD}$ (logic 1).

(C-1059) Fig. 1.8.2(c) : Equivalent circuit for $A = 1, B = 0$

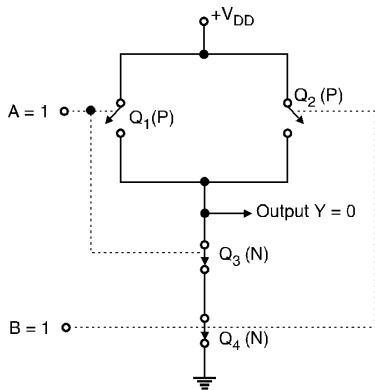
- So $Y = 1$ if $A = 1$ and $B = 0$.

4. With $A = 1, B = 1$:

- With $A = B = 1$ both P-MOSFETs i.e. Q_1 and Q_2 will be OFF and both the N-MOSFETs i.e. Q_3 and Q_4 will be ON.



- The equivalent circuit of this mode is shown in Fig. 1.8.2(d).



(C-1060) Fig. 1.8.2(d) : Equivalent circuit for A = 1, B = 1

- It shows that output Y = 0 (LOW).
So Y = 0 if A = B = 1.

Truth Table :

- Table 1.8.1 shows the truth table of the two input NAND gate.

Table 1.8.1 : Truth Table of a CMOS NAND gate

Inputs		Transistors				Output Y
A	B	Q ₁	Q ₂	Q ₃	Q ₄	
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

1.8.2 CMOS Series :

- The popular CMOS series are 4000/14000 series, 74C series, 74 HC/HCT (High speed CMOS), 74 AC/ACT (Advanced CMOS).

1.9 Standard CMOS Characteristics :

- Some of the important CMOS characteristics are as follows :

1.9.1 Power Supply Voltage :

- The 4000/14000 series and 74C series CMOS ICs are capable of operating over a wide range of power supply voltage (typically 3 V to 15 V).
- That means the CMOS ICs from these series are extremely versatile. They can operate on 3 V batteries as well as 5 V TTL compatible power supplies.

- These ICs can operate on higher power supply voltages, when higher noise margin is required.
- The other CMOS families such as 74 HC/HCT, 74 AC/ACT and 74 AHC/AHCT operate over a voltage range of 2 to 6 V.

1.9.2 Logic Voltage Levels :

- The logic voltage levels for different CMOS series have different values. Table 1.9.1 lists various logic voltage levels for different CMOS series.

Table 1.9.1 : Logic voltage levels (in volts) with

$$V_{DD} = V_{CC} = + 5 \text{ V}$$

Para-meter	CMOS								TTL			
	4000B	74 HC	74 HCT	74 AC	74 ACT	74 AHC	74 AHCT	74 LS	74 AS	74 ALS		
V _{IH(min)}	3.5	3.5	2.0	3.5	2.0	3.85	2.0	2.0	2.0	2.0	2.0	2.0
V _{IL(max)}	1.5	1.0	0.8	1.5	0.8	1.65	0.8	0.8	0.8	0.8	0.8	0.8
V _{OH(min)}	4.95	4.9	4.9	4.9	4.9	4.4	3.15	2.4	2.7	2.7	2.7	2.7
V _{OL(max)}	0.05	0.1	0.1	0.1	0.1	0.44	0.1	0.4	0.5	0.5	0.4	0.4
V _{NH}	1.45	1.4	2.9	1.4	2.9	0.55	1.15	0.4	0.7	0.7	0.7	0.7
V _{NL}	1.45	0.9	0.7	1.4	0.7	1.21	0.7	0.4	0.3	0.3	0.4	0.4

Important points from Table 1.9.1 :

- All the voltage levels shown in Table 1.9.1 correspond to a supply voltage of + 5 Volts.
- Note that V_{OH} ≈ 5 V and V_{OL} ≈ 0 V.

1.9.3 Noise Margins :

SPPU : May 10

University Questions

- Q. 1** Describe what happens to the following CMOS characteristic as V_{DD} is increased : Noise margin and in which applications is CMOS ideally suited ?
(May 10, 3 Marks)

- Table 1.9.1 contains the high level and low level noise margins V_{NH} and V_{NL} for each CMOS and TTL series.
- These are calculated as :
$$V_{NH} = V_{OH(min)} - V_{IH(min)},$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)}.$$
- It can be observed that the CMOS devices will have higher noise margins as compared to those of TTL. So CMOS ICs should be preferred to TTL for operation in the noisy environment.
- The noise margins in CMOS will increase further if the supply voltages are increased further.



1.9.4 Power Dissipation : SPPU : May 06

University Questions

Q. 1 Define power dissipation and give typical values of this parameters with respect to CMOS logic family.
(May 06, 2 Marks)

- The power dissipation of CMOS devices is extremely low when they are in the static (stable) state.
- Typically the dc power dissipation is 2.5 nW per gate when $V_{DD} = 5$ V and 10 nW for $V_{DD} = 10$ V. This is too small as compared to TTL gates ($P_D = 10$ mW).
- Hence CMOS devices are preferred for the battery operated systems.
- But power dissipation is low under the dc operating conditions and at low frequencies.
- But power dissipation increases as the circuit switching frequency is increased.
- The relation between power dissipation and frequency is demonstrated in Table 1.9.2.

Table 1.9.2 : Relation between P_D and frequency

Frequency	0 (dc)	100 kHz	1 MHz
Power dissipation P_D	10 nW	0.1 mW	1 mW

1.9.5 Fan Out : SPPU : May 06

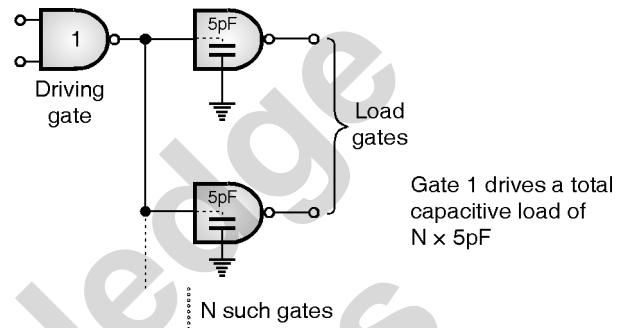
University Questions

Q. 1 Define fan out and give typical values of this parameters with respect to CMOS logic family.
(May 06, 2 Marks)

- The input resistance of CMOS devices is very high (10^{12} Ω). So their input current is very very small, almost zero.
- Therefore one CMOS gate can drive a large number of other CMOS gates. Hence fan out of CMOS devices will be large as compared to fan out of TTL.
- Typically the fan out is restricted to 50 for operation below 1 MHz.
- Why is fan out restricted to only 50 ? This is because the input capacitance of each CMOS gate is about 5 pF.

These capacitors act as load on the driving gate as shown in Fig. 1.9.1.

- The charging current for these capacitor has to be supplied by the driving gate. This current should not be too large. This will limit the fan-out to 50.



(C-1061) Fig. 1.9.1 : Fan out and switching speed are dependent on the input capacitance of load gates

1.9.6 Switching Speed : SPPU : May 06

University Questions

Q. 1 Define speed of operation and give typical values of this parameters with respect to CMOS logic family.
(May 06, 2 Marks)

- The output resistance of CMOS is low in both the states (0 or 1) of output.
- So even though it has to drive large capacitive loads, the switching speed can still be faster than the NMOS or PMOS devices.
- The NAND gate of 4000 series has following values of propagation delays :
 - Average t_{pd} = 50 nS at $V_{DD} = 5$ V
 - Average t_{pd} = 25 nS at $V_{DD} = 10$ V
- The average time delay for various CMOS ICs are given in Table 1.9.3.

(C-6352) Table 1.9.3 : Switching speeds for various CMOS ICs at $V_{DD} = 5$ V

Family	4000	74 HC/HCT	74 AC/ACT	74 AHC
Propagation delay t_{pd}	50nS	8nS	4.7nS	4.3nS

1.9.7 Unconnected Inputs :

- Note that the CMOS inputs should never be left floating.



- All the CMOS inputs should be either connected to 0 V (ground) or V_{DD} or to another inputs.
- This is necessary to avoid permanent damage of CMOS ICs.
- Sometimes there are some unused gates on a chip. The inputs of such gates also should be connected to ground or + V_{DD} .
- The CMOS ICs are damaged due to induced voltages at the floating inputs due to noise or static charges.
- Such voltages can bias the P-MOS or N-MOS in their conduction state that may cause overheating and damage.

1.9.8 Advantages of CMOS : SPPU : Dec. 07

University Questions

Q. 1 What are the advantages of CMOS devices over TTL devices ? Explain in short. (Dec. 07, 4 Marks)

1. Low power dissipation.
2. High fan out (typically 50).
3. High noise margin for higher values of V_{DD} .
4. Capable of working over a wide range of supply voltage.
5. Switching speeds comparable to those of TTL.
6. High packaging density since (more devices can be accommodated in the same space) MOS devices need less space.

1.9.9 Disadvantages of CMOS :

1. Propagation delays longer than those of TTL (25 to 100 nS).
2. Slower than TTL.
3. CMOS ICs can get damaged due to static charge.
4. Latch ups can take place which can damage the device.
5. Need protection circuitry.

1.10 Comparison of CMOS and TTL :

SPPU : Dec. 11, Dec. 12, Dec. 14, Dec. 18

University Questions

Q. 1 List difference between CMOS and TTL. (Dec. 11, 4 Marks)

Q. 2 Compare TTL and CMOS logic family. (Dec. 12, Dec. 18, 6 Marks)

Q. 3 Compare TTL and CMOS logic family. Draw CMOS NOR gate. (Dec. 14, 6 Marks)

Table 1.10.1 : Comparison of CMOS and TTL

Sr. No.	Parameter	CMOS	TTL
1.	Device used	N-channel MOSFET and P-channel MOSFET	Bipolar junction transistor
2.	$V_{IH\ (min)}$	$3.5\text{ V} (V_{DD} = 5\text{ V})$	2 V
3.	$V_{IL\ (max)}$	1.5 V	0.8 V
4.	$V_{OH\ (min)}$	4.95 V	2.7 V
5.	$V_{OL\ (max)}$	0.05 V	0.4 V
6.	High level noise margin	$V_{NH} = 1.45\text{ V}$	0.4 V
7.	Low level noise margin	$V_{NL} = 1.45\text{ V}$	0.4 V
8.	Noise immunity	Better than TTL	Less than CMOS
9.	Propagation delay	105 nS (Metal gate CMOS)	10 nS. (Standard TTL)
10.	Switching speed	Less than TTL.	Faster than CMOS
11.	Power dissipation per gate.	$P_D = 0.1\text{ mW}$. Hence used for battery backup applications	10 mW
12.	Speed power product.	10.5 pJ	100 pJ
13.	Dependence of P_D on frequency	P_D increases with increase in frequency.	P_D does not depend on frequency.
14.	Fan out	Typically 50.	10
15.	Unconnected inputs	Unused inputs should be returned to GND or V_{DD} . They should never be left floating.	Inputs can remain floating. The floating inputs are treated as logic 1s.



Sr. No.	Parameter	CMOS	TTL
16.	Component density	More than TTL since MOSFETs need smaller space while fabricating an IC.	Less than CMOS since BJT needs more space.
17.	Operating areas	MOSFETs are operated as switches. i.e. in the ohmic region or cut off region.	Transistors are operated in saturation or cut off regions.
18.	Power supply voltage	Flexible from 3 V to 15 V.	Fixed equal to 5 V.

Configuration	AND	NAND	OR	NOR	XOR	XNOR
Triple 3-Input	4073	4023	4075	4025		
Dual 4-Input	4082	4012	4072	4002		
Single 8-Input	4068	4068	4078	4078		

Quad 2-Input NAND = 4093 (schmitt trigger inputs)

Dual 2-Input NAND = 40107 open drain outputs

Other circuits :

4008	4-bit binary full adder
4006	Shift Registers
4017	Decade Counter
4024	Binary Counter
4026	7-Segment Decoder
4027	Dual M-S JK Flip-Flop
4028	10:1 Multiplexer
4046	PLL
4047	Monostable/astable Multivibrator

Review Questions**1.11 Case Study : CMOS 4000 series ICs :****CMOS Series :**

- The popular CMOS series are 4000/14000 series, 74C series, 74 HC/HCT (High speed CMOS), 74 AC/ACT (Advanced CMOS).
- The following is a list of CMOS 4000-series digital logic integrated circuits.

Logic gates :**1. One input logic gates :**

Quad Buffer/Inverter = 4041 (4x CMOS drive)

Quad Buffer = 40109 (dual power-rails for voltage-level translation)

Hex Buffer = 4504 (dual power-rails for voltage-level translation)

Hex Buffer = 4050 (4x 74LS drive)

Hex Inverter = 4049 (4x 74LS drive)

Hex Inverter = 4069

Hex Inverter = 40106 (S inputs)

2. Two to eight input logic gates :

Configuration	AND	NAND	OR	NOR	XOR	XNOR
Quad 2-Input	4081	4011	4071	4001	4070	4077

- Q. 1 Which are the different logic families ? Write their characteristics.
- Q. 2 Explain the use of multi-emitter inputs.
- Q. 3 Define the following terms regarding a logic family :
- Noise margin
 - Propagation delay
- Q. 4 Compare the performance of TTL and CMOS logic.
- Q. 5 Explain the features of complementary symmetry logic (CMOS).
- Q. 6 Mention the advantages and disadvantages of TTL, and CMOS IC families.
- Q. 7 Define any four characteristics of logic gates.
- Q. 8 Classify the IC according to their scale of integration.
- Q. 9 Explain what is meant by TTL ?



Q. 10 Draw the circuit diagram of two input TTL NAND gate and explain its function.

Q. 11 Explain briefly the operation of CMOS NAND gate.

Q. 12 Give the important characteristics of CMOS logic family and explain their importance.

Q. 13 State specifications of standard TTL family.



TechKnowledge
Publications

Unit 1

Chapter 2

Number Systems and Codes

Syllabus

Binary, BCD, Octal, Hexadecimal, Excess-3, Gray code and their conversions.

Case study : Practical applications of various codes in computers.

Chapter Contents

2.1 Introduction	2.11 Conversion from Binary to Other Systems
2.2 System or Circuit	2.12 Conversion from Other Systems to Binary System
2.3 Binary Logic and Logic Levels	2.13 Conversion from Octal to Other Systems
2.4 Number Systems	2.14 Conversions Related to Hexadecimal System
2.5 The Decimal Number System	2.15 Concept of Coding
2.6 The Binary Number System	2.16 Classification of Codes
2.7 Octal Number System	2.17 Binary Coded Decimal (BCD) Code
2.8 Hexadecimal Number System	2.18 Non – weighted Codes
2.9 Conversion of Number Systems	2.19 Gray Code
2.10 Conversions Related to Decimal System	2.20 Code Conversions

2.1 Introduction :

- In the modern world of electronics, the term "digital" is generally associated with a computer.
- This is because, the term "digital" is derived from the way computers perform operation, by counting digits.
- For many years, the main application of digital electronics was only in the computer systems.
- But today, the digital electronics is used in many other applications such as : TV, Radar, Military systems, Medical equipments, Communication systems, Industrial process control and Consumer electronics.

Signals :

- We can define "signal" as a physical quantity, which contains some information and which is a function of one or more independent variables.
- The signals can be of two types :
 1. Analog signals
 2. Digital signals.

Digital Signals :

Definition:

- A **digital signal** is defined as the signal which has only a finite number of distinct values.
- Digital signals are not continuous signal. They are discrete signals.

Binary signal :

- If a digital signal has only two distinct values, i.e. 0 and 1 then it is called as a binary signal.

Octal signal :

- A digital signal having eight distinct values is called as an octal signal.

Hexadecimal signal :

- A digital signal having sixteen distinct values is called as the hexadecimal number.

2.2 System or Circuit :

Definition :

- A system or circuit is defined as the physical device or group of devices or algorithm which performs the required operations on the signal applied at its input.
- System or circuits can be of two types :
 1. Analog circuits
 2. Digital circuits

2.2.1 Digital Systems :

Definition :

- We define the digital system as the system (circuit) which processes or works on the digital signals.
- The input signal to a digital system is digital and its output signal is also digital.



(B-1706) Fig. 2.2.1 : Digital circuit

Examples of digital systems :

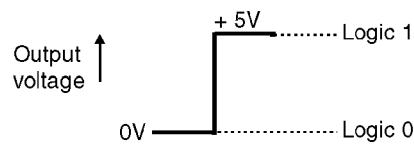
- The examples of digital circuits are adders, subtractors, registers, flip-flops, counters, microprocessors, digital calculators, computers etc.

2.3 Binary Logic and Logic Levels :

- A logic statement, is defined as a statement which is true if some condition is satisfied and false if that condition is not satisfied.
- For example, a bulb turns ON, if we close the switch, otherwise it is OFF.

2.3.1 Positive Logic :

- A "LOW" voltage level represents "logic 0" state and a comparatively "HIGH" output voltage level represents "logic 1" state, as shown in Fig. 2.3.1(a).



(B-439)Fig. 2.3.1(a) : Positive logic

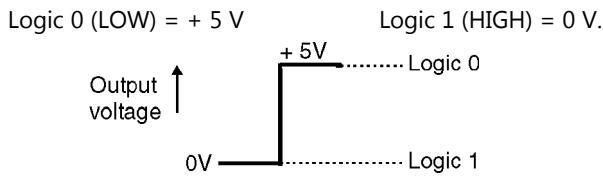
- For example, 0 Volt represent a logic 0 state and + 5 V represent logic 1. This is called as "positive logic".

Positive Logic :

Logic 0 (LOW) = 0 V, Logic 1 (HIGH) = + 5 V.

2.3.2 Negative Logic :

- A "LOW" voltage level represents "logic 1" state and a "HIGH" output voltage level represents "logic 0" state, as shown in Fig. 2.3.1(b).
- For example, 0 Volts represent a "logic 1" state and + 5 V represent "logic 0" state. This is called as "negative logic".

**Negative Logic :**

Note : In this chapter, we are going to consider only the positive logic. Also we will assume the logic 0 level corresponds to 0 Volts and logic 1 level corresponds to + 5 V.

2.4 Number Systems :

Definition :

- A number system defines a set of values used to represent a quantity.
- We talk about the number of people attending class, the number of modules taken per student and also use numbers to represent grades obtained by students in tests.
- The study of number systems is not just limited to computers.
- We apply numbers every day and knowing how numbers work will give us an insight into how a computer manipulates and stores numbers.

2.4.1 Important Definitions Related to All Numbering Systems :

- All the numbering systems have a few common elements as follows :

Radix or Base :

1. The number of values that a digit (one character) can have is equal to the base of the system. It is also called as the Radix of the system.

For example for a decimal system, the base is "10" because every digit can have 10 distinct values. (0, 1, 2, ..., 9).

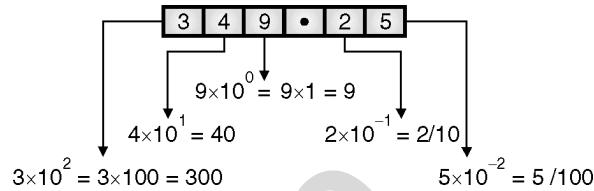
2. The largest value of a digit is always one less than the base :

For example, the largest digit in a decimal system is 9. (One less than the base 10).

2. Weight :

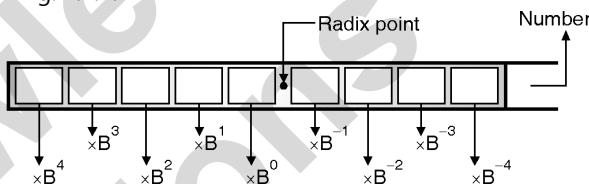
- Each place (position or column number) represents a different multiple of base or radix. These multiples are also called as weighted values or Weight.

- That means the numbers have positional importance. For example consider the decimal number (349.25)₁₀ shown in Fig. 2.4.1.



(C-5) Fig. 2.4.1 : Numbers have positional importance

- Hence we can implement a common rule for all the numbering systems as follows.
- For a general number, we have to multiply each of digit by some power of **base (B) or radix** as shown in Fig. 2.4.2.



Where B = Base or Radix

(C-6) Fig. 2.4.2

4. Column numbers :

- The column number is the number assigned to the digits placed in relation with the decimal point.
- Column numbers to the left of the decimal number start with 0 and go up (0, 1, 2, ...) as shown in Fig. 2.4.2.
- Column numbers to the right of the decimal number start from -1 and become more and more negative (-1, -2, -3, -4, ...) as shown in Fig. 2.4.2.

2.4.2 Various Numbering Systems :

- Various numbering systems used in practice and their bases are as shown in Table 2.4.1.

Table 2.4.1 : Various number systems and their bases

Name of number system	Base
Binary	2
Octal	8
Decimal	10
Duodecimal	12
Hexadecimal	16



2.5 The Decimal Number System :

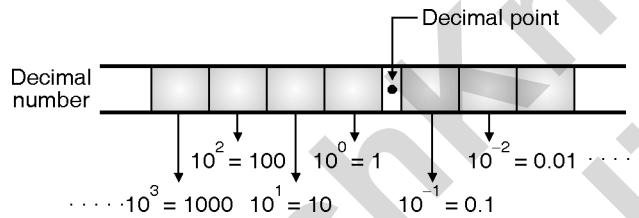
Definition :

- The number system which has a base of 10, is called as the decimal numbering system.
- We are all familiar with counting and mathematics that uses this system.
- Looking at its make up will help us to understand other numbering systems.

2.5.1 Characteristics of a Decimal System :

Some of the important characteristics of a decimal system are :

1. It uses the base of 10.
2. The largest value of a digit is 9.
3. Each place (column number) represents a different multiple of 10. These multiples are also called as **weighted values**. The weighted values of each position are as shown in Fig. 2.5.1.



(c-8) Fig. 2.5.1 : Positions and corresponding weighted values for a decimal system

Most Significant Digit (MSD) :

- The leftmost digit having the highest weight is called as the most significant digit of a number.

Least Significant Digit (LSD) :

- The rightmost digit having the lowest weight is called as the least significant digit of a number.

Ex. 2.5.1 : Represent the decimal number 532.86 in terms of powers of 10.

Soln. :

The required representation is shown in Fig. P. 2.5.1.

$$N = \boxed{5} \ \boxed{3} \ \boxed{2} \ \cdot \ \boxed{8} \ \boxed{6}$$

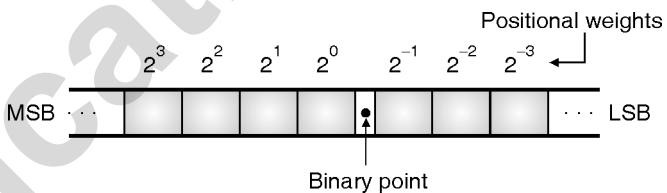
$$N = 5 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 + 8 \times 10^{-1} + 6 \times 10^{-2}$$

(c-9) Fig. P. 2.5.1

2.6 The Binary Number System :

Definition :

- A number system with a radix 2 is called as the binary number system.
- Most modern computer systems use the binary logic for their operation.
- A computer cannot operate on the decimal number system.
- A binary number system uses only two digits namely 0 and 1.
- The binary number system works like the decimal number system except one change. **It uses the base 2.**
- Hence the largest value of a digit is 1 and the number of values a digit can assume is two i.e. 0 and 1.
- The weighted values for different positions for a binary system are as shown in Fig. 2.6.1.



(c-10) Fig. 2.6.1 : Weights for different positions for a binary system

- The binary digits (0 and 1) are also called as **bits**. Thus binary system is a two bit system.
- The leftmost bit in a given binary number with the highest weight is called as **Most Significant Bit (MSB)** whereas the rightmost bit in a given number with the lowest weight is called as **Least Significant Bit (LSB)**.

Ex. 2.6.1 : Express the binary number 1011.011 in terms of powers of 2.

Soln. :

Step 1 : Express the given number in powers of 2 :

$$N = \boxed{1} \ \boxed{0} \ \boxed{1} \ \boxed{1} \ \cdot \ \boxed{0} \ \boxed{1} \ \boxed{1}$$

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

(c-11) Fig. P. 2.6.1



2.6.1 Binary Number Formats :

- We typically write binary numbers as a sequence of bits (bits is short for binary digits).
- We have defined boundaries for these bits. These boundaries are :

(C-7762) Table 2.6.1 : Binary number formats

Name	Size (bits)	Example
Bit	1	1
Nibble	4	0101
Byte	8	0000 0101
Word	16	0000 0000 0000 0101
Double word	32	0000 0000 0000 0000 0000 0000 0101

2.7 Octal Number System :

Definition :

- A number system with a radix 8 is called as the octal number system.

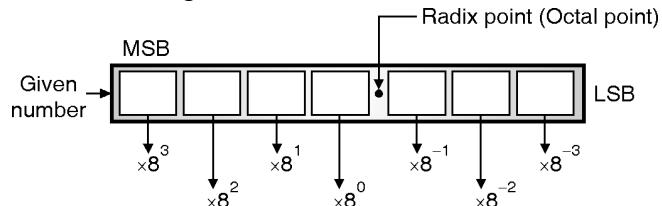
Features :

- The important features of the octal number systems are as follows :
 - Base :** The Base used for octal number system is 8.
 - The number of values assumed by each digit :**
 - The largest value of a digit :**
- The largest value of a digit in the octal system will be 7. That means the octal number higher than 7 will not be 8, instead of that it will be 10.
- Table 2.7.1 gives you a clear idea about this.

(C-7770) Table 2.7.1 : Octal numbers

0	10	20	30	40	-----	70	100
1	11	21	31	41	-----	71	101
2	12	22	32	42	-----	72	102
3	13	23	33	43	-----	73	103
4	14	24	34	44	-----	74	104
5	15	25	35	45	-----	75	105
6	16	26	36	46	-----	76	106
Largest value of a digit →	7	17	27	37	47	77	107

4. Each digit has a different multiple of base. This is as shown in Fig. 2.7.1.



(C-13) Fig. 2.7.1 : Weights for different positions for an octal system

Ex. 2.7.1 : Represent the octal number 645 in power of 8.

Soln. :

Representation in power of 8 :

$$N = \begin{array}{|c|c|c|} \hline 6 & 4 & 5 \\ \hline \end{array} \\ 6 \times 8^2 + 4 \times 8^1 + 5 \times 8^0$$

(C-14) Fig. P. 2.7.1

2.8 Hexadecimal Number System :

Definition :

- A number system with a radix 16 is called as the hexadecimal number system.

Features :

- The important features of a hexadecimal number system are as follows :
 - Base :** The base of hexadecimal system is 16.
 - Number of values assumed by each digit :**
- The number of values assumed by each digit is 16.
- The values include digits 0 through 9 and letters A, B, C, D, E, F. Hence the sixteen possible values are : 0 1 2 3 4 5 6 7 8 9 A B C D E F
- 0 represents the least significant digit whereas F represents the most significant digit.
- The base of 16 needs 16 digits. Hence it borrows 0 through 9 but needs another 6.
- For these the first six letters A, B, C, D, E, F are used. Here A represents 10, B represents 11 and so on.
- The hexadecimal digits and their values are as shown in Table 2.8.1.



(C-7772) Table 2.8.1 : Hexadecimal digits and their values

Hexadecimal digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(C-8060) Table 2.8.2 : Hexadecimal numbers

0	10	20	30	-----	90	A0	-----	F0
1	11	21	31	-----	91	A1	-----	F1
2	12	22	32	-----	92	A2	-----	F2
3	13	23	33	-----	93	A3	-----	F3
4	14	24	34	-----	94	A4	-----	F4
:	:	:	:		:	:		:
D	1D	3D	3D	-----	9D	AD	-----	FD
E	1E	2E	3E	-----	9E	AE	-----	FE
F	1F	2F	3F	-----	9F	AF	-----	FF

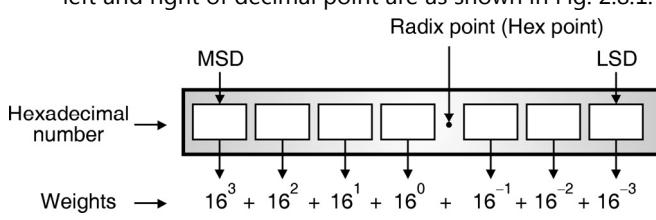
- When dealing with large values, binary numbers quickly become too unwieldy.
- The hexadecimal (base 16) numbering system solves this problem.
- Hexadecimal numbers offer the two features :
 1. Hex numbers are very compact.
 2. It is easy to convert from hex to binary and binary to hex.

Largest value of a digit :

- The largest value of a digit in the hexadecimal number system is 15 and it is represented by F.
- The hexadecimal number higher than F will be 10. Table 2.8.2 gives you a clear idea about hexadecimal numbers.
- The largest two digit hexadecimal number is FF which corresponds to 255 decimal. The next higher number after FF is 100.

Positional weights :

- The positional weights for a hexadecimal number to the left and right of decimal point are as shown in Fig. 2.8.1.



(C-15) Fig. 2.8.1 : Positional weights for a hexadecimal system

Ex. 2.8.1 : Represent the hexadecimal number 6DE in the powers of 16.

Soln. :

Representation in the powers of 16 :

$$N = \begin{array}{|c|c|c|} \hline 6 & D & E \\ \hline \end{array}$$

$$\downarrow \quad \downarrow \quad \downarrow$$

$$6 \times 16^2 + 13 \times 16^1 + 14 \times 16^0$$

(C-17) Fig. P. 2.8.1

2.9 Conversion of Number Systems :

- The conversion of a number in base "r" to decimal is done by expanding the given number in a power series and adding all the terms.
- In the subsequent sections we are going to present a general procedure for decimal to any base (radix) conversion.
- If the given number includes the radix point, then it is necessary to separate the number into an integer part and a fraction part.
- Then each part should be converted by considering separately.

2.10 Conversions Related to Decimal System :

- In this section we will perform the following conversions related to the decimal system :
 1. Decimal to other systems.
 2. Other systems to decimal.

2.10.1 Conversion from any Radix r to Decimal :

- The general procedure for conversion from binary to decimal is as given below :

Steps to be followed :

Step 1 : Note down the given number.

Step 2 : Write down the weights corresponding to different positions.

Step 3 : Multiply each digit in the given number with the corresponding weight to obtain product numbers.



Step 4 : Add all the product numbers to get the decimal equivalent.

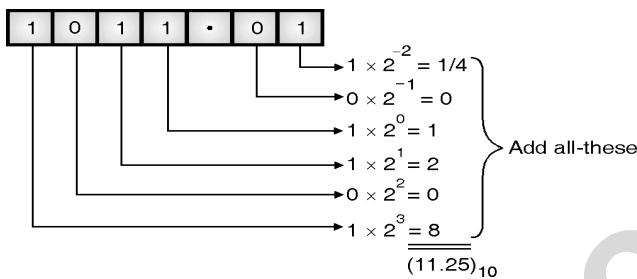
- The following example demonstrates the conversion of a binary, octal and hex number to its decimal equivalent.

Binary to decimal :

Ex. 2.10.1 : Convert the binary number 1 0 1 1 . 0 1 into its decimal equivalent.

Soln. :

Steps 1, 2 and 3 :



(C-19)

Step 4 : Addition :

$$\therefore (1011.01)_2 = (11.25)_{10} \quad \dots \text{Ans.}$$

Octal to decimal :

Ex. 2.10.2 : Convert the octal number $(314)_8$ into its decimal equivalent.

Soln. :

Step 1 : Get the octal number :

3	1	4
8^2	8^1	8^0
192	8	4

Step 2 : Write corresponding weights :

Step 3 : Multiply (columnwise) :

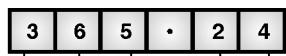
Step 4 : Add contents of row-3 : $192 + 8 + 4 = 204$

(C-6365)

$$\therefore (314)_8 = (204)_{10} \quad \dots \text{Ans.}$$

Ex. 2.10.3 : Convert the octal number $(365.24)_8$ into its equivalent decimal number.

Soln. :



Decimal number, $D = (3 \times 8^2) + (6 \times 8^1) + (5 \times 8^0) + (2 \times 8^{-1}) + (4 \times 8^{-2})$
 $= 192 + 48 + 5 + 0.25 + 0.0625 = (245.3125)_{10}$

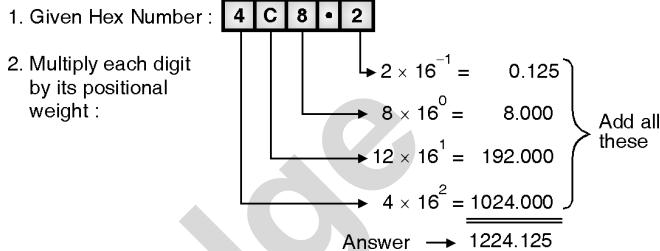
(C-20)

$$\therefore D = (245.3125)_{10}$$

Hex to decimal :

Ex. 2.10.4 : Convert the hex number $(4C8.2)_{16}$ into its equivalent decimal number.

Soln. :



$$\therefore (4C8.2)_{16} = (1224.125)_{10}$$

(C-21) Fig. P. 2.10.4

Ex. 2.10.5 : Perform the following operation :

$$(1011.101)_2 = (\underline{\hspace{2cm}})_{10}$$

Dec. 11, 8 Marks

Soln. : Solve it yourself.

Ans. :

$$(1011.101)_2 = (11.625)_{10}$$

Ex. 2.10.6 : Perform the following operations :

$$(1001.10)_2 = (\underline{\hspace{2cm}})_{10}$$

May 12, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$(1001.10)_2 = (9.5)_{10}$$

2.10.2 Conversion from Decimal to Other Systems :

- If the given decimal number consists of a decimal point, then we have to first separate out the integer and fractional parts.
- Then convert them separately to the desired radix, and combine the converted parts to obtain the complete converted number.
- The procedures for converting the integer part and the fractional part are completely different from each other.

2.10.2.1 Successive Division for Integer Part Conversion :

- Follow the procedure given below to convert the integer part of the given decimal number into any radix "r".

Steps to be followed :

Step 1 : Divide the integer part of given decimal number by the base and note down the remainder.

**Soln. :**

$$(1024)_{10} = (\underline{\hspace{2cm}})_{16}$$

(C-3205)

16	1024	0
16	64	0
16	4	4
0	0	

MSD ↑ LSD

$$\therefore (1024)_{10} = (400)_H$$

...Ans.

2.10.2.2 Successive Multiplication for Fractional Part Conversion :

- Now let us see how to convert the fractional part of a decimal number into any other radix number.
- The general procedure for such a conversion is as follows :

Steps to be followed :

- Step 1 :** Multiply the given fractional decimal number by the base (radix) r.
- Step 2 :** Note down the carry generated in this multiplication as MSD.
- Step 3 :** Multiply only the fractional number of the product in step 2 by the base, and note down the carry as the next bit to MSD.
- Step 4 :** Repeat steps 2 and 3 upto the end. The last carry will represent the LSD of equivalent i.e. converted number.

- Let us solve some examples to understand fraction conversion clearly.

Decimal to Binary :

Ex. 2.10.13 : Convert the decimal number $(0.42)_{10}$ into binary.

Soln. :

Decimal fraction	Base	Product	Carry
0.42	x 2	= 0.84	0
0.84	x 2	= 1.68	1
0.68	x 2	= 1.36	1
0.36	x 2	= 0.72	0
0.72	x 2	= 1.44	1

MSD 0 1 1 0 1 LSB

(C-27)

$$\text{So } (0.42)_{10} = (0.01101)_2$$

...Ans.

Note : We could have continued further. But the conversion is generally carried out only upto 5 digits.

Ex. 2.10.14 : Convert $(0.8)_{10}$ to equivalent binary number.

Soln. :

Decimal fraction	Base	Product	Carry
0.8	x 2	= 1.6	1
0.6	x 2	= 1.2	1
0.2	x 2	= 0.4	0
0.4	x 2	= 0.8	0
0.8	x 2	= 1.6	1

MSB 1 1 0 0 1 LSB

$\therefore (0.8)_{10} = (0.11001)_2$...Ans.

Decimal to Octal :

Ex. 2.10.15 : Convert $(0.6234)_{10}$ into its equivalent octal number.

Soln. :

Refer to Fig. P. 2.10.15 for the solution.

Decimal fraction	Base	Product	Carry
0.6234	x 8	= 4.9872	4—MSD
0.9872	x 8	= 7.8976	7
0.8976	x 8	= 7.1808	7
0.1808	x 8	= 1.4464	1
0.4464	x 8	= 3.5712	3—LSD

MSD 4 7 7 1 3 LSD

$\therefore (0.6234)_{10} = (0.47713)_8$...Ans.

(C-29) Fig. P. 2.10.15 : Conversion of fractional decimal to octal

Decimal to Hex :

Ex. 2.10.16 : Convert the decimal fraction $(0.122)_{10}$ to its equivalent hex number.

**Soln. :**

Decimal fraction	Base	Product	Carry	Hex
0.122	x 16	= 1.952	1	1
0.952	x 16	= 15.232	15	F
0.232	x 16	= 3.712	3	3
0.712	x 16	= 11.392	11	B
0.392	x 16	= 6.272	6	6
0.272	x 16	= 4.352	4	4

(C-30)

$$\therefore (0.122)_{10} = (0.1F3B64)_{16}$$

...Ans.

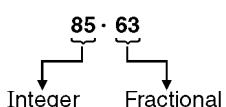
2.10.2.3 Conversion of Mixed Decimal Number to Any Other Radix :

- We have seen the conversion of the integer and fractional parts of a decimal number into desired radix number separately.
- Now let us see the conversion of mixed decimal number containing the integer and fractional parts into any other radix number.
- Follow the procedure given below for such a conversion.

Steps to be followed :

- Step 1 :** Separate the integer and fractional parts of the given decimal number.
- Step 2 :** Convert the integer part into desired radix.
- Step 3 :** Convert the fractional part into desired radix.
- Step 4 :** Combine the results of steps 2 and 3 to get the final answer.

Ex. 2.10.17 : Convert $(85.63)_{10}$ into its equivalent binary number.

Soln. :**Step 1 : Separate integer and fractional parts :**

(C-31)

Step 2 : Convert the integer :

2	85	
2	42	1
2	21	0
2	10	1
2	5	0
2	2	1
2	1	0
	0	1

LSB

MSB

$$\therefore (85)_{10} = (1010101)_2$$

Step 3 : Convert the fractional part :

0.63 × 2 = 1.26	1	MSB
0.26 × 2 = 0.52	0	
0.52 × 2 = 1.04	1	
0.04 × 2 = 0.08	0	
0.08 × 2 = 0.16	0	LSB

$$\therefore (0.63)_{10} = (0.10100)_2$$

(C-32)

Step 4 : Combine the results of steps 2 and 3 :

$$\therefore (85.63)_{10} = (1010101.10100)_2$$

...Ans.

Decimal to Octal :

Ex. 2.10.18 : Convert $(3000.45)_{10}$ into its equivalent octal number.

Soln. :**Step 1 : Separate the integer and fractional parts :**

Integer part = 3000 and fractional part = 0.45.

Step 2 : Convert the integer part :**Step 3 : Convert the fraction part :**

8	3000	
8	375	0
8	46	7
8	5	6
0	5	5

Column of remainders

Column of quotients

$$\therefore (3000)_{10} = (5670)_8$$

Base	Product	Carry	MSD
0.45	$0.45 \times 8 = 3.60$	3	
0.60	$0.60 \times 8 = 4.80$	4	
0.80	$0.80 \times 8 = 6.40$	6	
0.40	$0.40 \times 8 = 3.20$	3	LSD

$$\therefore (0.45)_{10} = (0.3463)_8$$

(C-33) Fig. P. 2.10.18

Step 4 : Combine the results of steps 2 and 3 :

Combining the results of steps 2 and 3 we get the answer.

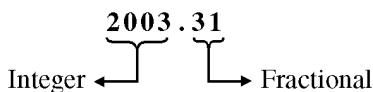
$$\therefore (3000.45)_{10} = (5670.3463)_8$$

**Decimal to Hex :**

Ex. 2.10.19 : Convert $(2003.31)_{10}$ into its equivalent hex number.

Soln. :

Step 1 : Separate the integer and fractional parts : (C-34)



Step 2 : Convert integer part :

16	2003	Hex	3 → 3 D → D 7 → 7
16	125	3	
16	7	D	
0	7	7	

LSD ↑
MSD ↓

$\therefore (2003)_{10} = (7D3)_{16}$ (C-35)

Step 3 : Convert the fractional part into hex :

Decimal fraction	\times	Base = 16	Product	Carry Decimal	Carry Hex	MSD
0.31	\times	16	= 4.96	4	4	
0.96	\times	16	= 15.36	15	F	
0.36	\times	16	= 5.76	5	5	
0.76	\times	16	= 12.16	12	C	
0.16	\times	16	= 2.56	2	2	LSD

(C-36)

$$\therefore (0.31)_{10} = (0.4F5C2)_{16}$$

Step 4 : Combine the results of steps 2 and 3 :

Combining the results of steps 2 and 3 we get

$$(2003.31)_{10} = (7D3.4F5C2)_{16} \quad \dots \text{Ans.}$$

Ex. 2.10.20 : Convert the following numbers into equivalent decimal numbers :

1. $(327.4051)_8$
2. $(5A.FF)_{16}$
3. $(101110111)_2$
4. $(3FFF)_{16}$.

Dec. 12, 8 Marks

Soln. : Solve it yourself.

Ans. :

1. $(327.4051)_8 = (215.50994)_{10}$
2. $(5A.FF)_{16} = (90.9960)_{10}$
3. $(101110111)_2 = (375)_{10}$
4. $(3FFF)_{16} = (16383)_{10}$

2.11 Conversion from Binary to Other Systems :

2.11.1 Conversion from Binary to Decimal :

We have already discussed this.

2.11.2 Binary to Octal Conversion :

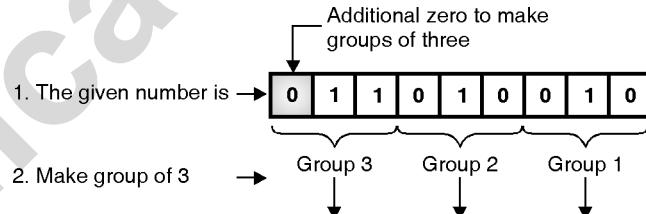
- For converting the given binary number into an equivalent octal number, follow the procedure given below :

Step 1 : Divide the binary bits into groups of 3 starting from the LSB.

Step 2 : Convert each group into its equivalent decimal. As the number of bits in each group is restricted to 3, the decimal number will be same as octal number.

Ex. 2.11.1 : Convert the binary number $(11010010)_2$ into its equivalent octal number.

Soln. :



$$\therefore (11010010)_2 = (322)_8$$

(C-37)

Note : In the third group, there are only 2 bits. Hence we have assumed the number to be 011 instead of 11. Always add the extra zeros on the MSB side, not on LSB side.

Ex. 2.11.2 : Convert the following binary numbers to octal then to decimal. Show the steps of conversions.

1. 11011100.101010
2. 01010011.010101
3. 10110011

Dec. 17, 6 Marks

Soln. :

$$1. (11011100.101010)_2 = (?)_8 = (?)_{10}$$

**Step 1 : Convert binary to octal : (C-6323)**

The given number : Additional zero to make groups of three
 Convert into octal : 3 3 4 . 5 2
 $\therefore (11011100.101010)_2 = (334.52)_8 \quad \dots\text{Ans.}$

Step 2 : Convert octal to decimal : (C-6324)

Octal number :
 Decimal number (N) : $(3 \times 8^2) + (3 \times 8^1) + (4 \times 8^0) \cdot (5 \times 8^{-1}) + (2 \times 8^{-2})$
 $\therefore N = 192 + 24 + 4 + 0.625 + 0.03125 = 220.65625$
 $\therefore (334.52)_8 = (220.65625)_{10} \quad \dots\text{Ans.}$

2. $(01010011.010101)_2 = (?)_8 = (?)_{10}$

Step 1 : Convert binary to octal : (C-6325)

The given number : Additional zero to make groups of three
 Convert into octal : 1 2 3 . 2 5
 $\therefore (01010011.010101)_2 = (123.25)_8 \quad \dots\text{Ans.}$

Step 2 : Convert octal to decimal : (C-6326)

Octal number :
 Decimal number (N) : $(1 \times 8^2) + (2 \times 8^1) + (3 \times 8^0) \cdot (2 \times 8^{-1}) + (5 \times 8^{-2})$
 $\therefore N = 64 + 16 + 3 + 0.25 + 0.078125$
 $= 83.328125$
 $\therefore (123.25)_8 = (83.328125)_{10} \quad \dots\text{Ans.}$

3. $(10110011)_2 = (?)_8 = (?)_{10}$

Step 1 : Convert binary to octal : (C-6327)

Given number : Additional zero to make groups of three
 Convert into octal : 2 6 3
 $\therefore (10110011)_2 = (263)_8 \quad \dots\text{Ans.}$

Step 2 : Convert octal into decimal : (C-6328)

Octal number :
 Decimal number (N) : $(2 \times 8^2) + (6 \times 8^1) + (3 \times 8^0)$
 $\therefore N = 128 + 48 + 3 = 179$
 $\therefore (263)_8 = (179)_{10} \quad \dots\text{Ans.}$

2.11.3 Binary to Hex Conversion :

- It is easy to convert from an integer binary number to hex. This is accomplished by :

Step 1 : Divide the binary number into 4-bit sections from the LSB to the MSB.

Step 2 : Convert each 4-bit binary number to its hex equivalent.

Ex. 2.11.3 : Convert the binary number 1010 1111 1011 0010 into the equivalent hex number.

Soln. :

1. Split the given number into groups of bits.

1010	1111	1011	0010
------	------	------	------

2. Convert each group into corresponding hex.

A	F	B	2
---	---	---	---

(C-6333)

$$\therefore (1010 1111 1011 0010)_2 = (\text{AFB2})_{16} \quad \dots\text{Ans.}$$

2.12 Conversion from Other Systems to Binary System :**2.12.1 Conversion from Decimal to Binary :**

- We have already discussed this earlier.

Ex. 2.12.1 : Express the following numbers in binary. Show your step by step equations and calculations.

1. $(1010.11)_{\text{Decimal}}$
2. $(428.10)_{\text{Decimal}}$

Dec. 09, 6 Marks

Soln. : Solve it yourself.

Ans. :

1. $(1010.11)_{10} = (111110010.0001)_2$
2. $(428.10)_{10} = (110101100.0001)_2$

Ex. 2.12.2 : Express the following numbers in binary, show the step-by-step equations and calculations :

1. $(110.110)_{\text{Decimal}}$
2. $(234.234)_{\text{Decimal}}$

May 10, 6 Marks

Soln. : Solve it yourself.

Ans. :

1. $(110.110)_{10} = (1101110.0001)_2$
2. $(234.234)_{10} = (11101010.0011)_2$

2.12.2 Octal to Binary Conversion :

- To get the binary equivalent of the given octal number we have to convert each octal digit into its equivalent 3-bit binary number.
- This is as explained in the following example.



Ex. 2.12.3 : Convert the octal number $(364)_8$ into equivalent binary number.

Soln. :

Step 1 : Given octal number :

3	6	4
011	110	100

Step 2 : Convert each digit to binary :

(C-6366)

$$\therefore (364)_8 = (011110100)_2 \quad \dots \text{Ans.}$$

Ex. 2.12.4 : Convert $(364.25)_8$ into its equivalent binary number.

Soln. :

- Follow the same procedure explained in the previous example.

Step 1 : Given octal number :

3	6	4	.	2	5
---	---	---	---	---	---

Step 2 : Convert each digit into binary :

011	110	100	.	010	101
-----	-----	-----	---	-----	-----

(C-6334)

$$\therefore (364.25)_8 = (011110100.010101)_2 \quad \dots \text{Ans.}$$

2.12.3 Hex to Binary Conversion :

- It is also easy to convert from an integer hex number to binary. This is accomplished by :

Step 1 : Convert each hex digit to its 4-bit binary equivalent.

Step 2 : Combine the 4-bit sections by removing the spaces.

Ex. 2.12.5 : Convert the hex number AFB2 into equivalent binary number.

Soln. :

- Each digit in the given hex number is converted into 4-bit binary numbers as shown in Fig. P. 2.12.5.

Given hex number	A	F	B	2
Each digit converted to its four bit binary equivalent	1010	1111	1011	0010

(C-38) Fig. P. 2.12.5 : Hex to binary conversion

$$\text{Hence } (A F B 2)_{16} = (1010 1111 1011 0010)_2 \quad \dots \text{Ans.}$$

Ex. 2.12.6 : Convert the following numbers in Binary form :

- $(125.12)_{10} = (?)_2$
- $(337.025)_8 = (?)_2$
- $(5DB.FA)_{16} = (?)_2$

Dec. 18, 6 Marks

Soln. :

1. Decimal to binary :

Step 1 : Convert the integer :

2	125	
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
1	1	

$$\therefore (125)_{10} = (111101)_2$$

(C-7434) Fig. P. 2.12.6(a)

Step 2 : Convert the fractional part :

0.12	$\times 2 = 0.24$	0	MSB
0.24	$\times 2 = 0.48$	0	
0.48	$\times 2 = 0.96$	0	
0.96	$\times 2 = 1.92$	1	
0.92	$\times 2 = 1.84$	1	LSB

$$\therefore (0.12)_{10} = (0.00011)_2$$

(C-7435) Fig. P. 2.12.6(b)

$$\therefore (125.12)_{10} = (111101.00011)_2 \quad \dots \text{Ans.}$$

2. Octal to binary :

Given number :	3	3	7	.	0	2	5
Binary :	011	011	111	.	000	010	101

(C-7436)

$$\therefore (337.025)_8 = (01101111.000010101)_2 \quad \dots \text{Ans.}$$

3. Hex to binary :

Hex :	5	D	B	.	F	A
Binary :	0101	1101	1011	.	1111	1010

(C-7437)

$$\therefore (5DB)_{16} = (010111011011.11111010)_2 \quad \dots \text{Ans.}$$

Ex. 2.12.7 : Express the following numbers in binary format. Write step by step solution.

- $(7762)_{\text{octal}} = (?)_2$
- $(432A)_{\text{hex}} = (?)_2$
- $(2946)_{\text{decimal}} = (?)_2$
- $(1101.11)_{\text{decimal}} = (?)_2$

Dec. 10, 12 Marks



Soln. : Solve it yourself.

Ans. :

1. $(7762)_{\text{octal}} = (111111110010)_{\text{binary}}$
2. $\therefore (432A)_{\text{hex}} = (0100001100101010)_{\text{binary}}$
3. $\therefore (2946)_{10} = (101110000010)_2$
4. $(1101.11)_{10} = (10001001101.0001)_2$

2.13 Conversion from Octal to Other Systems :

- We have already discussed the following two conversions :
 1. Octal to decimal
 2. Octal to binary.

2.13.1 Octal to Hex Conversion :

- For converting octal to hex, follow the steps given below :

Step 1 : Convert the given octal number into equivalent binary.

Step 2 : Then convert this binary number into hex.

- The octal to hex conversion is demonstrated in Fig. 2.13.1.

Given octal number = $(436)_8$

Step 1 : Convert octal to binary :

Given octal number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>4</td><td>3</td><td>6</td></tr></table>	4	3	6
4	3	6		
Binary equivalent	<table border="0" style="margin-left: 10px; vertical-align: middle;"><tr><td>1 0 0</td><td>0 1 1</td><td>1 1 0</td></tr></table>	1 0 0	0 1 1	1 1 0
1 0 0	0 1 1	1 1 0		

Step 2 : Convert binary to hex :

Binary $\rightarrow (10001110)_2$

Add three zeros on extreme left (on MSB side) to get, $(00010001110)_2$

Binary number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0 0 0 1</td><td>0 0 0 1</td><td>1 1 1 0</td></tr></table>	0 0 0 1	0 0 0 1	1 1 1 0	Groups of 4 bits
0 0 0 1	0 0 0 1	1 1 1 0			
Hex number :	<table border="0" style="margin-left: 10px; vertical-align: middle;"><tr><td>1</td><td>1</td><td>E</td></tr></table>	1	1	E	
1	1	E			

$$\therefore (436)_8 = (11E)_{16} \quad \dots \text{Ans.}$$

(C-6336) Fig. 2.13.1

Ex. 2.13.2 : For a maximum 3-digit octal number obtain equivalent hex, binary, decimal number.

May 11, 3 Marks

Soln. :

1. Octal to binary :

Largest 3-digit octal number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>7</td><td>7</td><td>7</td></tr></table>	7	7	7	...Ans.
7	7	7			
Equivalent binary	<table border="0" style="margin-left: 10px; vertical-align: middle;"><tr><td>1 1 1</td><td>1 1 1</td><td>1 1 1</td></tr></table>	1 1 1	1 1 1	1 1 1	...Ans.
1 1 1	1 1 1	1 1 1			

(C-6337)

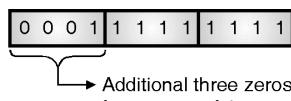
2. Octal to hex :

$$\therefore (777)_8 = (11111111)_8$$

...Ans.

- For equivalent hex group the binary number into group of 4.

i.e.



(C-1948)

$$\therefore \text{Equivalent hex is } 1FF.$$

3. Octal to decimal :

$$\therefore (777)_8 = (1FF)_{16}$$

...Ans.



(C-1949)

$$\begin{aligned} \text{Decimal number : } & 7 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 \\ & = 448 + 56 + 7 = 511 \end{aligned}$$

$$\therefore (777)_{\text{octal}} \rightarrow (1FF)_{\text{Hex}} \rightarrow (11111111)_{\text{Binary}} \rightarrow (511)_{\text{Decimal}}$$

...Ans.

Ex. 2.13.3 : Do the required conversions for the following numbers :

$$(377)_8 = (\underline{\hspace{2cm}})_{16}$$

Dec. 11, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$(377)_8 = (FF)_{16}$$

Ex. 2.13.4 : Do the required conversion for the following number :

$$(36)_8 = (\underline{\hspace{2cm}})_{16}$$

May 12, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$(36)_8 = (1E)_{16}$$

2.13.2 Conversion from Other Systems to Octal :

- We have already discussed the following two conversions :

1. Decimal to octal
2. Binary to octal.

Hex to octal conversion :

- For the hex to octal conversion follow the steps given below :

Step 1 : Represent each hex digit by a 4-bit binary number.

Step 2 : Combine these 4-bit binary sections by removing the spaces.



Step 3 : Now group these binary bits into groups of 3 bits, starting from the LSB side.

Step 4 : Then convert each of this 3-bit group into an octal digit.

Ex. 2.13.5 : Convert the hex number 4CA into its equivalent octal form.

Soln. :

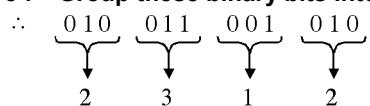
Step 1 : Convert $(4CA)_{16}$ into binary :

4	C	A
0100	1100	1010

Step 2 : Combine the 4 bit binary sections by removing spaces :

$$\therefore (4CA)_{16} = (0100\ 1100\ 1010)_2$$

Step 3 : Group these binary bits into groups of 3 bits :



Each 3 bit group is converted to an octal digit. (C-39)

Step 4 :

$$\therefore (4CA)_{16} = (2\ 3\ 1\ 2)_8 \quad \dots \text{Ans.}$$

Fractional hex to octal conversion :

- To convert the fractional hex number into octal we use the following steps :

Step 1 : Convert the given fractional hex number into its equivalent binary number.

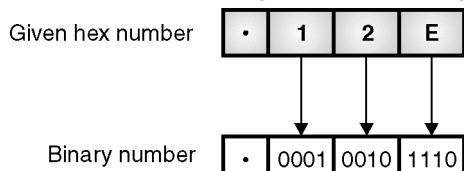
Step 2 : Group the binary bits into groups of 3 bits.

Step 3 : Convert each group of 3 bits into an octal digit.

Ex. 2.13.6 : Convert $(0.12E)_{16}$ into equivalent octal number.

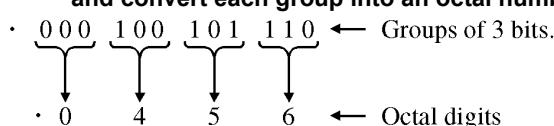
Soln. :

Step 1 : Convert each hex digit into 4-bit binary word :



(C-40)

Step 2, 3 : Group the binary bits into groups of 3 bits and convert each group into an octal number :



(C-41)

$$\therefore (0.12E)_{16} = (0.0456)_8$$

...Ans.

Note : The bits are grouped starting from the fractional point and moving towards right.

Conversion of mixed hex number to octal :

- The procedure to be followed for conversion of mixed hex number is same as the one discussed for fractional hex conversion.

Ex. 2.13.7 : Convert the hex number $(68.4B)_{16}$ into equivalent octal number.

Soln. :

1. Given hex number :	6	8	.	4	B
2. Convert to binary :	0110	1000	.	0100	1011

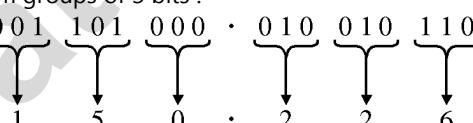
(C-6368)

- Now put additional zeros to extreme left and right :



(C-42)

- Form groups of 3 bits :



(C-43)

- Convert into octal :

$$\therefore (68.4B)_{16} = (150.226)_8$$

...Ans.

Ex. 2.13.8 : Convert the following octal numbers into its equivalent decimal and hex :

$$1. (555)_{\text{octal}}$$

$$2. (777)_{\text{octal}}$$

May 10, 6 Marks

Soln. : Solve it yourself.

Ans. :

$$1. (555)_{\text{octal}} = (365)_{10} = (16d)_{\text{H}}$$

$$2. (777)_{\text{octal}} = (511)_{10} = (1FF)_{\text{H}}$$

Ex. 2.13.9 : Convert the following numbers to octal form. Show the steps of conversion :

$$1. (111110001.10011001101)_2$$

$$2. (3287.51)_{10}$$

$$3. (0.BF85)_{16}$$

$$4. (1234)_{16}$$

Dec. 12, 8 Marks

Soln. : Solve it yourself.

**Ans. :**

1. $(111110001.10011001101)_2 = (761.4632)_8$...Ans.
 2. $(3287.51)_{10} = (6327.4050)_8$...Ans.
 3. $(0.BF85)_{16} = (0000.101111110000101)_2$...Ans.
 4. $(1234)_{16} = (11064)_8$...Ans.

2.14 Conversions Related to Hexadecimal System :

2.14.1 Other Systems to Hex :

- We are supposed to discuss the following conversions :
 1. Decimal to Hex
 2. Binary to Hex 3. Octal to Hex
- We have already discussed them.

Ex. 2.14.1 : Convert the following number into its equivalent hexadecimal, decimal and binary number (show step-by-step process of conversion) :

1. $(357.2)_8$ 2. $(453.54)_8$

May 14, 6 Marks

Soln. :

1. $(357.2)_8 = (?)_{16} = (?)_{10} = (?)_2$:

Step 1 : Convert octal to binary :

Given octal number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>3</td><td>5</td><td>7</td><td>.</td><td>2</td></tr></table>	3	5	7	.	2
3	5	7	.	2		
Binary equivalent :	<table border="0" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>101</td><td>111</td><td>.</td><td>010</td></tr></table>	011	101	111	.	010
011	101	111	.	010		

(C-6338)

$$\therefore (357.2)_8 = (011101111.010)_2 \quad \dots\text{Ans.}$$

Step 2 : Convert octal to hex :

$$\text{Binary} = (011101111.010)_2$$

Add one zero on extreme right side to get $(11101111.0100)_2$

Binary number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>.</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	1	1	1	0	1	1	1	.	0	1	0	0	Groups of 4 bits
1	1	1	0	1	1	1	.	0	1	0	0			
	<table border="0" style="display: inline-table; vertical-align: middle;"><tr><td>E</td><td>F</td><td>.</td><td>4</td></tr></table>	E	F	.	4									
E	F	.	4											

(C-6339)

$$\therefore (357.2)_8 = (EF.4)_{16} \quad \dots\text{Ans.}$$

Step 3 : Convert octal to decimal :

Octal number	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>3</td><td>5</td><td>7</td><td>.</td><td>2</td></tr></table>	3	5	7	.	2
3	5	7	.	2		
Decimal number :	$(3 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1})$ (C-4807)					

$$\begin{aligned}
 &= 192 + 40 + 7 + 0.25 \\
 &= 239 + 0.25 = 239.25 \\
 \therefore (357.2)_8 &= (239.25)_{10} \quad \dots\text{Ans.}
 \end{aligned}$$

$$2. (453.54)_8 = (?)_{16} = (?)_{10} = (?)_2 :$$

Step 1 : Convert octal to binary :

Given octal number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>4</td><td>5</td><td>3</td><td>.</td><td>5</td><td>4</td></tr></table>	4	5	3	.	5	4
4	5	3	.	5	4		
Binary equivalent :	<table border="0" style="display: inline-table; vertical-align: middle;"><tr><td>100</td><td>101</td><td>011</td><td>.</td><td>101</td><td>100</td></tr></table>	100	101	011	.	101	100
100	101	011	.	101	100		

(C-6340)

$$\therefore (453.54)_8 = (100101011.101100)_2 \quad \dots\text{Ans.}$$

Step 2 : Convert octal to hex :

$$\text{Binary} = (100101011.101100)_2$$

Add three zeros on extreme left to get $(000100101011.1011)_2$

Binary number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>.</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	0	1	0	1	1	.	1	0	1	1
0	0	0	1	0	0	1	0	1	1	.	1	0	1	1		
Hex number :	<table border="0" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>B</td><td>.</td><td>B</td></tr></table>	1	2	B	.	B										
1	2	B	.	B												

(C-6341)

$$\therefore (453.54)_8 = (12B.B)_{16} \quad \dots\text{Ans.}$$

Step 3 : Convert octal to decimal :

Octal number :	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>4</td><td>5</td><td>3</td><td>.</td><td>5</td><td>4</td></tr></table>	4	5	3	.	5	4
4	5	3	.	5	4		
	$(4 \times 8^2) + (5 \times 8^1) + (3 \times 8^0) + (5 \times 8^{-1}) + (4 \times 8^{-2})$ (C-806)						

$$= 256 + 40 + 3 + 0.625 + 0.0625$$

$$= 299.6875$$

$$\therefore (453.54)_8 = (299.6875)_{10} \quad \dots\text{Ans.}$$

Ex. 2.14.2 : Do the following :

$$1. (735.25)_{10} = (?)_{16}$$

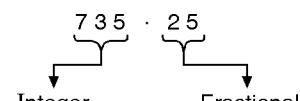
$$2. (101011.111011)_2 = (?)_8 (?)_{16}$$

May 17, 2 Marks

Soln. :

$$1. (735.25)_{10} = (?)_{16} :$$

Step 1 : Separate integer and fractional part :



(C-5902)

**Step 2 : Convert the integer part :**

Hex			
16	735	15	→ F LSD
16	45	2	→ D
16	2	2	→ 2 MSD
	0		

$$\therefore (735)_{10} = (2DF)_{16}$$

(C-5903)

Step 3 : Convert the fractional part :

Decimal	Base	Product	Carry
0.25	16	4.0	4 MSD
0.0	16	0.0	0 LSD

$$\therefore (0.25)_{10} = (0.40)_{16}$$

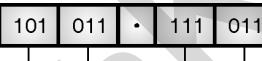
(C-5904)

Step 4 : Combine results of steps 2 and 3 :

$$(735.25)_{10} = (2DF.40)_{16}$$

...Ans.

2. $(101011.111011)_2 = (?)_8 (?)_{16}$ **Step 1 : Convert binary to octal :**

Make a group of 3 bits : 

Octal number : 5 3 . 7 3

(C-5907)

$$(101011.111011)_2 = (53.73)_8$$

...Ans.

Step 2 : Convert binary to hexadecimal : (C-5908)

Make a group of 4 bits : 

Hex number : 2 B . E C

(C-5706)

$$\therefore (101011.111011)_2 = (2B.EC)_{16}$$

...Ans.

Ex. 2.14.3 : Convert the following octal number into its equivalent Binary, Decimal and Hexadecimal
 $(357.3)_8$. **Dec. 19, 6 Marks**

Soln. :

$$(357.3)_8 = (?)_{16} = (?)_{10} = (?)_2$$

Step 1 : Convert octal to binary : (C-6338(a))

Given octal number : 

Binary equivalent : 011 101 111 . 011

$$\therefore (357.3)_8 = (011101111.011)_2$$

...Ans.

Step 2 : Convert octal to hex :

$$\text{Binary} = (011101111.011)_2$$

Add one zero on extreme right side to get
 (11101111.0110)

Binary number : 

Groups of 4 bits

↓ ↓ ↓ ↓

E F . 6

(C-6339(a))

$$\therefore (357.3)_8 = (EF.6)_{16}$$

...Ans.

Step 3 : Convert octal to decimal :

Octal number : 

↓ ↓ ↓ ↓ ↓ ↓

Decimal number : $(3 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (3 \times 8^{-1})$

(C-4807(a))

$$= 192 + 40 + 7 + 0.375$$

$$= 239.375$$

$$\therefore (357.3)_8 = (239.375)_{10}$$

...Ans.

Ex. 2.14.4 : Convert the following numbers to hexadecimal form. Show the steps of conversion :

$$1. (675.625)_{10} \quad 2. (451)_8$$

$$3. (95.5)_{10} \quad 4. (11001011101)_2$$

Dec. 12, 8 Marks**Soln. : Solve it yourself.****Ans. :**

$$1. \therefore (675.625)_{10} = (2A3.A0)_{16}$$

$$2. \therefore (451)_8 = (129)_{16}$$

$$3. \therefore (95.5)_{10} = (5F.80)_{16}$$

$$4. \therefore (11001011101)_2 = (65D)_{16}$$

Ex. 2.14.5 : Convert the following decimal numbers into its equivalent binary, hexadecimal and octal numbers : 1. 456 2. 25.55.

May 07, 6 Marks**Soln. : Solve it yourself.****Ans. :**

$$1. (456)_{10} = (111001000)_2 = (710)_8 = (1C8)_{16}$$

$$2. (25.55)_{10} = (11001 \cdot 10001)_2 = (31.431)_8 \\ = (19.8CC)_{16}$$



Ex. 2.14.6 : Convert the following octal numbers into its equivalent Hexadecimal, Binary and Decimal numbers :
 1. $(76)_8$ 2. $(0.7634)_8$ 3. $(1567)_8$
 4. $(65.04)_8$ **Dec. 08, 12 Marks**

Soln. : Solve it yourself.

Ans. :

1. $(76)_8 = (111110)_2 = (3E)_{16} = 56 + 6 = (62)_{10}$
2. $(0.7634)_8 = (0.111110011100)_2 = (0.F9C)_{16} = (0.9755)_{10}$
3. $(1567)_8 = (001101110111)_2 = (377)_{16} = (887)_{10}$
4. $(65.04)_8 = (110101.000100)_2 = (35.10)_{16} = 53.0625$

2.14.2 Hex to Other Systems :

- We are supposed to discuss the following conversions :
 1. Hex to Decimal.
 2. Hex to Binary
 3. Hex to Octal.
- We have already discussed them.

Ex. 2.14.7 : What is the maximum equivalent decimal number represented by its maximum equivalent 4-digit Hex number ? Also convert the following Hex numbers to get its equivalent octal and decimal :

1. $(ABC)_{Hex}$.
2. $(DEF)_{Hex}$.

Dec. 09, 8 Marks

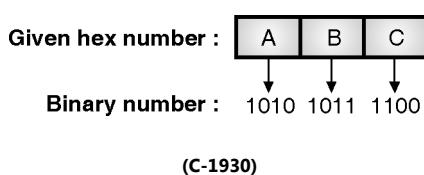
Soln. :

The maximum equivalent decimal number represented by its maximum equivalent 4 digit hex number i.e. $(FFFF)_{16}$ is $(65535)_{10}$.

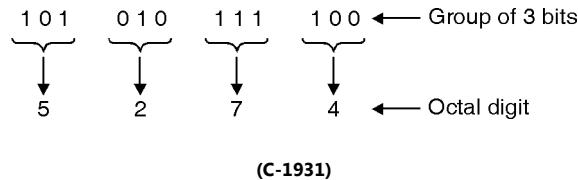
1. $(ABC)_{Hex}$:

Conversion of hex to octal :

Step 1 : Convert each hex digit into 4-bit binary word :

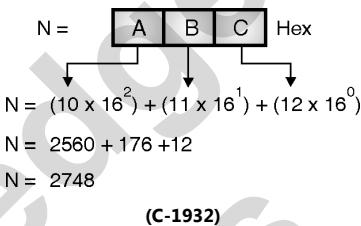


Step 2 : Group these binary bits into groups of 3 bits :



$$\therefore (ABC)_{16} = (5274)_8 \quad \dots \text{Ans.}$$

Conversion of hex to decimal :

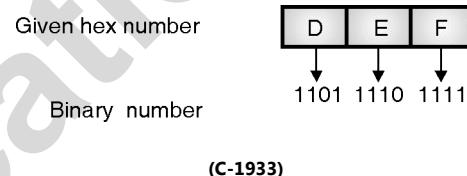


$$\therefore (ABC)_{16} = (2748)_{10} \quad \dots \text{Ans.}$$

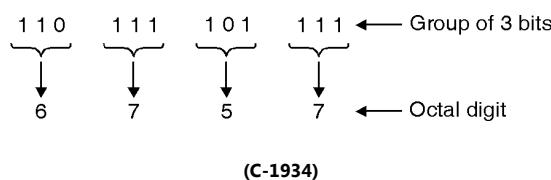
2. $(DEF)_{Hex}$:

Conversion of hex to octal :

Step 1 : Convert each hex digit into 4-bit binary word :

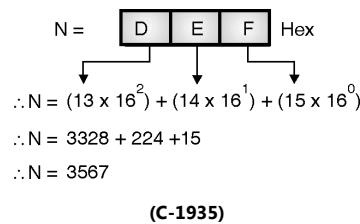


Step 2 : Group these binary bits into groups of 3 bits :



$$\therefore (DEF)_{16} = (6757)_8 \quad \dots \text{Ans.}$$

Conversion of hex to decimal :



$$\therefore (DEF)_{16} = (3567)_{10} \quad \dots \text{Ans.}$$

Ex. 2.14.8 : Convert the following numbers, show all the steps :

1. $(101101.10101)_2 = ()_{10}$
2. $(247)_{10} = ()_8$
3. $(0.BF85)_{16} = ()_8$

Dec. 14, 6 Marks

**Soln. :**

$$1. \quad (101101.10101)_2 = (?)_{10}$$

Convert binary to decimal :

$N = 1 \times 2^5 + 0 + 1 \times 2^3 + 1 \times 2^2 + 0 + 1 \times 2^0 \cdot 1 \times 2^{-1} + 0 + 1 \times 2^{-3} + 0 + 1 \times 2^{-5}$

(C-4936)

$$N = 32 + 8 + 4 + 1 + 0.5 + 0.125 + 0.03125$$

$$N = 45.65625$$

$$\therefore (101101.10101)_2 = (45.65625)_{10}$$

...Ans.

$$2. \quad (247)_{10} = (?)_8 :$$

Convert decimal to octal :

(C-4937)

$$\therefore (247)_{10} = (367)_8$$

...Ans.

$$3. \quad (0.BF85)_{16} = (?)_8 :$$

Convert hex to octal :

1. Given hex number :

0	.	B	F	8	5
0000	.	1011	1111	1000	0101

(C-6342)

3. Form groups of 3 bits :

0	0	0	·	0	0	1	0	1	1	1	1	1	0	0	0	1	0	1	0
0	·	1		3		7		6		0		5							

(C-4938)

$$\therefore (0.BF85) = (0.137605)_8$$

...Ans.**Ex. 2.14.9 :** Convert the following numbers, show all steps :

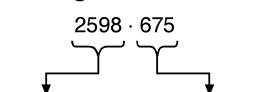
$$1. \quad (2598.675)_{10} = (?)_{16}$$

$$2. \quad (110101.101010)_2 = (?)_8$$

$$3. \quad (A72E)_{16} = (?)_8 \quad \boxed{\text{Dec. 15, 6 Marks}}$$

Soln. :

$$1. \quad (2598.675)_{10} = (?)_{16} :$$

Step 1 : Separate integer and fractional parts :**(C-5112)****Step 2 : Convert the integer part :**

16	2598	6	→ 6	LSD
16	162	2	→ 2	
16	10	10	→ A	MSD

(C-5113)

$$\therefore (2598)_{10} = (A26)_{16}$$

Step 3 : Convert the fractional part :

Base	Product	Carry	Hex	
0.675 × 16	= 10.8	10	→ A	MSD
0.8 × 16	= 12.8	12	→ C	
0.8 × 16	= 12.8	12	→ C	
0.8 × 16	= 12.8	12	→ C	LSD

(C-5114)

$$\therefore (0.675)_{10} = (0.ACCC)_{16}$$

Step 4 : Combine results of steps 2 and 3 :

$$(2598.675)_{10} = (A26.ACCC)_{16}$$

...Ans.

$$2. \quad (110101.101010)_2 = (?)_8 :$$

1. The given binary number :
2. Make group : Group 4 Group 3 Group 2 Group 1
3. Convert into : 6 5 5 2 octal

(C-5115)

$$\therefore (110101.101010)_2 = (65.52)_8$$

...Ans.

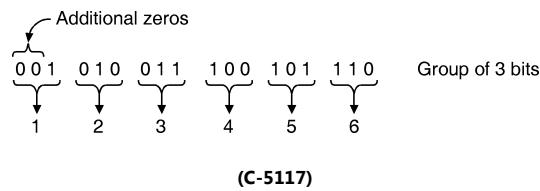
$$3. \quad (A72E)_{16} = (?)_8 :$$

Step 1 : Convert hex to binary :

Given hex number :	A	7	2	E
Binary :	1010	0111	0010	1110

(C-5116)**Step 2 : Combine the 4 bit binary bits into groups of 3 bits :**

$$\therefore (A72E)_{16} = (1010011100101110)_2$$

Step 3 : Group these binary bits into groups of 3 bits :**(C-5117)**

$$\therefore (A72E)_{16} = (123456)_8$$

...Ans.



Ex. 2.14.10 : Do the following conversions :

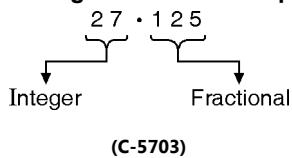
1. $(27.125)_{10} = (?)_2$
2. $(3A.2F)_{16} = (?)_{10}$
3. $(1101.0011)_2 = (?)_{10}$

Dec. 16, 6 Marks

Soln. :

$$1. \quad (27.125)_{10} = (?)_2 :$$

Step 1 : Separate integer and fractional part :



Step 2 : Convert the integer :

2	27	1	LSB
2	13	1	
2	6	0	
2	3	1	
2	1	1	MSB
	0		

$$\therefore (27)_{10} = (11011)_2$$

(C-5115)

Step 3 : Convert the fractional part :

Decimal fraction	Base	Product	Carry	
0.125	2	= 0.25	0	MSB
0.25	2	= 0.5	0	
0.5	2	= 1.0	1	
0.0	2	= 0.0	0	LSB

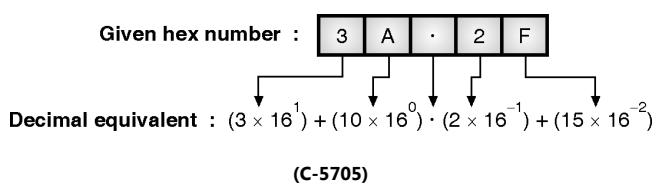
$$\therefore (0.125)_{10} = (0.0010)_2$$

(C-5704)

Step 4 : Combine the results of steps 2 and 3 :

$$(27.125)_{10} = (11011.0010)_2 \quad \dots \text{Ans.}$$

$$2. \quad (3A.2F)_{16} = (?)_{10} :$$



$$\therefore N = 48 + 10 + 0.125 + 0.0585 = 58.1835$$

$$\therefore (3A \cdot 2F)_{16} = (58.1835)_{10} \quad \dots \text{Ans.}$$

3. $(1101.0011)_2 = (?)_{10} :$

Given binary number :
 Decimal : $(1 \times 2^3) + (1 \times 2^2) + 0 + (1 \times 2^1) + \dots + 0 + 0 + (1 \times 2^{-3}) + (1 \times 2^{-4})$
 equivalent

(C-5706)

$$N = 8 + 4 + 0 + 0.125 + 0.0625 = 13.1875$$

$$\therefore (1101.0011)_2 = (13.1875)_{10} \quad \dots \text{Ans.}$$

Ex. 2.14.11 : Do the required conversions for the following numbers :

$$(BF8)_{16} = (?)_{10}$$

Dec. 11, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$(BF8)_{16} = (3064)_{10}$$

Ex. 2.14.12 : Do the required conversions for the following numbers :

$$(1FFF)_{16} = (?)_{10}$$

May 12, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$(1FFF)_{16} = (8191)_{10}$$

Ex. 2.14.13 : Convert the following hexadecimal numbers into its equivalent binary, decimal and octal numbers : 1. $(4A)_H$ 2. $(2E)_H$

May 07, 6 Marks

Soln. : Solve it yourself.

Ans. :

$$1. \quad (4A)_{16} = (74)_{10} = (0100 1010)_2 = (112)_8$$

$$2. \quad (2E)_{16} = (46)_{10} = (0010 1110)_2 = (56)_8$$

Ex. 2.14.14 : Express the following numbers in decimal. Show your step by step equations and calculations : 1. $(10110.0101)_2$ 2. $(16.5)_{16}$

Dec. 07, 6 Marks

Soln. : Solve it yourself.

Ans. :

$$1. \quad (10110.0101)_2 = (22.3125)_{10}$$

$$2. \quad (16.5)_{16} = (22.3125)_{10}$$

Ex. 2.14.15 : What is the maximum equivalent decimal number represented by 4-digit hexadecimal number ? Convert the following hexadecimal numbers to equivalent binary and octal numbers : 1. 68BE H 2. 77BA H

Dec. 07, 6 Marks



Soln. : Solve it yourself.

Ans.:

1. $(68BE)_H = (0110\ 1000\ 1011\ 1110)_2 = (64276)_8$
2. $(77BA)_H = (0111\ 0111\ 1011\ 1010)_2 = (73672)_8$

2.15 Concept of Coding :

Definition :

- When numbers, letters or text characters are represented by a specific group of symbols, it is said that the number, letter or word is being **encoded**.
- And the group of symbols is called as the **code**.

Binary codes :

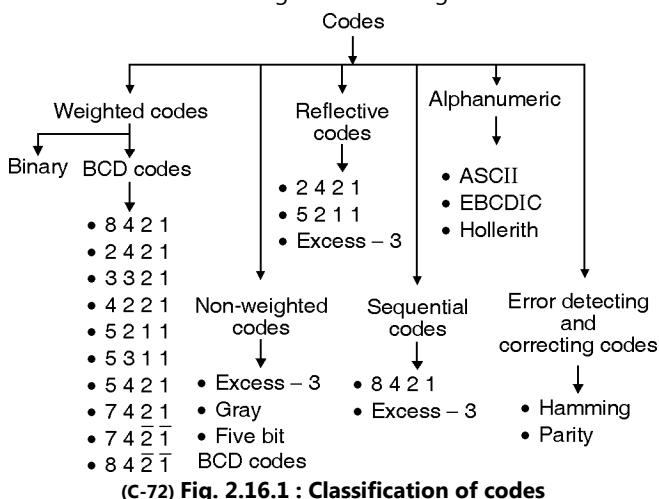
- The digital data is represented, stored and transmitted as group of binary bits. Such a group of binary bits is also called as **binary code**.
- The binary codes can be used for representing the numbers as well as alphanumeric letters.

Applications of Binary Codes :

1. In digital communication.
2. In digital computers.

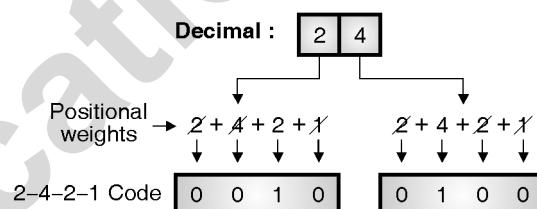
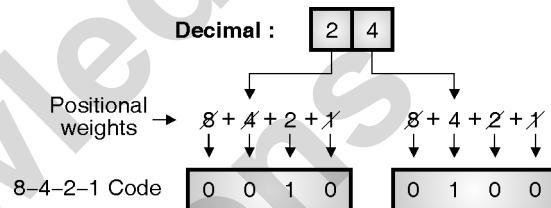
2.16 Classification of Codes :

- Fig. 2.16.1 shows the classification of codes. The codes are broadly categorized into following six categories :
 1. Weighted codes.
 2. Non-weighted codes.
 3. Reflective codes.
 4. Sequential codes.
 5. Alphanumeric codes.
 6. Error detecting and correcting codes.



2.16.1 Weighted Binary Codes :

- Weighted binary codes are those codes which are based on the principle of positional weight.
- Each position of a number represents a specific weight.
- Several systems of codes are used to express the decimal digits 0 through 9. These codes have been listed in Fig. 2.16.2.
- The codes 8421, 2421, 3321 ... all are the weighted codes.
- In these codes each decimal digit is represented by a group of four bits as shown in Fig. 2.16.2.



(C-73) Fig. 2.16.2

2.16.2 Non Weighted Codes :

- The codes in which the positional weights are not assigned, are known as non weighted codes.
- The examples of non-weighted codes are excess-3 and gray codes.

2.16.3 Alphanumeric Codes :

- The special code designed to represent numbers as well as alphabetic characters are called as the alphanumeric codes.
- Some of these codes are capable to representing some symbols and instructions as well, in addition to the numbers and alphabetic characters.
- Examples of alphanumeric codes are : ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code) and Hollerith code.



2.17 Binary Coded Decimal (BCD) Code :

Definition :

- BCD is the short form of "Binary Coded Decimal." In this code each decimal digit is represented by a 4-bit binary number.
- Thus BCD is a way to express each of the decimal digits with a binary code.
- The positional weights associated to the binary bits in BCD code are 8-4-2-1 with 1 corresponding to LSB and 8 corresponding to MSB.
- These weights are actually 2^3 , 2^2 , 2^1 and 2^0 which are same as those used in the normal binary system.

Conversion from decimal to BCD :

- The decimal digits 0 to 9 are converted into a BCD, exactly in the same way as binary.
- For example decimal 4 corresponds to 0100 BCD which is same as binary.
- Similarly the BCD numbers corresponding to the decimal numbers 0 to 9 are exactly same as the corresponding binary numbers. This is illustrated in Table 2.17.1.

(C-6142)Table 2.17.1

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

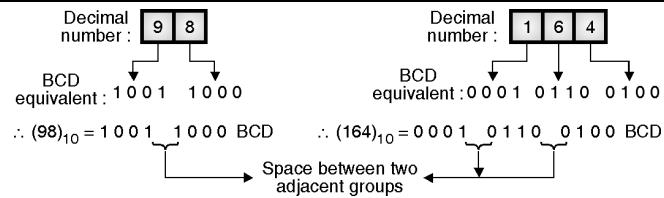
BCD numbers are same as 4 bit binary numbers

Invalid BCD codes :

- With four bits we can represent sixteen numbers (0000 to 1111).
- But in BCD code only the first ten of these are used (0000 to 1001).
- The remaining six code combinations ie 1010 to 1111 are invalid in BCD.

Conversion of big decimal numbers to BCD :

- To express any decimal number in BCD, simply express each decimal digit with its equivalent 4-bit BCD code as illustrated below.
- Fig. 2.17.1 shows the BCD codes for various decimal numbers.



(C-74) Fig. 2.17.1 : Decimal to 8-4-2-1 BCD conversion

Smallest and largest number in BCD :

- The smallest digit in BCD is (0000) i.e. 0 and the largest one is (1001) i.e. 9. The next number to 9 will be (10)₁₀ which is expressed as (0001 0000) in BCD.

2.17.1 Comparison with Binary :

1. BCD is less efficient than binary :

- Conversion of a decimal number 78 into BCD and binary is illustrated in Fig. 2.17.2.
- Fig. 2.17.2 illustrates that in order to encode the same decimal number, BCD needs more number of bits than binary.
- Hence BCD is less efficient as compared to binary.

Decimal

7	8	Binary : (1 0 0 1 1 1 0) ₂
		7 bits

BCD : 0 1 1 1 1 0 0 0	8 bits
-----------------------	--------

BCD needs more bits than binary to encode the same decimal number

(C-75) Fig. 2.17.2 : BCD is less efficient than binary

2. BCD arithmetic is more complicated than Binary arithmetic.
3. The advantage of a BCD code is that the conversion from Decimal to BCD or vice versa is simpler.
4. Table 2.17.2 shows the comparison of binary and BCD numbers from 0 to 15 decimal.

(C-6143)Table 2.17.2

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

↑
BCD and binary numbers are same

↓
BCD and binary numbers are different

**Packed BCD :**

- The BCD numbers corresponding to decimal numbers beyond 9 are called as packed BCD. Some examples of packed BCD are presented in Table 2.17.3.

Table 2.17.3 : Examples of packed BCD

Decimal	Packed BCD		
25		0010	0101
169	0001	0110	1001
523	0101	0010	0011

2.17.2 Advantages of BCD Codes :

- It is very similar to decimal system.
- We need to remember binary equivalents of decimal numbers 0 to 9 only.

2.17.3 Disadvantages :

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the same decimal number. So BCD is less efficient than binary.

Ex. 2.17.1 : Convert the following decimal numbers to BCD : (a) 35 (b) 174 (c) 2479.

Soln. :

Decimal			
BCD	0011 0101	0001 0111 0100	0010 0100 0111 1001

(C-6143(a))

2.18 Non – weighted Codes :

These codes do not work of the principle of positional weights. We will consider two such codes : 1. Excess – 3 code and 2. Gray code.

2.18.1 Excess – 3 Code :

SPPU : Dec. 04, May 07, Dec. 08

University Questions

Q. 1 With the help of suitable example, explain the meaning of self-complementing code.

(Dec. 04, 3 Marks)

Q. 2 What is excess-3 code of binary numbers : 0010 B, 0110 B and 0111 B.

(May 07, 3 Marks)

Q. 3 With the help of suitable example, explain the meaning of self complementing code.

(Dec. 08, 2 Marks)

- Excess – 3 is also called as XS - 3 code. It is a nonweighted code used to express decimal numbers as shown in Table 2.18.1.
- The Excess-3 code words are derived from the 8421 BCD code words by adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess - 3 codes are obtained as follows :

(C-6683)

Decimal number \longrightarrow 8421 BCD $\xrightarrow{\text{Add}} 0011$ Excess – 3 code

- Excess – 3 codes for the single digit decimal numbers are listed in Table 2.18.1.

(C-6146) Table 2.18.1 : Excess – 3 codes

Decimal	BCD				Excess - 3			
	8	4	2	1	BCD + 0	0	1	1
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Note : Excess - 3 is a sequential code, because each succeeding code is one binary number greater than its preceding code.

- Excess – 3 is a **sequential code** because we get any code word by adding binary 1 to its previous code word as illustrated in Table 2.18.1.
- Excess – 3 is a **self complementing code**. This is because in Excess – 3 we get the 9's complement of a number by just complementing each bit that means by replacing a 0 by 1 and 1 by 0.

Ex. 2.18.1 : Obtain the XS - 3 code for $(428)_{10}$.

Soln. :

Given number :	4	2	8	Decimal
	\downarrow	\downarrow	\downarrow	
	0100	0010	1000	BCD
+3 →	0011	0011	0011	
	0111	0101	1011	

(C-6147)



\therefore XS - 3 equivalent : 0111 0101 1011 ...Ans.

Ex. 2.18.2 : Convert the following decimal numbers into excess-3 code :

1. $(5)_{10}$
2. $(37)_{10}$
3. $(247.6)_{10}$

Soln. :

Decimal to excess - 3 conversion

1. $N = (5)_{10}$:

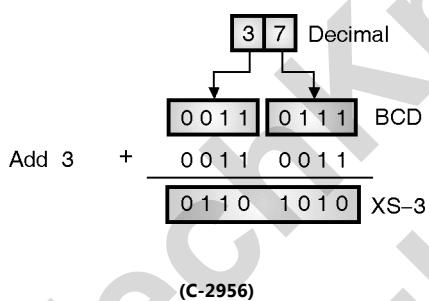
Step 1 : Write the BCD equivalent :

$$(5)_{10} = 0101$$

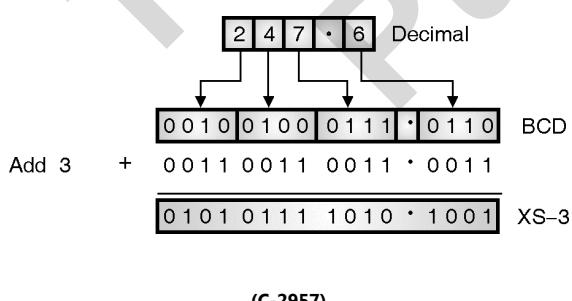
Step 2 : Convert to excess-3 :

$$\text{Excess-3 of } (5)_{10} = 0101 + 0011 = 1000 \quad \dots\text{Ans.}$$

2. $N = (37)_{10}$:



3. $N = (247.6)_{10}$:



Ex. 2.18.3 : Do the following : Convert the decimal number 25 into Binary format, Excess-3 format and BCD format. **May 18, 3 Marks**

Soln. :

$$(25)_{10} = (?)_2 = (?)_{\text{EXCESS-3}} = (?)_{\text{BCD}}$$

Step 1 : Convert decimal to binary :

2	25	1	LSB
2	12	0	
2	6	0	
2	3	1	
2	1	1	MSB
	0		

(C-7146) Fig. P. 2.18.3(a)

$$\therefore (25)_{10} = (11001)_2$$

...Ans.

Step 2 : Convert decimal to BCD :

$$\therefore (25)_{10} = (00100101)_{\text{BCD}}$$

...Ans.

Step 3 : Convert BCD to Excess-3 :

$$\text{BCD equivalent : } 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1$$

$$\begin{array}{r} \text{Add - 3 : } 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \\ \hline 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \\ \underbrace{0\ 1\ 0\ 1}_5 \quad \underbrace{1\ 0\ 0\ 0}_8 \end{array}$$

(C-7147) Fig. P. 2.18.3(b)

$$\therefore (25)_{10} = (58)_{\text{EXCESS-3}}$$

...Ans.

2.19 Gray Code :

SPPU : May 06, Dec. 06, May 07

University Questions

- Q. 1** Prepare a table of 4 bit gray code along with relationship with binary code. **(May 06, 2 Marks)**
- Q. 2** What will be the gray code of any given 4-bit binary number ? Show the truth table. **(Dec. 06, 6 Marks)**
- Q. 3** What is gray code ? What will be the gray code of binary number 1100B, 0111B and 1101 B **(May 07, 6 Marks)**

- Gray code is another non-weighted code. It is not an arithmetic code.
- It has a very special feature that only one bit in the gray code will change, each time the decimal number is incremented as shown in Table 2.19.1.
- As only one bit changes at a time, the Gray code is called as a **unit distance code**. The Gray code is a **Cyclic code**.



(C-91) Table 2.19.1

Decimal	Binary	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	0 0 0 0
16	1 0 0 0	0 0 0 0

- Note :
- In gray code only the encircled bit changes when the decimal number is incremented by 1.
 - In binary one, two, three or sometimes all the 4 bits change when the decimal number is incremented by 1.

2.19.1 Application of Gray Code :

SPPU : May 06, May 12

University Questions

- Q. 1** How gray codes are useful in digital system ?
(May 06, 2 Marks)
- Q. 2** State the applications of gray code.
(May 12, 2 Marks)

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

2.19.2 Advantages of Gray Code :

SPPU : May 06

University Questions

- Q. 1** What are the advantages of gray code over pure binary code ?
(May 06, 2 Marks)

- Consider the disc which produces the binary codes. Imagine a situation in which the existing position is 111 and the position is about to change to 000.
- If one light source out of the three is slightly ahead of the others, then the detector output would be "011" instead of 111 or 000.

- Hence a 180° error in the disc position would result and the user would not even notice it, because in binary codes, any number of digits can change their values at a given instant of time.
- This problem can be eliminated by using the Gray code instead of binary.
- In Gray code, only one bit changes at a time. So in a 3-bit code the probability of error introduction will be reduced to 33% whereas in a 4-bit code it reduces to 25%. This is the advantage of using gray code.

2.19.3 Gray-to-Binary Conversion :

Steps :

- For gray to binary conversion, follow the steps given below :

Step 1 : The MSB of Gray and binary are same. So write it directly.

Step 2 : Add (mod-2 addition) binary MSB to the next bit of Gray code. Note down the result and ignore the carries.

Step 3 : Continue this process until the LSB is reached.

- Note that the addition mentioned in step 2 is a modulo 2 addition (MOD - 2). It is equivalent to an Ex-OR operations hence denoted by \oplus sign instead of simple (+) sign.
- The rules of MOD-2 addition are as follows :

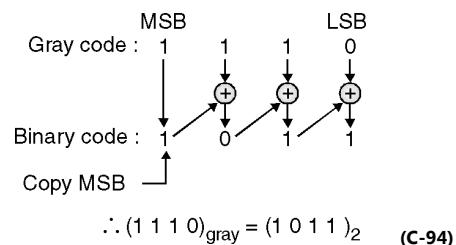
Table 2.19.2 : Rules of MOD-2 addition

0	\oplus	0	=	0
0	\oplus	1	=	1
1	\oplus	0	=	1
1	\oplus	1	=	0

- The gray to binary conversion is illustrated in the following example.

Ex. 2.19.1 : Convert 1110 gray to binary.

Soln. :





In general we can say that the conversion of a 4-bit gray number $G_3 G_2 G_1 G_0$ into a 4-bit binary number $B_3 B_2 B_1 B_0$ takes place as follows :

$$B_3 (\text{MSB}) = G_3 (\text{MSB}),$$

$$B_2 = B_3 \oplus G_2$$

$$B_1 = B_2 \oplus G_1,$$

$$B_0 = B_1 \oplus G_0$$

2.19.4 Binary to Gray Conversion :

Steps :

- A straight binary number can be converted in gray by following the steps given below :

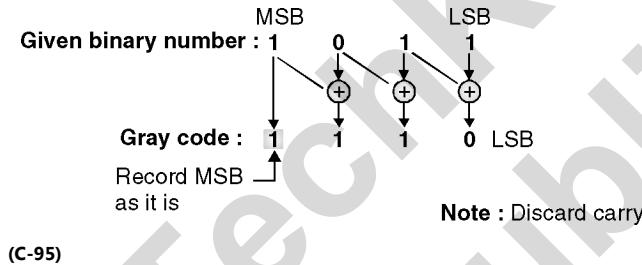
Step 1 : Record the MSB as it is, because the MSB of Gray is same as that of binary.

Step 2 : Add this bit to the next position, note down the sum and neglect any carry.

Step 3 : Repeat step 2.

Ex. 2.19.2 : Convert binary 1011 to gray.

Soln. :



(C-95)

In general the conversion of binary to gray takes place as follows :

$$G_3 (\text{MSB}) = B_3 (\text{MSB}),$$

$$G_2 = B_3 \oplus B_2$$

$$G_1 = B_2 \oplus B_1,$$

$$(LSB) G_0 = B_1 \oplus B_0$$

- The \oplus sign in the above expressions, also indicates an EX-OR (exclusive OR) function.

2.20 Code Conversions :

- In this section we are going to learn the following code conversions :
 - Binary to BCD.
 - BCD to Binary.

3. BCD to Excess – 3.

4. Excess – 3 to BCD.

2.20.1 Binary to BCD Conversion :

- For the binary to BCD conversion the steps to be followed are as given below :

Step 1 : Convert the Binary number to decimal.

Step 2 : Convert decimal number into BCD.

Ex. 2.20.1 : Convert the binary number $(110101)_2$ into BCD.

Soln. :

Step 1 : Conversion from binary to decimal :

Binary	1	1	0	1	0	1
--------	---	---	---	---	---	---

↓ ↓ ↓ ↓ ↓ ↓ ↓
 $2^5 + 2^4 + 0 + 2^2 + 0 + 1$
 $= 32 + 16 + 4 + 1 = (53)_{10}$ (C-2255)

Step 2 : Decimal to BCD :

5	3
---	---

↓ ↓
 BCD : (0 1 0 1 0 0 1 1) (C-2256)
 $\therefore (110101)_2 = (0101\ 0011)_{BCD}$...Ans.

2.20.2 BCD to Binary Conversion :

Steps to be followed :

Step 1 : Convert the BCD number into decimal.

Step 2 : Convert decimal number to binary.

Ex. 2.20.2 : Convert $(0101\ 0011)_{BCD}$ into binary.

Soln. :

Step 1 : Convert BCD to decimal :

BCD → 0 1 0 1 | 0 0 1 1

↓ ↓ ← Decimal (C-2257)
 5 3

\therefore Equivalent decimal number is 53.

Step 2 : Convert decimal to binary :

- Use the long division method for decimal to binary conversion to get,

$$\therefore (53)_{10} = (110101)_2$$

$$\therefore (0101\ 0011)_{BCD} = (110101)_2$$

2.20.3 BCD to Excess – 3 :

- For BCD to Excess – 3 conversion, follow the steps given below :



Step 1 : Convert BCD to decimal.

Step 2 : Add $(3)_{10}$ to this decimal number.

Step 3 : Convert the decimal number of step 2 into binary, to get the excess – 3 code.

Ex. 2.20.3 : Convert $(1001)_{BCD}$ to excess – 3.

Soln. : (C-2258)

$$\begin{array}{r}
 \text{BCD} \quad \boxed{1 \ 0 \ 0 \ 1} \\
 \text{Decimal} \quad \downarrow \\
 \text{Add 3} \quad + \ 3 \\
 \hline (12)_{10} \quad \xrightarrow{\text{Convert to binary}} (1 \ 1 \ 0 \ 0)_2 \\
 \therefore (1001)_{BCD} = (1100)_{xs-3}
 \end{array}$$

Ex. 2.20.4 : Convert $(0101\ 0011)_{BCD}$ into excess – 3.

Soln. : (C-2259)

$$\begin{array}{r}
 \text{BCD number} \rightarrow \boxed{0 \ 1 \ 0 \ 1 \ | \ 0 \ 0 \ 1 \ 1} \\
 \text{Decimal} \rightarrow \begin{matrix} 5 & 3 \\ 3 & 3 \end{matrix} \\
 \text{Add 3} \rightarrow \begin{matrix} 5 & 3 \\ 3 & 3 \end{matrix} \\
 \text{to each digit} \quad \hline \begin{matrix} 8 & 6 \end{matrix} \\
 \\
 \text{Convert each digit to 4 bit binary} \\
 86 = 1000 \ 0110 \\
 \therefore (0101\ 0011)_{BCD} = (1000\ 0110)_{xs-3}
 \end{array}$$

Alternative method for BCD to XS – 3 conversion :

- Add $(0011)_2$ to each 4-bit BCD number to obtain the corresponding Excess – 3 number.

Ex. 2.20.5 : Convert BCD 0101 0011 to excess – 3.

Soln. : (C-6372)

$$\begin{array}{r}
 \text{Given BCD number} \quad 0 \ 1 \ 0 \ 1 \quad 0 \ 0 \ 1 \ 1 \\
 \text{Add } (0011)_2 \quad - \quad 0 \ 0 \ 1 \ 1 \quad 0 \ 0 \ 1 \ 1 \\
 \text{Excess - 3 number} \quad \hline \quad 1 \ 0 \ 0 \ 0 \quad 0 \ 1 \ 1 \ 0 \\
 \therefore (01010011)_{BCD} = (10000110)_{xs-3} \quad \dots \text{Ans.}
 \end{array}$$

2.20.4 Excess – 3 to BCD Conversion :

- Subtract $(0011)_2$ from each 4 bit excess – 3 digit to obtain the corresponding BCD code.

Ex. 2.20.6 : Obtain the BCD code equivalent of $(1001\ 1010)$.

Soln. :

$$\begin{array}{r}
 \text{Given XS - 3 number} \quad 1 \ 0 \ 0 \ 1 \quad 1 \ 0 \ 1 \ 0 \\
 \text{Subtract } (0011)_2 \quad - \quad 0 \ 0 \ 1 \ 1 \quad 0 \ 0 \ 1 \ 1 \\
 \text{BCD} \quad \hline \quad 1 \ 1 \quad 1 \ 1 \ 1 \\
 \therefore (10011010)_{xs-3} = (01100111)_{BCD}
 \end{array}$$

(C-6362)

Ex. 2.20.7 : Represent the decimal numbers : (a) 396 and (b) 4096 in : 1. BCD 2. Excess - 3 code.

May 08, 4 Marks

Soln. : Solve it yourself.

Ans. :

$$\begin{aligned}
 (396)_{10} &= (0011\ 1001\ 0110)_{BCD} \\
 &= (0110\ 1100\ 1001)_{xs-3} \\
 (4096)_{10} &= (0100\ 0000\ 1001\ 0110)_{BCD} \\
 &= (0111\ 0011\ 1100\ 1001)_{xs-3}
 \end{aligned}$$

Ex. 2.20.8 : Represent decimal number 327 in : 1. BCD code 2. Excess-3 code.

Dec. 08, 2 Marks

Soln. : Solve it yourself.

Ans. :

$$\begin{aligned}
 (327)_{10} &= (0011\ 0010\ 0111)_{BCD} \\
 &= (011001011010)_{xs-3}
 \end{aligned}$$

Ex. 2.20.9 : Find gray codes for the following binary numbers : 1. 11001100 2. 01011110 .

Dec. 08, 2 Marks

Soln. : Solve it yourself.

Ans. :

1. $(11001100)_2 = (10101010)_{gray}$
2. $(01011110)_2 = (01110001)_{gray}$

Review Questions

Q. 1 Define base or radix of a number system.

Q. 2 Explain the following :

- Bit
- Nibble
- Word
- Double word

Q. 3 What are the disadvantages of a binary system ?

Q. 4 Describe the hexadecimal system.

Q. 5 Write a short note on : Octal system.

Q. 6 What is BCD code ?

Q. 7 What is excess-3 code ?

Q. 8 Write short note on ASCII code.

Q. 9 What are the advantages of BCD code over binary code ?

Q. 10 What are the different types of codes used in digital systems ? Explain them.



- Q. 11 What is gray code ? What are its applications ?
- Q. 12 What is BCD code ?
- Q. 13 What is excess-3 code ?
- Q. 14 Write short note on ASCII code.
- Q. 15 What are the different types of codes used in digital systems ? Explain them.

- Q. 16 Give four comparison between BCD code and Gray code.
- Q. 17 Differentiate between Binary and Gray code.
- Q. 18 Distinguish between Excess-3 code and Gray code.
- Q. 19 Explain the rules of BCD addition.

□□□

TechKnowledge
Publications

Unit 1

Chapter

3

Binary Arithmetic

Syllabus

Signed binary number representation and arithmetic : Sign magnitude, 1's complement and 2's complement representation, Unsigned binary arithmetic (Addition, subtraction, multiplication and division), Subtraction using 2's complement; IEEE standard 754 floating point number representations.
Case study : Four basic arithmetic operations using floating point numbers in a calculator.

Chapter Contents

3.1 Introduction	3.7 IEEE-754 Standard for Representing Floating Point Numbers
3.2 Unsigned Binary Numbers	3.8 Introduction to Boolean Algebra
3.3 Sign-Magnitude Numbers	3.9 Definition of Boolean Algebra
3.4 Complements	3.10 Two Valued Boolean Algebra
3.5 2's Complement Arithmetic	3.11 Basic Theorems and Properties of Boolean Algebra
3.6 Floating Point Representation	3.12 Boolean Expression and Boolean Function



3.1 Introduction :

- Let us develop various rules for carrying out the arithmetic operations such as addition, subtraction, multiplication and division.
- Binary arithmetic is essential in all the digital computers and many other digital systems.

3.2 Unsigned Binary Numbers :

- In some applications, all the data is either positive or negative.
- Then we can just forget about the (+) or (-) signs, and concentrate only on the magnitude (absolute value) of the data.
- For example, the smallest 8 bit binary number is 0000 0000 i.e. all zeros, and the largest 8 bit binary number is 1111 1111.
- Hence the complete range of unsigned 8 bit binary numbers extends from $(00)_H$ to $(FF)_H$ or from $(00)_{10}$ to $(255)_{10}$.

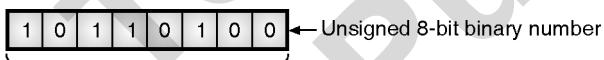
It is important to note that we have not included (+) or (-) signs with these numbers.

- Similarly for 16-bit numbers the complete range is given by,

Smallest : 0000 0000 0000 0000 = $(0000)_H$

Largest : 1111 1111 1111 1111 = $(FFFF)_H$

- This represents the magnitude of 0 to 65,535 decimal.



All the bits are used for representing only the magnitude

(C-6109) Fig. 3.2.1 : Unsigned 8 bit binary number

- Data of this type is called as **unsigned binary** numbers because all of the bits in a binary number are used to represent the magnitude of the corresponding decimal number, as shown in Fig. 3.2.1.

3.2.1 Important Features of Unsigned Numbers :

- We can add or subtract the unsigned binary numbers if certain conditions are satisfied.
- The microcomputers of first generation were able to process only 8 bits at a time. Therefore with 8-bit

unsigned arithmetic all the magnitudes were restricted between 0 and $(255)_{10}$ or $(00)_H$ to $(FF)_H$.

- That means all the numbers being added or subtracted must be in the range 0 to 255. More important is that the answer also should be in the range 0 to 255.
- For the magnitudes greater than 255 we have to use 16-bit arithmetic.

Overflow :

- If the addition or multiplication of two 8-bit numbers results in generation of a number, greater than $(255)_{10}$, then it is said that overflow has taken place.

3.2.2 Unsigned Binary Arithmetic :

- In the following subsections we will discuss the four basic arithmetic operations on the binary numbers: Addition, subtraction, multiplication and division

3.2.3 Binary Addition :

- Binary addition is the key for binary multiplication, subtraction and division.
- The four most basic cases of binary addition are shown in Table 3.2.1.

(C-6107) Table 3.2.1 : Four cases of binary addition

A	B	Addition	Comment
Case 1	0	+ 0 = 0	{ Same as
Case 2	0	+ 1 = 1	decimal addition
Case 3	1	+ 0 = 1	
Case 4	1	+ 1 = 10	• + • = • • = $(10)_2$

- For cases 1, 2 and 3 of Table 3.2.1, the binary addition takes place by following the rules of decimal addition.
- But concentrate on case 4. Addition of binary 1 + 1 represents the combining of one pebble and one pebble to obtain a total of two pebbles.

$$1 + 1 = \bullet \bullet \text{ two pebbles}$$

- Since binary 10 stands for • • two pebbles, the result of binary addition 1 + 1 is 10.

$$\therefore 1 + 1 = (10)_2 \quad \dots(3.2.1)$$

3.2.4 Sum and Carry :

- Thus the fourth case yields a binary two (10). When the binary numbers are added, the fourth case in Table 3.2.1 creates a sum of 0 in the given column and a carry of 1 over to the next column.



- The four basic rules of binary addition in terms of sum and carry are as follows.

(C-6902(a)) Table 3.2.2 : Rules of binary addition

Rule	A	+	B	=	Sum	Carry
1	0	+	0	=	0	0
2	0	+	1	=	1	0
3	1	+	0	=	1	0
4	1	+	1	=	0	1

Ex. 3.2.1 : Add the following binary numbers : 011 and 101.

Soln. :

$$\begin{array}{r}
 \text{Carry} & \boxed{1} & \boxed{1} & \boxed{1} \\
 & \downarrow & \downarrow & \downarrow \\
 + \quad A & 0 & 1 & 1 \\
 + \quad B & 1 & 0 & 1 \\
 \text{Ans. :} & \boxed{1} & \boxed{0} & \boxed{0} \\
 & \downarrow & \downarrow & \downarrow \\
 & 1 & 0 & 0 & 0
 \end{array}
 \quad \therefore 011 + 101 = 1000 \\
 \text{i.e. } 3 + 5 = 8$$

(C-57)

3.2.5 Binary Subtraction :

Rules :

- In order to understand the binary subtraction, we should remember some of the important rules of decimal subtraction.
- They are as follows :

 - To carry out the subtraction $(A - B)$ where A and B are the two single digit decimal numbers. We have to consider two cases.

2. Case I : Digit A > Digit B :

$$\text{Let } A = (5)_{10} \text{ and } B = (3)_{10}$$

$$\text{Then } A - B = (\bullet\bullet\bullet) - (\bullet\bullet) = \bullet\bullet$$

$$\therefore (5)_{10} - (3)_{10} = (2)_{10}$$

3. Case II : Digit A < Digit B :

- If $A = (3)_{10}$ and $B = (5)_{10}$ then we cannot perform $(3 - 5)$ because we cannot take out 5 pebbles from 3. **Therefore we have to borrow 1.** After borrowing, the subtraction is changed to

$$\begin{array}{r}
 \boxed{1}3 - 5 = 8 \\
 \downarrow \text{Borrow} \\
 \text{(C-4898)}
 \end{array}$$

- We have to do the same thing for subtracting the binary numbers.

3.2.6 Subtraction and Borrow :

- These two words will be used very frequently for the binary subtraction.
- For binary subtraction we have to remember the following four cases given in Table 3.2.3.

(C-6343) Table 3.2.3 : Four basic rules for binary subtraction

Case	A	B	Subtraction	Borrow	Comment
1	0	- 0	0	0	Same as decimal
2	1	- 0	1	0	Same as decimal
3	1	- 1	0	0	Same as decimal
4	0	- 1	1	1	Borrow needs to be taken

- Consider case 4 in Table 2.1.3. It is $[0 - 1]$. Hence a logic 1 is borrowed.
- This will change the subtraction from $[0 - 1]$ to $[10 - 1]$ that means $[\bullet\bullet - \bullet] = \bullet = 1$.

Ex. 3.2.2 : Subtract the decimal numbers $(38)_{10}$ and $(29)_{10}$ by converting them into binary.

Soln. :

Step 1 : Convert $(38)_{10}$ and $(29)_{10}$ into their binary equivalents :

Convert the given decimal numbers into binary numbers to get,

$$(38)_{10} = (100110)_2, (29)_{10} = (11101)_2$$

Step 2 : Perform column by column subtraction from LSB to MSB :

$$\begin{array}{r}
 \text{Column of LSBs} \\
 \downarrow \\
 \text{Borrow} & \boxed{1} \\
 - \quad A & 1 & 0 & 0 & 1 & 1 & 0 & \text{Decimal 38} \\
 - \quad B & 0 & 1 & 1 & 1 & 0 & 1 & \text{Decimal 29} \\
 \hline
 & & & & & & 1
 \end{array}$$

(C-63)

Step 3 : Repeat the procedure for all the columns :

$$\begin{array}{r}
 \text{Column 5} \quad \text{Column 4} \\
 \downarrow \quad \downarrow \\
 \text{Borrow} & \boxed{1} & \boxed{1} & \boxed{1} \\
 - \quad A & 1 & 0 & 0 & 1 & 1 & 0 \\
 - \quad B & 0 & 1 & 1 & 1 & 0 & 1 \\
 \hline
 \text{Ans. :} & 0 & 0 & 1 & 0 & 0 & 1
 \end{array}$$

(C-64)

$$\text{Column 4 : } 10 - 1 = 1$$

$$\text{Column 5 : } 10 - 1 - 1 = \bullet\bullet - \bullet - \bullet = 0$$

$$\text{Column 6 : } 1 - 1 = 0$$



3.2.7 Binary Multiplication :

- The procedure used for binary multiplication is exactly same as that for the decimal multiplication.
- In fact binary multiplication is simpler than decimal multiplication because only 0s and 1s are involved.

Rules of binary multiplication :

Rules of binary multiplication are as follows :

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

- The multiplication process of two binary numbers has been illustrated in the following example.

Ex. 3.2.3 : Perform the following binary multiplication

$$101.11 \times 111.01.$$

Soln. :

A		×	1	0	1	·	1	1
B			1	1	1	·	0	1
			1	0	1	1	1	1
+			0	0	0	0	0	-
+			1	0	1	1	1	-
+			1	0	1	1	1	-
+			1	0	1	1	1	-
1	0	1	0	0	1	1	0	1

The binary point is placed after 4 positions from LSB

$$\therefore \text{Ans.} : 101001.1011 \quad (\text{C-6108})$$

Cross check :

$$A = (101.11)_2 = (5.75)_{10}$$

$$\text{and } B = (111.01)_2 = (7.25)_{10}$$

$$\therefore A \times B = (41.6875)_{10}$$

Ans. =

$$32 + 8 + 1 + 0.5 + 0.125 + 0.0625 = (41.6875)_{10}$$

(C-6344)

...Ans.

Thus we have the correct answer.

Ex. 3.2.4 : Perform : $(11001)_2 \times (101)_2$

Soln. :

$$(11001)_2 \times (101)_2 : \quad (\text{C-6771})$$

$$\begin{array}{r} 1 & 1 & 0 & 0 & 1 \\ \times & & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 1 \end{array}$$

$$+ 0 0 0 0 0 - \quad \leftarrow \text{Shift left by 1 position}$$

$$+ 1 1 0 0 1 - - \quad \leftarrow \text{Shift left by 2 positions}$$

Answer :

$$1 1 1 1 1 1 0 1$$

$$\therefore (11001)_2 \times (101)_2 = (1111101)_2$$

...Ans.

3.2.8 Binary Division :

- The division of binary numbers takes place in a similar way as that of decimal number. It is called as the long division procedure.

Ex. 3.2.5 : Perform following operation without converting to any other base.

$$(10101011)_2 \div (101)_2$$

Soln. :

$$(10101011)_2 \div (101)_2 : \quad (\text{C-2172})$$

$$\begin{array}{r} 1 0 0 0 1 0 \\ 1 0 1) 1 0 1 0 1 1 \\ - 1 0 1 \\ \hline 0 0 1 0 1 \\ - 1 0 1 \\ \hline 0 0 1 \end{array}$$

Quotient = $(100010)_2$

Remainder = $(1)_2$

...Ans.

Ex. 3.2.6 : Perform $(11010)_2 \div (101)_2$

Soln. :

$$(11010)_2 \div (101)_2 : \quad (\text{C-2174})$$

$$\begin{array}{r} 1 0 1 \\ 1 0 1) 1 1 0 1 0 \\ - 1 0 1 \\ \hline 1 1 0 \\ - 1 0 1 \\ \hline 1 \end{array}$$

Quotient = 101

Remainder = 1

Ex. 3.2.7 : Perform $(11001)_2 \div (101)_2$

Soln. :

$$\begin{array}{r} 1 0 1 \\ 1 0 1) 1 1 0 0 1 \\ - 1 0 1 \\ \hline 0 0 1 0 1 \\ - 1 0 1 \\ \hline 0 0 0 \end{array}$$

(C-3704)

$$\therefore (11001)_2 \div (101)_2 = (101)_2$$

...Ans.

3.3 Sign-Magnitude Numbers :

SPPU : Dec. 07, Dec. 11

University Questions

Q. 1 What do you mean by signed magnitude representation of a number ? (Dec. 07, 2 Marks)

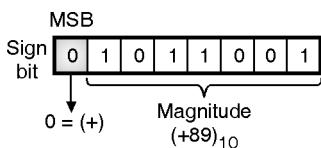
Q. 2 What are different ways of representing signed binary numbers ? Explain with examples.

(Dec. 11, 6 Marks)

- If the data has positive as well as negative numbers then the signed binary numbers should be used, for their representation.



- For a sign-magnitude representation, the + or - signs are also represented in the binary form i.e. by using 0 or 1. So a 0 is used to represent the (+) sign and 1 is used to represent the (-) sign.
- The MSB of a binary number is used to represent the sign and the remaining bits are used for representing the magnitude. 8-bit signed binary numbers are shown in Fig. 3.3.1.



(C-6110) Fig. 3.3.1(a) : A positive binary number

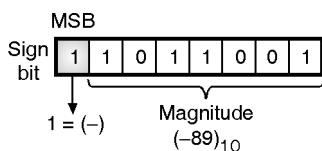


Fig. 3.3.1(b) : A negative binary number 8-bit signed binary numbers

Definition of signed numbers :

- The numbers shown in Fig. 3.3.1 contain a sign bit followed by magnitude bits.
- Numbers represented in this form are called as **sign-magnitude numbers** or only sign-numbers.

3.3.1 Range of Sign-Magnitude Numbers :

- The unsigned 8-bit numbers cover the decimal range of 0 to 255.
- But in the sign magnitude numbers, the MSB is utilized for representing the sign.
- Therefore the range gets modified. (as there are only 7 bits left to represent the magnitude).
- For an 8-bit sign-magnitude number, the largest negative number is (- 127) given by,

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

(C-6369)

- And the largest positive number is (+ 127) given by,

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

(C-6369)

In this way the range of sign-magnitude, 8-bit binary numbers is modified to decimal (- 127) to (+127) from 0 to 255.

- The largest magnitude is 127, which is approximately half of the largest magnitude obtained for unsigned binary numbers.
- With the sign-magnitude numbers, we can use the 8-bit arithmetic as long as the input data range falls in decimal – 127 to + 127.
- It is still necessary to check all the sums for overflow.
- If the magnitude of data is greater than 127 then 16 bit arithmetic should be used.
- With 16-bit numbers the range of sign-magnitude numbers extends from decimal (- 32,767) to (+ 32,767).

Advantages of sign-magnitude numbers :

- The main advantage is their simplicity.
- We can easily find the magnitude by deleting the sign bit.

Disadvantage :

- The sign-magnitude numbers have a limited use because they require complicated circuits.
- These numbers are often used in analog-to-digital (A to D) converters.

3.4 Complements :

- Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations.

3.4.1 1's Complement :

- The 1's and 2's complement of a binary number are important because we can use them for representation of negative numbers.

Definition :

- The 1's complement of a number is found by inverting all the bits in that number. This is called as taking complement.
- This complemented value represents the negative of the original number. The 1's complement system is very easy to implement merely using inverters.

Ex. 3.4.1 : Obtain the 1's complement of the following numbers. $(1010)_2$, $(11010101)_2$.

**Soln. :**

Given number :	1010	1101 0101
1's complement :	0101	0010 1010

(C-6279(a))

Ex. 3.4.2 : Convert following binary numbers to 1's complement (a) $(1101)_2$ (b) $(1011)_2$ **Soln. :**

- (a) Given number : 1 1 0 1
1's complement : 0 0 1 0 ...Ans.
- (b) Given number : 1 0 1 1
1's complement : 0 1 0 0 ...Ans.

3.4.2 Representation of Signed Numbers using 1's Complement :

- Positive number in 1's complement form are represented in the same way as the positive sign magnitude numbers. That means $(+5)_{10}$ can be represented as 0101.
- But the negative numbers are the 1's complement of the corresponding positive numbers.
- For example $(-5)_{10}$ is represented as follows :

$$\begin{aligned} N = (-5)_{10} &\quad \dots \text{ Given number} \\ (+5)_{10} = 0101 &\quad \dots \text{ In sign magnitude form} \\ &\quad \begin{array}{c} | \\ \text{Magnitude} \end{array} \\ &\quad \text{Sign} \\ \therefore N = 1010 &\quad \dots \text{ 1's complement form} \end{aligned}$$

(C-69)

- Thus $(-5)_{10}$ is represented as 1010 in the 1's complement form.

Range of 1's complement number system :

- From the discussion done till now, the maximum positive number in the 1's complement form for a 4 bit number is 0111.

∴ For $n = 4$ the maximum positive number is 7. i.e. $[2^{n-1} - 1]$

- The maximum negative number is represented as 1111. Hence for $n = 4$ the maximum negative number is -7 i.e. $[2^{n-1} - 1]$.

∴ Range of 1's complement number system is given by,

$$\text{Maximum positive number} \Rightarrow (2^{n-1} - 1)$$

$$\text{Maximum negative number} \Rightarrow -(2^{n-1} - 1)$$

- Table 3.4.1 shows the full range of 4 bit numbers in the 1's complement form.

(C-6111) Table 3.4.1

Decimal number	1's Complement
+ 7	0 1 1 1
+ 6	0 1 1 0
+ 5	0 1 0 1
+ 4	0 1 0 0
+ 3	0 0 1 1
+ 2	0 0 1 0
+ 1	0 0 0 1
+ 0	0 0 0 0
- 0	1 1 1 1
- 1	1 1 1 0
- 2	1 1 0 1
- 3	1 1 0 0
- 4	1 0 1 1
- 5	1 0 1 0
- 6	1 0 0 1
- 7	1 0 0 0

Positive numbers in their normal form.

Negative numbers in their 1's complement form.

- One of the difficulties in using 1's complement is its representation of zero.
- As seen from Table 3.4.1 both 0 0 0 0 and 1 1 1 1 represent zero. The [0 0 0 0] is called as positive zero and [1 1 1 1] is called as the negative zero.

3.4.3 2's Complement :

Definition :

- The 2's complement of a binary number is obtained by adding 1 to the LSB of 1's complement of that number.
- ∴ 2's complement = 1's complement + 1
- The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

Ex. 3.4.3 : Obtain the 2's complement of $(1011\ 0010)_2$.**Soln. :**

$$\begin{array}{rcl} 1. \text{ Given number} & : & 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ 2. \text{ 1's complement} & : & 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ 3. \text{ Add 1 to 1's complement} & : & \hline & & 1 \\ 4. \text{ 2's complement} & : & 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \end{array}$$

(C-6345)

Hence the 2's complement of $(1011\ 0010)_2$ is $(0100\ 1110)_2$.

3.4.4 Representation of Signed Numbers using 2's Complement :

- Positive numbers in 2's complement form are represented the same way as in the sign-magnitude and 1's complement form.



- For example $(+ 6)_{10}$ is represented as 0110 in the 2's complement form.
- Negative numbers are the 2's complements of the corresponding positive numbers.
- For example $(- 6)_{10}$ is represented in the 2's complement form as follows :

$$N = (-6)_{10} \dots \text{given}$$

$$(+6)_{10} = \begin{array}{l} 0 \ 1 \ 1 \ 0 \\ \text{Invert all bits and add 1} \end{array}$$

$$1 \ 0 \ 1 \ 0 \dots \text{2's complement of } (-6)_{10}$$

(C-6112)

- 2's complement is another method of storing negative values. In a microcomputer the positive and negative numbers are represented with the help of 2's complement.
- The representation of four bit (+) and (-) numbers using 2's complement is shown in Table 3.4.2.
- The range of a 4 - bit sign-magnitude number extends from decimal $(- 7)$ to $(+ 7)$. The $(0)_{10}$ and positive numbers are represented by normal binary numbers in Table 3.4.2.

(C-6114) Table 3.4.2 : Two's complement numbers

Decimal	Signed binary	2's complement
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
0	0000	0000
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	1000	1000

- But the negative numbers are represented in the 2's complement form.
- However note that the MSB (sign bit) is not changed while obtaining the 2's complement. For example 2's complement of $(- 6)_{10}$ is obtained as follows :

$$(-6)_{10} = (1 \ 1 \ 1 \ 0)_2$$

$$\text{Its 2's complement} = \boxed{1} \ 0 \ 1 \ 0$$

No change

(C-6113)

- Similarly the 2's complement of other negative numbers are obtained.

Important observations from Table 3.4.2 :

- Two's complement numbers have some interesting characteristics. They are as follows :
 - There is one unique 0.
 - The two's complement of 0 is 0.
 - The MSB of a sign-magnitude number cannot be used to express quantity. It can only be used as a sign bit.
 - There are eight negative integers, seven positive integers and one zero, making a total of 16 unique numbers.

\therefore In an n bit 2's complement system, there will always be,

- 2^{n-1} – 1 positive integers
- 2^{n-1} negative integers
- and one 0

Total 2^n unique states

(C-6115)

- To convert a negative number to a positive number, find its 2's complement.
- 2's complement of a number results in the original number itself. For example if the given number is $(4)_{10} = (0 \ 1 \ 0 \ 0)_2$ then its 2's complement is $(1 \ 1 \ 0 \ 0)_2$ and the 2's complement of $(1 \ 1 \ 0 \ 0)_2$ is $(0 \ 1 \ 0 \ 0)_2$ i.e. $(4)_{10}$.

3.4.5 Signed Complement Numbers :

- When arithmetic operations are carried out in a computer, it is more convenient to use a different system for representing the negative numbers.
- This system is called as the signed – complement system.
- In this system a negative number is represented by its complement.
- In the signed – magnitude system a number is negated by changing its sign, but the signed complement system will negate the number by taking its complement.
- We know that a positive number always starts with a zero (plus) in its MSB position, therefore the



complement will always start with a 1, indicating a negative number.

- The signed – complement system can use either 1's complement or 2's complement, but the 2's complement is most commonly used.
- As an example let us see how to represent $-(7)_{10}$ using different techniques using 8 binary bits.

	Sign	
1. Signed - Magnitude representation :		
2. Signed - 1's complement representation :		
3. Signed - 2's complement representation :		

(C-71)

- In the signed-magnitude representation, the left most 1 represents $(-)$ sign and the remaining seven bits represent the magnitude 7.
- In the signed 1's complement representation, -7 is obtained by complementing all the bits of $+7$ including the sign bit.
- In the signed 2's complement, -7 is obtained by obtaining the 2's complement of $+7$, including the sign bit.
- Table 3.4.3 shows all the possible 4 bit signed binary numbers in the three representations discussed now.

(C-8115) Table 3.4.3 : Signed binary numbers

Decimal	Signed 2's complement	Signed 1's complement	Signed magnitude
+7	0 1 1 1	0 1 1 1	0 1 1 1
+6	0 1 1 0	0 1 1 0	0 1 1 0
+5	0 1 0 1	0 1 0 1	0 1 0 1
+4	0 1 0 0	0 1 0 0	0 1 0 0
+3	0 0 1 1	0 0 1 1	0 0 1 1
+2	0 0 1 0	0 0 1 0	0 0 1 0
+1	0 0 0 1	0 0 0 1	0 0 0 1
+0	0 0 0 0	0 0 0 0	0 0 0 0
-0	-	1 1 1 1	1 0 0 0
-1	1 1 1 1	1 1 1 0	1 0 0 1
-2	1 1 1 0	1 1 0 1	1 0 1 0
-3	1 1 0 1	1 1 0 0	1 0 1 1
-4	1 1 0 0	1 0 1 1	1 1 0 0
-5	1 0 1 1	1 0 1 0	1 1 0 1
-6	1 0 1 0	1 0 0 1	1 1 1 0
-7	1 0 0 1	1 0 0 0	1 1 1 1
-8	1 0 0 0	-	-

Observations :

1. Table 3.4.3 shows that the positive numbers with all the three representations are identical and they have a 0 in their left most position.
2. The signed 2's complement system has only one representation for 0 which is always positive but the other two systems have either a positive zero or a negative zero.
3. All the negative numbers have a 1 in the leftmost bit position. This is how we can distinguish the negative numbers from the positive ones.
4. With four bits we can represent 16 binary numbers. But in the signed magnitude and 1's complement systems there are eight positive and eight negative numbers including two zeros.
- In 2's complement system there are eight positive numbers including one zero and eight negative numbers.
- The signed magnitude system is awkward when used in computer arithmetic because the sign and magnitude need to be handled separately.
- Therefore the signed complement system is preferred in computer arithmetic.
- The 1's complement creates some difficulties and is generally not used for the arithmetic operations.
- But it is useful as a logical operation because the change of 0 to 1 and 1 to 0 is equivalent to logical complement operation.
- The signed 2's complement system is commonly used in the computer arithmetic.

Ex. 3.4.4 : Express the decimal numbers +25 and -25 in the 8 bit sign magnitude, 1's complement and 2's complement forms.

Soln. :

Representation of + 25 :

+ 25 is represented in the sign magnitude, 1's complement and 2's complement forms as follows :

**Step 2 : Add $(4)_{10}$ to 2's complement of $(9)_{10}$:**

$$(4)_{10} : \quad 0 \ 1 \ 0 \ 0$$

$$\text{2's complement of } (9)_{10} : + \ 0 \ 1 \ 1 \ 1$$

Carry : 1

Final carry

0

1 0 1 1
Answer is negative and in the 2's complement form.

"0" carry indicates that the result is negative and in its 2's complement form.

(C-6124)

Step 3 : Convert the answer into its true form :

Answer : 1 0 1 1 In 2's complement form

Subtract 1 : - 1

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \\ \hline 1 \ 0 \ 1 \ 0 \end{array}$$

Invert all bits

$$0 \ 1 \ 0 \ 1 \quad \text{Answer in true form}$$

(C-6125)

Thus the answer is $-(0101)_2$ i.e. $(-5)_{10}$Ans.

Ex. 3.5.3 : Perform subtraction using 2's complement for given numbers $(-48) - (+23)$ use 8 bit representation of number.

Dec. 13, 3 Marks, May 19, 3 Marks**Soln. :****Step 1 : Convert both the numbers to their 2's complement :**

Decimal	Binary	2's complement
$(48)_{10}$	$(00110000)_2$	$(11010000)_2$
$(23)_{10}$	$(00010111)_2$	$(11101001)_2$

Step 2 : Add 2's complement of $(48)_{10}$ and $(23)_{10}$:

$$\text{2's complement of } (48)_{10} \quad \begin{array}{r} 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \\ + \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

$$\text{2's complement of } (23)_{10} \quad \begin{array}{r} 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \\ + \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

Final carry

1

Result is negative and in 2's complement form

(C-6128)

Step 3 : Convert answer to its true form :

Answer : 1 0 1 1 1 0 0 1

Subtract 1 : - 1

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ - \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \\ \hline \end{array}$$

Invert all bits : 0 1 0 0 0 1 1 1 Answer in true form

(C-6349)

 $\therefore -(01000111)_2 = (-71)_{10}$ $\therefore (-48) - (+23) = -(01000111)_2 = (-71)_{10}$

Ex. 3.5.4 : Perform the following operations using 2's complement method.

$$-(48)_{10} - (-23)_{10}$$

Dec. 13, 6 Marks, May 19, 3 Marks**Soln. :**

$$-(48)_{10} - (-23)_{10} :$$

We have to perform $-(48)_{10} + (23)_{10}$ **Step 1 : Convert number $(-48)_{10}$ to its 2's complement :**

(C-8292)

Decimal	Binary	2's complement
$(-48)_{10}$	$(0110000)_2$	$(1010000)_2$

Step 2 : Add 2's complements of $(-48)_{10}$ and $(23)_{10}$:

$$\text{2's complement of } (-48)_{10} = 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0$$

$$+ \quad (23)_{10} = 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1$$

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \end{array}$$

(C-4183)

- As final carry is not generated the answer is negative and in 2's complement form.

Step 3 : Convert the answer into its true form :

(C-8293)

$$\begin{array}{ccc} & \begin{matrix} 1's \\ \text{complement} \end{matrix} & \begin{matrix} \text{add} \\ 1 \end{matrix} \\ (1100111)_2 & \xrightarrow{\hspace{1cm}} & (0011000)_2 \xrightarrow{\hspace{1cm}} (0011001)_2 \end{array}$$

$$\therefore -(48)_{10} - (-23)_{10} = -(11001)_2 = -(25)_{10}$$

...Ans.

Ex. 3.5.5 : Subtract $(27.50)_2$ from $(68.75)_2$ using 2's complement method.

Dec. 14, 4 Marks, May 18, 6 Marks**Soln. :****Step 1 : Convert decimal number to binary :**

$$A = (27.50)_{10} = (00011011.1000)_2$$

$$B = (68.75)_{10} = (01000100.1100)_2$$

Step 2 : Obtain 2's complement of $(27.50)_{10}$:

$$\begin{array}{ccc} & \begin{matrix} 1's \\ \text{complement} \end{matrix} & \begin{matrix} \text{add} \\ 1 \end{matrix} \\ (00011011.1000)_2 & \xrightarrow{\hspace{1cm}} & (11100100.0111)_2 \xrightarrow{\hspace{1cm}} (11100100.1000)_2 \end{array}$$

(C-8294)

Step 3 : Add $(68.75)_{10}$ and 2's complement of $(27.50)_{10}$:

$$(68.75)_{10} = 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \cdot 1 \ 1 \ 0 \ 0$$

$$-(27.50)_{10} = 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \cdot 1 \ 0 \ 0 \ 0$$

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \cdot 1 \ 1 \ 0 \ 0 \\ + \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \cdot 1 \ 0 \ 0 \ 0 \\ \hline \end{array}$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \cdot 0 \ 1 \ 0 \ 0 \\ \hline \end{array}$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \cdot 0 \ 1 \ 0 \ 0 \\ \hline \end{array}$$

(C-4934)

- Final carry indicates that the answer is positive and in its true form.

$$\therefore (68.75)_{10} - (27.50)_{10} = (00101001.0100)_2$$



- Convert answer into decimal,

$$\text{Binary : } \begin{array}{ccccccccc} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & \cdot & 0 & 1 & 0 & 0 \\ \downarrow & \downarrow \\ 0 + 0 + (1 \times 2^5) + 0 + (1 \times 2^3) + 0 + 0 + (1 \times 2^0) + 0 + (1 \times 2^{-2}) + 0 + 0 \end{array} \quad (\text{C-4935})$$

$$N = 32 + 8 + 1 + 0.25 = 41.25$$

$$\therefore (68.75)_{10} - (27.50)_{10} = (41.25)_{10} \quad \dots\text{Ans.}$$

Ex. 3.5.6 : Perform 2's complement arithmetics of :

1. $(7)_{10} - (11)_{10}$
 2. $(-7)_{10} - (11)_{10}$
 3. $(-7)_{10} + (11)_{10}$
- May 15, 6 Marks**

Soln. :

$$1. \quad (7)_{10} - (11)_{10} :$$

Step 1 : Convert both numbers to their binary form :

$$(7)_{10} = (0111)_2 \text{ and } (11)_{10} = (1011)_2$$

Step 2 : Obtain 2's complement of $(11)_{10}$:

(C-8295)

Decimal	Binary	1's complement	2's complement
$(11)_{10}$	$(1011)_2$	(0100)	(0101)

Step 3 : Add $(7)_{10}$ to 2's complement of $(11)_{10}$:

(C-6129)

$$(7)_{10} : \begin{array}{r} 0 \ 1 \ 1 \ 1 \\ + 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

$$\text{2's complement of } (11)_{10} : \begin{array}{r} + 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

Carry 1 1 1

$$\boxed{0} \ 1 \ 1 \ 0 \ 0$$

"0" indicates negative sign and answer in 2's complement form

Step 4 : Convert the answer into its true form :

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \\ - 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 1 \ 1 \end{array} \quad \text{Invert all bits}$$

0 1 0 0 Answer in true form

(C-6130)

Convert answer into decimal $-(0100)$ i.e. $(-4)_{10}$

$$\therefore (7)_{10} - (11)_{10} = -(4)_{10} \quad \dots\text{Ans.}$$

$$2. \quad (-7)_{10} - (11)_{10} :$$

Step 1 : Convert both numbers to their 2's complement :

Decimal	Binary	2's complement
$(7)_{10}$	$(00000111)_2$	(11111001)
$(11)_{10}$	$(00001011)_2$	(11110101)

Step 2 : Add 2's complement of $(7)_{10}$ and $(11)_{10}$:

$$\text{2's complement of } (7)_{10} : \begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ + 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

$$\text{2's complement of } (11)_{10} : \begin{array}{r} + 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

$$\text{Carry : } \begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline \end{array}$$

$$\boxed{1} \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0$$

Result is negative and in 2's complement form

(C-6131)

Step 3 : Convert answer to its true form :

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \\ - 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

Invert bits

$$00010010 = -(18)_{10}$$

(C-6132)

$$\therefore (-7)_{10} - (11)_{10} = -(18)_{10} \quad \dots\text{Ans.}$$

$$3. \quad (-7)_{10} + (11)_{10} :$$

Step 1 : Convert $(7)_{10}$ to its 2's complement :

Decimal	Binary	2's complement
$(7)_{10}$	$(0111)_2$	(1001)

Step 2 : Add 2's complement of $(7)_{10}$ and $(11)_{10}$:

$$\text{2's complement of } (7)_{10} : \begin{array}{r} 1 \ 0 \ 0 \ 1 \\ + 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$(11)_{10} : \begin{array}{r} + 1 \ 0 \ 1 \ 1 \\ \hline \end{array}$$

$$\text{Carry } 1 \ 1$$

$$\boxed{1} \ 0 \ 1 \ 0 \ 0$$

"1" indicates positive result and answer is in true form

(C-6133)

$$\text{Answer} = (0100)_2 = (4)_{10}$$

$$\therefore (-7)_{10} + (11)_{10} = + (4)_{10} \quad \dots\text{Ans.}$$

Ex. 3.5.7 : Do the following : $(7F)_{16} - (5C)_{16}$ using 2's complement method. **May 17, 2 Marks**

Soln. :

$$1. \quad (7F)_{16} - (5C)_{16} \text{ using 2's complement method :}$$

Step 1 : Convert hex to binary :

$$(7F)_{16} = (0111 1111)_2 \text{ and}$$

$$(5C)_{16} = (0101 1100)_2$$

Step 2 : Obtain 2's complement of $(5C)_{16}$:

$$(01011100)_2 \xrightarrow{\text{1's complement}} (10100011)_2 \xrightarrow{\text{Add 1}} (10100100)_2$$

(C-6371)

**Step 3 : Add $(7F)_{16}$ and 2's complement of $(5C)_{16}$:**

$$(7F)_{16} : \begin{array}{ccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$\text{2's complement of } (5C)_{16} : \begin{array}{ccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

Discard carry

(C-5909)

$$(7F)_{16} - (5C)_{16} = (23)_{16} \quad \dots\text{Ans.}$$

3.5.2 Subtraction of Signed Binary Numbers :

- Subtraction of two signed binary numbers when the negative numbers are in the 2's complement form can be performed as follows :

Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign bit position should be discarded.

- We can use this procedure because a subtraction is equivalent to an addition if the sign of the subtrahend is changed.
- This is demonstrated as follows :

$$\begin{array}{rcl} \text{Minuend} & \rightarrow & \pm A \\ \text{Subtrahend} & \rightarrow & -(+B) \quad \text{i.e. } (\pm A) - (+B) = (\pm A) + (-B) \\ & & (\pm A) + (-B) \\ & & \text{And } (\pm A) - (-B) = (\pm A) + (+B) \end{array} \quad (\text{C-6386})$$

- Consider the subtraction $(-7) - (-12) = +5$ in binary with eight bits.
- The numbers (-7) and (-12) are expressed in the 2's complement form as (11111001) and (11110100) respectively.
- The subtraction of these numbers is changed to addition by taking the 2's complement of subtrahend (-12) to $(+12)$.
- The 2's complement of $-12 = (00001100)$. Then the addition is performed as follows :

$$\begin{array}{r} -7 \quad 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\ +12 \quad 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \\ \hline +5 \quad 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \end{array}$$

Discard Answer is +5
(C-6134)

- Discarding the final carry we get the answer of + 5.

Conclusion :

- The binary numbers in the signed complement system are added and subtracted by using the rules of the conventional addition and subtraction procedures for the unsigned numbers.
- Due to this the computers need only one common hardware circuit to handle the addition as well as subtraction.
- The user must interpret the results of such addition or subtraction differently depending on whether it is assumed that the numbers are signed or unsigned.

3.6 Floating Point Representation :

- We need to use many bits in order to represent very large **integer** (whole) numbers.
- The same problem is faced when numbers with both integer and fractional parts such as 64.8629 needs to be represented.
- The **floating point number system** is the remedy to this problem. It is based on scientific notation and is capable of representing very large and very small numbers without increasing the number of bits.
- This system can be used for representing numbers which have both integer and fractional parts. It uses power of ten.

3.6.1 Parts of Floating Point Numbers :

- The floating point numbers are also called as the real numbers. They consist of two parts and a sign. The two parts are :
 - Mantissa
 - Exponent
- Mantissa** is the part of a floating point number which represents the magnitude of the number.
- Exponent** is the part of the floating point number that represents the number of places that the decimal point (or binary point) is to be moved.

Ex. 3.6.1 : Convert the following decimal number into a floating point number.
 $N = 541, 369, 841$.

Soln. :

- The given number can be represented in the floating point number as follows :



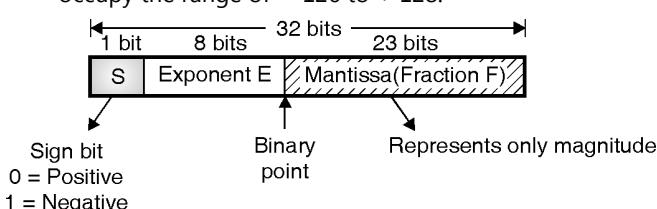
- Thus mantissa is the magnitude and exponent (9) represents the number of places that the decimal point is moved.

3.6.2 Binary Floating Point Numbers :

- For binary floating point numbers, the format is defined by ANSI / IEEE standard 754 – 1985 in the following three forms :
 - Single precision floating point binary numbers.
 - Double precision floating point binary numbers.
 - Extended precision floating point binary numbers.

3.6.3 Single Precision Floating Point Binary Numbers :

- Fig. 3.6.1 shows the 32 bit standard format for a single precision floating point binary number.
- The MSB is assigned to indicate the sign (S). The next eight bits indicate the exponent (E) and the last 23 bits are used for representing the mantissa or the fractional part (F).
- In the mantissa (F) part, the binary point is assumed to placed to the left of the 23 bits as shown in Fig. 3.6.1.
- The eight bit exponent (E) actually represents a biased exponent. That means this exponent is obtained by adding 127 to the actual exponent.
- The bias is used to allow very large or very small numbers without requiring a separate sign bit for the exponent. T
- he biased exponent will allow the actual exponent to occupy the range of - 126 to + 128.



(C-1790) Fig. 3.6.1 : Format of single precision floating point binary numbers

Ex. 3.6.2 : Represent the following binary number using a single precision floating point format.
 $N = (1001010011101)_2$.

Soln. :

Step 1 : Represent the given number in fractional form :

- The given binary number contains 13 bits. Let us express it as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiply by appropriate power of two as follows :

$$\begin{aligned} N &= 1001010011101 \quad \dots \text{Given 13 bit number} \\ &\quad \text{Binary point} \\ &= 1 \cdot 001010011101 \times 2^{12} \dots \text{Exponent} \\ &\quad \downarrow \\ &\quad \text{12 bits mantissa} \end{aligned} \quad \dots (1)$$

(C-4922)

Step 2 : Decide the values of S, E and F :

- Assuming the number to be the positive one the sign bit will have a zero value.

$$\therefore S = 0$$

- The exponent is actually 12 because we have shifted the binary point by 12 places. In order to get the biased exponent we have to add 127 to the actual exponent.

$$\therefore \text{Biased exponent } E = \text{Actual exponent} + 127$$

$$\therefore E = 12 + 127 = (139)_{10}$$

- Convert the biased exponent to binary

(C-6257)

2	139	
2	69	1
2	34	1
2	17	0
2	8	1
2	4	0
2	2	0
2	1	0
	0	1

LSB ↑
 $\therefore E = (10001011)_2$
MSB ↓

- The mantissa is the fractional part (F) of the binary number. Because there is always a 1 to the left of the binary point as shown in Equation (1), it is not included in the mantissa.

- The 12 bits in the mantissa are same as those in Equation (1) but eleven zeros are appended to these 12 bits to form the 23 bit mantissa as follows :



$$F = \begin{array}{c} \text{eleven zeros} \\ \boxed{001010011101} \quad \boxed{000000000000} \\ \text{L } 12 \text{ bits from Equation (1)} \end{array} \quad \begin{array}{l} 23 \text{ bit mantissa} \\ \text{(C-4922(a))} \end{array}$$

Step 3 : Write the complete floating point number :

- The complete single precision floating point number is as follows :

(C-8299)

S	Exponent E	Mantissa(Fraction F)
0	10001011	00101001110100000000000

8 bits 23 bits

Conversion from floating point to binary :

- Now let us see the conversion from floating point number to binary number.
 - Use the formulae given below for the same.
- Binary number = $(-1)^S (1 + F) (2^{E-127})$... (2)
- The values of S, F and E are to be obtained from the given floating point number.

Ex. 3.6.3 : Convert the following floating point number into equivalent binary number.

(C-8300)

S	Exponent E	Mantissa(Fraction F)
1	10001011	00101001110100000000000

Soln. :

From the given floating point number we get,

$$S = 1, E = 10001011, F = 001010011101$$

$$\therefore \text{Binary number} = (-1)^S (1 + F) (2^{E-127})$$

$$\begin{aligned} \therefore \text{Binary number} &= (-1)^1 (1 + 0.001010011101) (2^{139-127}) \\ &= -(1.001010011101) \times 2^{12} \\ &= -1001010011101 \end{aligned}$$

$$\text{But } E = \begin{array}{c} 1 \\ \downarrow \\ 0 \\ \downarrow \\ 0 \\ \downarrow \\ 0 \\ \downarrow \\ 1 \\ \downarrow \\ 0 \\ \downarrow \\ 1 \\ \downarrow \\ 1 \end{array} \quad \begin{array}{l} 128 \\ + \\ 8 \\ + \\ 2 \\ + \\ 1 \end{array} = 139$$

(C-1791)

Note :

- Since the biased exponent (E) can take any value between - 126 to + 128, it is possible to express extremely large and small numbers using the floating point number systems.

- A 32 bit floating point number can replace a binary integer number having 129 numbers.
- The number 0.0 is represented by all 0s and infinity is represented by all 1s in the exponent and all zeros in the mantissa.

Ex. 3.6.4 : Represent the following decimal numbers in single precision floating point format :

$$1. \quad 255.5 \quad 2. \quad 110.65$$

May 15, 6 Marks**Soln. :**

$$1. \quad (255.5)_{10} :$$

Step 1 : Convert decimal to binary :

2	255	1	LSB
2	127	1	
2	63	1	
2	31	1	
2	15	1	
2	7	1	
2	3	1	
2	1	1	MSB
	0		

$$\therefore (255)_{10} = (11111111)_2$$

$$\begin{array}{ccccccc}
 & \text{Base} & \text{Product} & \text{Carry} & & & \\
 0.5 & \times & 2 & = & 1.0 & \curvearrowright & \text{MSB} \\
 0.0 & \times & 2 & = & 0.0 & \curvearrowright & \text{LSB}
 \end{array}$$

$$\therefore (0.5)_{10} = (0.10)_2$$

(C-5125)

$$\therefore (255.5)_{10} = (11111111.10)_2 \quad \dots (1)$$

Step 2 : Decide the values of S, E and F :

$$N = 1 \cdot \underbrace{11111110}_{\substack{\text{Binary} \\ \text{point}}} \times 2^7 \quad \dots \text{Exponent}$$

Mantissa

(C-5126)

- Assuming the number to be the positive one the sign bit will have a zero value.
- $\therefore S = 0$
- Exponent E = Actual exponent + 127 = 7 + 127 = $(134)_{10}$



- Convert the biased exponent to binary

$$E = (134)_{10} = (10000110)_2$$

- The mantissa is the fractional part of binary number.

To form 23 bit mantissa, fourteen zeros are appended.

$$\therefore F = 1111111000000000000000000$$

Step 3 : Write the complete floating point number :

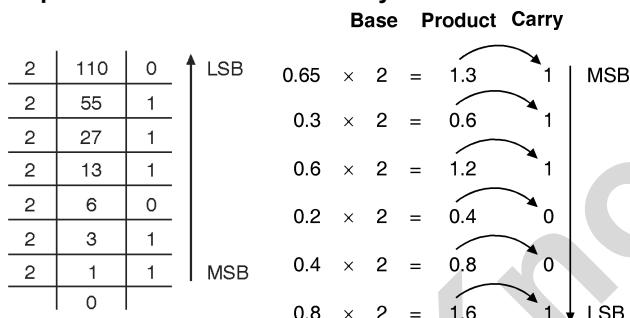
(C-8319)

S	Exponent E	Mantissa(Fraction F)
0	10000110	1111111000000000000000000

8 bits 23 bits

2. $(110.65)_{10}$:

Step 1 : Convert decimal to binary :



$$(110)_{10} = (1101110)_2$$

$$\therefore (0.65)_{10} = (0.111001)_2$$

(C-5127)

$$\therefore (110.65)_{10} = (1101110.111001)_2$$

Step 2 : Decide the values of S, E and F :

$$N = 1 \cdot 101110111001 \times 2^6 \quad \dots \text{Exponent}$$

Binary point

Mantissa

(C-5128)

- Assuming the number to be the positive one the sign bit will have a zero value.

$$\therefore S = 0$$

$$\begin{aligned} \text{Exponent } E &= \text{Actual exponent} + 127 = 6 + 127 \\ &= (133)_{10} \end{aligned}$$

$$\therefore E = (133)_{10} = (10000101)_2$$

- The mantissa is fractional part of binary number.

- To form 23 bit mantissa, 11 zeros are appended.

$$\therefore F = 10111011100100000000000$$

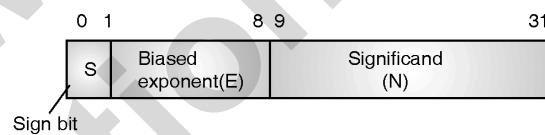
Step 3 : Write the floating point number : (C-8320)

S	Exponent E	Mantissa(Fraction F)
0	10000101	10111011100100000000000

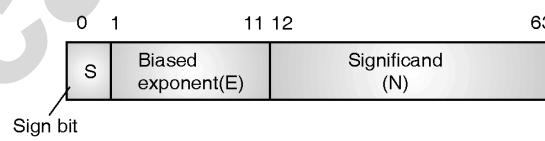
8 bits 23 bits

3.7 IEEE-754 Standard for Representing Floating Point Numbers :

- Representation of floating point number discussed in section 3.6 has many subtle problems.
- IEEE floating point standards addresses a number of such problems.
- Zero has definite representation in IEEE format.
- $\pm \infty$ has been represented in IEEE format. A + ∞ indicated that the result of an arithmetic expression is too large to be stored.
- If an underflow occurs, implying that a result is too small to be represented as a normalized number, it is encoded in a denormalized scale.
- Fig. 3.7.1 gives the representation of floating point numbers.



(a)Single precision = 32 bits



(b)Double precision = 64 bits

- base = 2
- Significand is in normalized form i.e. the first bit is 1 and it is hidden.
- S is sign bit.

(co 2.51) Fig. 3.7.1 : IEEE standard format

(a) Single precision (32 bits) :

	Exponent (E)	Significant (N)	Value/comments
	255	Not equal to 0	Does not represent a number
	255	0	$-\infty$ or $+\infty$ depending on sign bit
Normalized scale	$0 < E < 255$	Any	$\pm (1.N) 2^{E-127}$
Denormalized scale	0	Not equal to 0	$\pm (0.N) 2^{-126}$
	0	0	± 0 depending on sign bit



(b) Double precision (64 bits) :

	Exponent (E)	Significant (N)	Value/comments
	2047	Not equal to 0	Does not represent a number
	2047	0	$-\infty$ or $+\infty$ depending on sign bit
Normalized scale	$0 < E < 2047$	Any	$\pm(1.N)2^{E-1023}$
Denormalized scale	0	Not equal to 0	$\pm(0.N)2^{-1022}$
	0	0	± 0 depending on sign bit

Fig. 3.7.2 : Values of floating point numbers as per IEEE format

Ex. 3.7.1 : Represent $(20.625)_{10}$ in both single precision as well as double precision

Soln. :

Step 1 : Convert to binary :

16	20		
16	12	$C^8 \uparrow$	$(20)_{10} = (C8)_{16}$
	0		$= (110100)_2$

$$0.625 \times 16 = 10 = A$$

$$\therefore (0.625)_{10} = (0.A)_{16} = (01010)_2$$

$$(20.625)_{10} = (110100.1010)_2$$

Step 2 : Normalization :

$$110100101 \times 2^7$$

Step 3 : Calculate biased exponent :

For single precision

$$\text{Biased exp} = \text{exp} + 127 = (134)_{10} = (100110)_2$$

For double precision :

$$\begin{aligned} \text{Biased exp} &= \text{exp} + 1023 = (1030)_{10} \\ &= (10000110)_2 \end{aligned}$$

Step 4 : Find representation :

Single precision : (C-8301)

31	30	23 22	0
S	Biased exponent	Mantissa	
0	100110	101001010	

Double precision :

(C-8302)

63	62	52 51	0
S	Biased exponent	Mantissa	
0	10000110	101001010	

Ex. 3.7.2 : Represent $-(0.125)_{10}$ in both single precision as well as double precision

Soln. :

Step 1 :

$$(0.125)_{10} = 2 = (0.010)_2$$

Step 2 : Normalization :

$$10 \times 2^{-3}$$

Step 3 : Single precision :

$$\begin{aligned} \text{Biased exp} &= -3 + 127 = (124)_{10} \\ &= (01110)_2 \end{aligned}$$

Double precision

$$\begin{aligned} &= -3 + 1023 = 1020 \\ &= (01111110)_2 \end{aligned}$$

Single precision :

(C-8303)

31	30	23 22	0
S	Biased exponent	Mantissa	
1	011110	0...	

Double precision :

(C-8304)

63	62	52 51	0
S	Biased exponent	Mantissa	
1	011110	0...	

Note : The biased exponent can be in the range 1-254 for single precision (exponent range is -126 to +127) and 1-2046 for double precision (exp range -1022 to +1023). The biased exp values 0 and 255 for single precision & (0 and 2047 for double precision) are used to represent zero, ∞ , denormalized form, NaN (Not a Number.)

Ex. 3.7.3 : Represent $(178.1875)_{10}$ in single and double precision floating point format.

Soln. :

Convert given decimal number into its equivalent binary

$$178 = 101110$$



$$.1875 \times 2 = 0.3750$$

$$.3750 \times 2 = 0.750$$

$$.750 \times 2 = 1.50$$

$$.50 \times 2 = 1.00$$

$$\therefore (178.1875)_{10} = (101110.011)_2$$

(a) Single precision format :

- In IEEE format, the value of a number for given exponent (E) and significant (N) is given by $(1.N) 2^{E-127}$ in order to represent $(101110.011)_2$, we must convert it into the form $(1.N) 2^{E-127}$.

$$10111.011 = 1.0111011 \times 2^5$$

$$5 = E - 127$$

$$\therefore E = 127 + 5 = 132$$

$$(132)_{10} = (10010)_2$$

0	1	8	9	31
0	10000100	01111011000.....000		

Sign bit Exponent(E) Significand(N)

(b) Double precision format :

- In IEEE double precision format, the value of a number for given exponent (E) and significant (N) is given by $(1.N) 2^{E-1023}$.

In order to represent $(101110.011)_2$, we must convert it into the form $(1.N) 2^{E-1023}$.

$$10111.011 = 1.0111011 \times 2^5$$

$$5 = E - 1023$$

$$\therefore E = 1023 + 5 = 1028$$

$$(1028)_{10} = (1000010)_2$$

0	1	11	12	63
0	10000000100	011110110000.....0000		

Sign bit Exponent(E) Significand(N)

Ex. 3.7.4 : Represent $(309.1875)_{10}$ in single precision and double precision format.

Soln. :

Convert given decimal number into its equivalent binary.

$$(309)_{10} = (10110101)_2$$

$$.1875 \times 2 = 0.3750$$

$$.3750 \times 2 = 0.750$$

$$.750 \times 2 = 1.500$$

$$.500 \times 2 = 1.00$$

$$\therefore 309.1875 = 10110101.011$$

(a) Single precision format :

- In IEEE single precision format, the value of a number for given exponent (E) and significant (N) is given by $(1.N) 2^{E-127}$.

In order to represent 10110101.011 , we must convert it into the form $(1.N) 2^{E-127}$.

$$10110101.011 = 1.0110101011 \times 2^8$$

$$8 = E - 127$$

$$\therefore E = 127 + 8 = 135$$

$$(135)_{10} = (10011)_2$$

0	1	8	9	31
0	10000111	001101010011.....0000		

Sign bit Exponent(E) Significand(N)
(0 for positive number)

(b) Double precision format :

- In IEEE double precision format, the value of a number for given exponent (E) and significant (N) is given by,

$$(1.N) 2^{E-1023}$$

In order to represent 10110101.011 , we must convert it into the form $(1.N) 2^{E-1023}$.

$$10110101.011 = 1.0110101011 \times 2^8$$

$$8 = E - 1023$$

$$\therefore E = 1023 + 8 = 1031$$

$$(1031)_{10} = (1000011)_2$$

0	1	11	12	63
0	10000000111	001101010011000.....0000		

Sign bit Exponent(E) Significand(N)
(0 for positive)

Ex. 3.7.5 : Express $(35.25)_{10}$ in the IEEE single precision standard of floating point representation.

Soln. :

(35.25)₁₀

Step 1 : $(35)_{10} = (23)_{16} = (010011)_2$

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

$$\therefore (35.25)_{10} = (10011.01)_2$$

Step 2 : $(35.25)_{10} = (1.001101)_2 \times 2^5$

**Step 3 :**

$$\text{Biased exponent} = 127 + 5 = (132)_{10} = (100\ 010)_2$$

5 Biased exponent mantissa **(C-8305)**

31	30	23 22	0
0	10010	00110100....	

Ex. 3.7.6 : Convert $(127.125)_{10}$ in IEEE-754 single and double precision floating point representation.

Dec. 16, 10 Marks

Soln. :

Step 1 :

16	127	15	→ F
16	7	7	→ 7
0			

(C-8306)

$$\therefore (127)_{10} = (7F)_{16} = (11111111)_2$$

$$0.125 \times 16 = 5.00$$

$$\therefore (0.125)_{10} = (2)_{16} = (0.01)_2$$

$$\therefore (0.125)_{10} = (2)_{16} = (0.00)_2$$

Step 2 : $(127 - 125)_{10} = (011\ 111.010)_2$

$$= (1.111101)_2 \times 2^6$$

Step 3 : Biased exponent :

- For single precision $\Rightarrow 127 + 6 = (133)_{10} = (100101)_2$

- For double precision $\Rightarrow 1023 + 6 - (1029)_{10} = (10000101)_2$

(a) Single precision :

(C-8307)

31	30	23 20	0
S	Biased exponent	Mantissa	

0 100101 1111010.....

(b) Double precision :

(C-8308)

63	62	52 51	0
S	Biased exponent	Mantissa	

0 10000101 1111010.....

3.8 Introduction to Boolean Algebra :

- In the decimal system which is used in our day-to-day life, the arithmetic operations such as addition, subtraction, multiplication, division, square, square root etc. are used to solve arithmetic equations.

- In 1850s, the Irish mathematician George Boole developed a mathematical system for formulating logic statements with symbols so that problems can be written and solved in a manner similar to ordinary algebra.
- This is called Boolean algebra and it is extremely useful in the design and analysis of digital systems.
- Boolean algebra is used to write or simplify the logical expressions.

3.8.1 Basic Logical Operations (Logic Variables) :

- To solve or simplify the logical expressions, used in digital circuits we need to use "logical operators". The three main logic operators are :
 1. AND operator.
 2. OR operator.
 3. NOT operator (Inverter).

3.8.2 NOT Operator (Inversion) :

- The NOT operation represents a process of logical inversion called as complementing.
- It is implemented by using an inverter.
- The NOT operation is denoted by a bar ($\bar{}$) over the variable which is being inverted.

$$\therefore \bar{A} = \text{NOT } A \quad \dots \text{Logical inversion.}$$

3.8.3 AND Operator :

- The AND operation produces a high (1) output only if all the inputs of a logic circuit are high (1).
- The "AND" operator represents logical multiplication. It is denoted by a dot between the two variables to be multiplied. i.e.

$$A \cdot B \quad \dots \text{Logical multiplication}$$

- However sometimes this dot is not used and we denote the logical multiplication of A and B as AB.

3.8.4 OR Operator :

- The OR-operation produces a HIGH (1) output when any one or all the inputs of a logic circuit are HIGH(1).



- The "OR" operator represents logical addition. It is denoted by a (+) sign between the two variables to be added.
- If A and B are to be added in a logic circuit then it is represented as,
A + B ...Logical addition
- And it is to be read as "A OR B".

3.8.5 Logic Gates :

- Logic gates are the logic circuits which act as the basic building blocks of any digital system.
- It is an electronic circuit having one or more than one inputs and only one output.
- The relationship between the input and the output is based on a "certain logic".
- Depending on this logic, the gates are named as NOT gate, AND gate, OR, NAND, NOR etc.

Truth table :

- The operation of a logic gate or a logic circuit can be best understood with the help of a table called "Truth Table".
- The truth table consists of all the possible combinations of the inputs and the corresponding state of output produced by that logic gate or logic circuit.

Boolean expression :

- The relation between the inputs and the outputs of a gate can be expressed mathematically by means of the **Boolean Expression**.
- Let us now discuss the operation of various logic gates.

3.8.6 Gates, Symbols and Boolean Expression :

- In order to understand Boolean algebra , we need to use the gates.
- So the symbols and Boolean expressions should be known to us. Table 3.8.1 gives information.

(C-196) Table 3.8.1 : Various logic gates

Sr. No.	Name of gate	Boolean Expression	Logical Operation
1.	NOT gate or inverter 	$Y = \bar{A}$	Inversion
2.	AND gate 	$Y = AB$	Logical multiplication
3.	OR gate 	$Y = A + B$	Logical addition
4.	NAND gate 	$Y = \overline{AB}$	NOT AND
5.	NOR gate 	$Y = \overline{A+B}$	NOT OR
6.	Exclusive OR 	$Y = A \oplus B$	Addition/Subtraction
7.	Exclusive NOR 	$Y = \overline{A \oplus B}$	NOT EXOR

Note : A and B are the inputs whereas Y denotes the output.

3.8.7 Postulates :

SPPU : Dec. 04, Dec. 08

University Questions

Q. 1 With suitable examples, explain the following in Boolean algebra :

1. Commutative law
2. Associative law
3. Distributive law (Dec. 04, Dec. 08, 3 Marks)

2. Associative law :

- For a given set "S" a binary operator * can be said to be associative if the following equation is satisfied.

$$(A * B) * C = A * (B * C) \text{ for all } (A, B, C) \in S \quad \dots(3.8.1)$$

- Note that * can be an operator such that · (product) or + (addition) etc.

3. Commutative law :

For a given set "S" a binary operator * is said to be commutative, if the following equation is satisfied.

$$A * B = B * A \text{ for all } A, B \in S \quad \dots(3.8.2)$$



5. Inverse :

- A set S having an identity element x with respect to a binary operator $*$ is said to have an inverse if for every A which belongs to S , there exists another element B which also belongs to S such that,

$$A * B = x$$

$$A + (-A) = 0$$

Given element Operator Identity element
Another element in the set

(C-128)

- Take the example of set of integers (\mathbb{I}) for which 0 is the identity element with respect to $+$ operation that means $x = 0$ and $* = +$. Hence the inverse of an element A would be $-A$ because only then the condition $A * B = x$ is satisfied as follows :

6. Distributive law :

If $*$ and \cdot are two binary operators working on a set S , then $*$ is said to be distributive over \cdot if the following condition is satisfied.

$$A * (B \cdot C) = (A * B) \cdot (A * C) \quad \dots(3.8.3)$$

- The operators and postulates of such a field have the following meanings :
 1. We define the binary operator $+$ as addition.
 2. The identity element for $+$ operator is 0.
 3. We define the additive inverse as subtraction.
 4. The binary operator \cdot is defined as multiplication.
 5. The identity element for \cdot operator is 1.
 6. We define the multiplicative inverse of A as $1/A$ and it is called as division.
 7. The only distributive law that is applicable is that of \cdot over $+$. That means,

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

3.9 Definition of Boolean Algebra :

Definition :

- Boolean algebra is used to analyze and simplify the digital (logic) circuits.
- Since it uses only the binary numbers i.e. 0 and 1 it is also called as "Binary Algebra", or "Logical Algebra".
- It was invented by George Boole in the year 1854.
- The rules of Boolean algebra are different from those of the conventional algebra in the following manner :

1. Symbols used in Boolean algebra (usually letters) do not represent numerical values.
2. Arithmetic operations (addition, subtraction, division etc.) are not performed in boolean algebra. Also there are no fractions, negative numbers, square, square root, logarithms, imaginary numbers etc.
3. Third and most important point is Boolean algebra allows only two possible values ("0" to "1") for any variable.

Rules in boolean algebra :

- There are some rules to be followed while using a Boolean algebra, these are :
- 1. Variables used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- 2. Complement of a variable is represented by a overbar (\bar{A}). Thus complement of variable B is represented as \bar{B} . Thus if $B = 0$ then $\bar{B} = 1$ and if $B = 1$ then $\bar{B} = 0$.
- 3. ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as $A + B + C$.
- 4. Logical ANDing of the two or more variables is represented by writing a dot between them such as $A \cdot B \cdot C \cdot D \cdot E$. Sometimes the dot may be omitted like ABCDE.

3.9.1 Boolean Postulates and Laws :

- Boolean algebra is an algebraic structure, which is defined by a set of elements and two binary operators namely $(+)$ and (\cdot) if and only if it satisfies the postulates given below :

1(a) Closure with respect to operator $(+)$:

When operator $(+)$ i.e. OR is used over two binary elements in a given set, it produces a unique binary element e.g. $1 + 0 = 1$, $0 + 0 = 0$.

1(b) Closure with respect to operator (\cdot) :

When operator (\cdot) i.e. AND is used over two binary elements in a given set, it produces a unique binary element e.g. $1 \cdot 0 = 0$, $1 \cdot 1 = 1$ etc.

2(a) An identity element with respect to $(+)$, designated by 0 :

This is because for any A , the following expression is always true.

$$A + 0 = 0 + A = A$$



2(b) An identity element with respect to (\cdot), designated by 1 :

This is because for any A the following expression is always true.

$$A \cdot 1 = 1 \cdot A = A.$$

3(a) Commutative law with respect to (+) :

$$A + B = B + A$$

3(b) Commutative law with respect to (\cdot) :

$$A \cdot B = B \cdot A$$

4(a) (\cdot) is distributed over (+) :

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

4(b) (+) is distributed over (\cdot) :

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

5. For every binary element A, there exists an element called complement of A (it is denoted by \bar{A}) such that,

$$A + \bar{A} = 1 \text{ and } A \cdot \bar{A} = 0$$

Note that if $A = 0$ then $\bar{A} = 1$ and if $A = 1$ then $\bar{A} = 0$.

6. In a set of binary elements, there always exist atleast two elements A and B such that $A \neq B$. If $A = 0$ then $B = 1$ and vice versa.

3.10 Two Valued Boolean Algebra :

SPPU : Dec. 04, Dec. 08

University Questions

Q. 1 With suitable examples, explain the following in Boolean algebra :

1. Commutative law 2. Associative law
3. Distributive law (Dec. 04, Dec. 08, 3 Marks)

- It is possible to formulate many Boolean algebras on the basis of elements we choose and the rules of operation.
- In this book we are going to deal with only the two valued Boolean algebra.
- It is called two valued because it has only two elements 0 and 1 with binary operators AND (\cdot), OR (+) and NOT (inversion).
- The following tables show the truth tables for these operators.

Table 3.10.1(a) : AND operator

Inputs	Output	
A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.10.1(b) : OR operator

Inputs	Output	
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.10.1(c) : NOT operator

Input	Output
A	\bar{A}
0	1
1	0

- We are now going to demonstrate that Huntington postulates are valid for the two valued Boolean Algebra. Let $S = \{0, 1\}$ and the operators be $+$, \cdot and NOT.

1. Closure :

- The closure is obvious from Tables 3.10.1(a), (b) and (c) which shows that the result of any operation (AND, OR, NOT) is either 0 or 1, and we know that $0, 1 \in S$.

2. Identity elements :

- From Tables 3.10.1(a) and (b), we can write that,

$$0 + 0 = 0 \quad 0 + 1 = 1 + 0 = 1$$

$$1 \cdot 1 = 1 \quad 1 \cdot 0 = 0 \cdot 1 = 0$$

- From these expressions, two identity elements are established which are 0 for $+$ and 1 for \cdot as defined by postulate 2.

3. The commutative laws :

- The commutative laws are

$$0 + 1 = 1 + 0 \quad \text{and} \quad 0 \cdot 1 = 1 \cdot 0$$

- These laws are satisfied due to the symmetry of Tables 3.10.1(a) and (b). This is illustrated by using the following truth table.

(C-6135) **Table 3.10.2 : Verification of the commutative law**

Inputs		A · B	B · A	A + B	B + A
A	B	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	1	1	1

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

4. The distributive law :

- (a) The distributive law of $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ can be proved by using the truth tables as shown below.



(C-6136)Table 3.10.3 : Verification of the distributive law

Inputs			B + C	A · (B + C)	AB	AC	AB + AC
A	B	C					
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

$$A \cdot (B + C) = AB + AC$$

- (b) The distributive law of $+$ over \cdot can also be proved using the truth tables in the similar way.

5. Inverse :

- From Table 3.10.1(b) it is easily seen that,

$$A + \bar{A} = 1 \quad \text{i.e. } 0 + 1 = 1 \text{ and } 1 + 0 = 1$$

And from Table 3.10.1(a) it can be shown that

$$A \cdot \bar{A} = 0 \quad \text{i.e. } 0 \cdot 1 = 0 \text{ and } 1 \cdot 0 = 0$$

This verifies postulate 5.

6. Postulate 6 says that there exists at least two elements A and B such that $A \neq B$. As Boolean algebra has two distinct elements 0 and 1 (A and B), this postulate is getting satisfied because $0 \neq 1$ ($A \neq B$),

7. Associative law :

- This law states that the order in which the logic operations are performed is not important because for any order the effect is the same.

$$\text{i.e. } (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

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

- The associative law can be verified by referring to the following truth table.

(C-6137)Table 3.10.4 : Verification of the associative law

Inputs			B · C	A · (B · C)	A · B	(A · B) · C
A	B	C				
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	1	0
1	1	1	1	1	1	1

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

- Similarly we can verify the other statement i.e.

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

3.11 Basic Theorems and Properties of Boolean Algebra :

3.11.1 Duality :

- According to the duality theorem the following conversions are possible in a given Boolean expression :

 - Change each AND operation to an OR operation.
 - Change each OR operation to an AND operation.
 - Complement any 1 or 0 appearing in the expression.

- Duality theorem is sometimes useful in creating new expressions from the given Boolean expressions.
- For example if the given expression is $A + 1 = 1$ then replace the OR (+) operation by AND (·) operation and take complement of 1 to write the dual of the given relation as,

$$A \cdot 0 = 0$$

- The dual of $A(B + C) = AB + AC$ is given by,

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

- It is possible to verify this theorem by constructing truth tables for both the sides of equations.

3.11.2 Basic Theorems :

SPPU : May 12

University Questions

- Q. 1 State and prove any two theorems of Boolean algebra.
(May 12, 4 Marks)

1. AND Laws :

These laws are related to the AND operation therefore they are called as "AND" laws. The AND laws are as follows :

A · 0 = 0 :

- That means if one input of an AND operation is permanently kept at the LOW (0) level, then the output is zero irrespective of the other variable.

A · 1 = A :

- That means if one input of an AND operation is HIGH (1) permanently then the output is always equal to the other variable.
- So if $A = 0$, then $Y = 0 \cdot 1 = 0$ i.e. $Y = A$ and if $A = 1$, then $Y = 1 \cdot 1 = 1$ so $Y = A$.

A · A = A :

That means if both the inputs in an AND operation have the same value either "0" or "1" then the output will also have the same value as that of the input.

 **$A \cdot \bar{A} = 0$:**

- This law states that the result of an "AND" operation on a variable (A) and its complement (\bar{A}) is always LOW (0).
- If $A = 0$ then $\bar{A} = 1$ and $Y = 0 \cdot 1 = 0$ whereas if $A = 1$ then $\bar{A} = 0$ and $Y = 1 \cdot 0 = 0$.

Summary of AND Laws :

- | | |
|--------------------|--------------------------|
| 1. $A \cdot 0 = 0$ | 3. $A \cdot A = A$ |
| 2. $A \cdot 1 = A$ | 4. $A \cdot \bar{A} = 0$ |

2. OR Laws :

- These laws use the OR operation. Therefore they are called as OR laws. The OR laws are as follows :

 $A + 0 = A$:

- That means if one variable of an "OR" operation is LOW (0) permanently, then the output is always equal to the other variable.
- $B = 0$ permanently therefore for $A = 0$, $Y = 0 + 0 = 0$ i.e. $Y = A$ and for $A = 1$, $Y = 1 + 0 = 1$ i.e. $Y = A$.

 $A + 1 = 1$:

- That means if one variable of an "OR" operation is HIGH (1) permanently, then the output is HIGH (1) permanently irrespective of the value of the other variable.
- Here $B = 1$ permanently, so if $A = 0$ then $Y = 0 + 1 = 1$ and if $A = 1$ then $Y = 1 + 1 = 1$.
- Thus output remains HIGH (1) always, irrespective of the value of A .

 $A + A = A$:

- This law states that if both the variables of an OR operation have the same value either "0" or "1" then the output also will be equal to the input i.e. 0 or 1 respectively.
- For $A = 0$, $Y = 0 + 0 = 0$ i.e. $Y = A$ and for $A = 1$, $Y = 1 + 1 = 1$ i.e. $Y = A$.

 $A + \bar{A} = 1$:

- This law states that the result of an "OR" operation on a variable and its complement is always 1 (HIGH).
- If $A = 0$ then $\bar{A} = 1$ and $Y = 0 + 1 = 1$ whereas if $A = 1$ then $\bar{A} = 0$ and $Y = 1 + 0 = 1$.

Summary of OR Laws :

- | | |
|----------------|----------------------|
| 1. $A + 0 = A$ | 3. $A + A = A$ |
| 2. $A + 1 = 1$ | 4. $A + \bar{A} = 1$ |

3. INVERSION Law :

- This law uses the "NOT" operation. The inversion law states that if a variable is subjected to a double inversion then it will result in the original variable itself i.e.

$$\overline{\overline{A}} = A$$

- Inversion law is being illustrated in Fig. 3.11.1 which shows that if $A = 0$ then $\bar{A} = 1$ and $(\bar{A}) = 0 \therefore Y = A$, whereas if $A = 1$ then $\bar{A} = 0$ and $(\bar{A}) = 1 \therefore Y = A$.

(B-481) Fig. 3.11.1 : Illustration of inversion law : $\overline{\overline{A}} = A$

Table 3.11.1 : Collection of Boolean laws

Sr. No.	Name	Statement of the law
1.	Commutative Law	$A \cdot B = B \cdot A$ $A + B = B + A$
2.	Associative Law	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$ $(A + B) + C = A + (B + C)$
3.	Distributive Law	$A \cdot (B + C) = AB + AC$
4.	AND Laws	$A \cdot 0 = 0$ $A \cdot 1 = A$ $A \cdot A = A$ $A \cdot \bar{A} = 0$
5.	OR Laws	$A + 0 = A$ $A + 1 = 1$ $A + A = A$ $A + \bar{A} = 1$
6.	Inversion Law	$\overline{\overline{A}} = A$
7.	Other Important Laws	$A + BC = (A + B)(A + C)$ $\bar{A} + AB = \bar{A} + B$ $\bar{A} + \bar{A}B = \bar{A} + B$ $A + AB = A$ $A + \bar{A}B = A + B$

Ex. 3.11.1 : Obtain the dual of following Boolean equations :

1. $A + AB = A$
2. $A + \bar{A}B = A + B$
3. $A + \bar{A} = 1$
4. $(A + B)(A + C) = A + BC$.

**Soln. :****1. $A + AB = A$:**

- Replace $(+)$ by (\cdot) and (\cdot) by $(+)$ to get the dual of the given equation as :

$$A \cdot (A + B) = A \quad \dots \text{Ans.}$$
- Similarly the duals of the other expressions are shown in Table P. 3.11.1.

Table P. 3.11.1

Given expression	Dual
$A + \bar{A}B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
$(A + B)(A + C) = A + BC$	$AB + AC = A \cdot (B + C)$

3.11.3 De-Morgan's Theorems :**SPPU : Dec. 05, May 08, Dec. 09****University Questions**

- Q. 1** What are De Morgan's theorems ? How will define them in terms of your own words ?

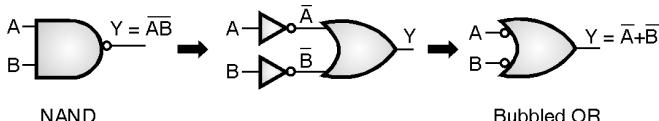
(Dec. 05, May 08, Dec. 09, 4 Marks)

- The two theorems suggested by De-Morgan and which are extremely useful in Boolean algebra are as follows :

Theorem 1 : $\overline{AB} = \bar{A} + \bar{B}$: NAND = Bubbled OR :

- This theorem states that the complement of a product is equal to addition of the complements.
- This rule is illustrated in Fig. 3.11.2.
- The Left Hand Side (LHS) of this theorem represents a NAND gate with inputs A and B whereas the Right Hand Side (RHS) of the theorem represents an OR gate with inverted inputs.
- Such an OR gate is called as "Bubbled OR". Thus we can state De-Morgan's first theorem as a NAND operation is equivalent to a bubbled OR operation.

$$\text{NAND} \equiv \text{Bubbled OR}$$

**(B-482)Fig. 3.11.2 : Illustration of De-Morgan's first theorem**

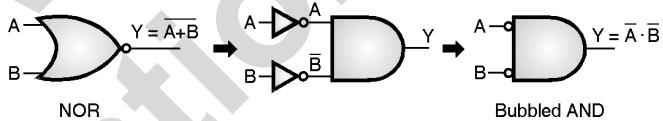
- This theorem can be verified by writing a truth table as shown in Fig. 3.11.3.

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

LHS $\overline{AB} = \bar{A} + \bar{B}$ RHS(C-6138) **Fig. 3.11.3 : Verification of the theorem $\overline{AB} = \bar{A} + \bar{B}$** **Theorem 2 : $A + B = \bar{\bar{A}} \cdot \bar{\bar{B}}$: NOR = Bubbled AND :**

- This theorem is illustrated in Fig. 3.11.4. The LHS of this theorem represents a NOR gate with inputs A and B whereas the RHS represents an AND gate with inverted inputs.
- Such an AND gate is called as "Bubbled AND". Thus we can state De-Morgan's second theorem as a NOR function is equivalent to a bubbled AND function.

$$\text{NOR} \equiv \text{Bubbled AND}$$

**(B-484) Fig. 3.11.4 : Illustration of De-Morgan's second theorem**

- This theorem can be verified by writing a truth table for both the sides of the theorem statement. This truth table is shown in Fig. 3.11.5, which shows that LHS = RHS.

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

LHS $A + B = \bar{A} \cdot \bar{B}$ RHS(C-6139) **Fig. 3.11.5 : Truth table to verify De-Morgan's theorem****3.11.4 Operator Precedence :**

- The operator precedence i.e. which logical operator should be considered first for solving Boolean expressions is as follows :
 1. Parenthesis
 2. NOT
 3. AND
 4. OR
- That means the expression inside the parenthesis should be evaluated before all the operations.



- The next priority should be given to the inversion or complement i.e. NOT operation, followed by the AND and OR operations.
- For example if $y = (\bar{A} + B)$ is to be evaluated, then we have to evaluate the expression inside the parenthesis first and the result should then be complemented.

3.12 Boolean Expression and Boolean Function :

- Boolean algebra deals with binary variables and logic operations.
- A Boolean function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1 and the logic operation symbols.
- Consider the following example :

$$(C-6140) \quad F(A, B, C, D) = \underbrace{A + BC}_{\text{Boolean function}} + \underbrace{\bar{B}C + \bar{A}D}_{\text{Boolean expression}} \quad \dots(3.12.1)$$

- Here the left side of the equation represents the output Y of a logic circuit. So we can state Equation (3.12.1) as,

$$Y = A + \bar{B}C + \bar{A}D \quad \dots(3.12.2)$$

- Another example of Boolean function is as follows :

$$F(A, B, C) = A + BC \quad \dots(3.12.3)$$

$$\text{or simply } Y = A + BC \quad \dots(3.12.4)$$

3.12.1 Truth Table Formation :

- It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

$$f(A, B, C) = A + BC$$

- It shows that there are three inputs A, B, and C and one output f(A, B, C) or simply F.
- The output will be high (1) if A = 1 or BC = 1 or both are 1 due to the (+) i.e. OR function present between A and BC.
- The truth table for this equation is shown by Table 3.12.1. The number of rows in the truth table is 2^n where n is the number of input variables (n = 3 for the given equation).

- Hence there are $2^3 = 8$ possible input combinations of inputs from ABC = 000 to ABC = 111.

(C-6141) Table 3.12.1 : Truth table for $f(A, B, C) = A + BC$

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

- If there are two inputs (n = 2), then there are $2^2 = 4$ combinations of inputs. For four inputs (n = 4), there are $2^4 = 16$ combinations.

3.12.2 Examples on Reducing the Boolean Expression :

Ex. 3.12.1 : Prove the following Boolean expressions :

1. $A + AB = A$
2. $A + \bar{A}B = A + B$
3. $(A + B)(A + C) = A + BC$

Soln. :

1. To prove that $A + AB = A$:

$$\begin{aligned} \text{LHS} &= A + AB = A(1 + B) \\ \text{But } (1 + B) &= 1 \quad \therefore \text{LHS} = A \cdot 1 \text{ But } A \cdot 1 = A \\ \therefore \text{LHS} &= A = \text{RHS} \\ \therefore A + AB &= A \end{aligned} \quad \text{...Proved.}$$

2. To prove that $A + \bar{A}B = A + B$:

$$\begin{aligned} \text{LHS} &= A + \bar{A}B = A + AB + \bar{A}B \\ &\quad \dots \text{since } A = A + AB \\ \therefore \text{LHS} &= A + B(A + \bar{A}) \\ \text{But } (A + \bar{A}) &= 1 \quad \therefore \text{LHS} = A + (B \cdot 1) = A + B \\ \therefore \text{LHS} &= \text{RHS} \\ \therefore A + \bar{A}B &= A + B \end{aligned} \quad \text{... Proved.}$$

3. To prove that $(A + B)(A + C) = A + BC$:

$$\begin{aligned} \text{LHS} &= (A + B)(A + C) \\ \therefore \text{LHS} &= AA + AC + BA + BC \\ &\quad \dots \text{According to the distributive law.} \\ \text{But } AA &= A, \\ \therefore \text{LHS} &= A + AC + BA + BC \\ \therefore \text{LHS} &= A(1 + C) + AB + BC \end{aligned}$$



$$\text{But } (1 + C) = 1$$

$$\therefore \text{LHS} = A + AB + BC = A(1 + B) + BC$$

$$\text{But } (1 + B) = 1$$

$$\therefore \text{LHS} = A + BC$$

$$\text{Thus } (A + B)(A + C) = A + BC \quad \dots \text{Proved.}$$

Ex. 3.12.2 : Prove that $(A + \bar{B} + AB)(A + \bar{B})(\bar{A}B) = 0$

Soln. :

$$\text{LHS} = (A + \bar{B} + AB)(A + \bar{B})(\bar{A}B)$$

$$\text{But } A + AB = A \quad \dots \text{Refer to Ex. 3.12.1}$$

$$\text{LHS} = (A + \bar{B})(A + \bar{B})(\bar{A}B)$$

$$= (AA + \bar{A}\bar{B} + \bar{A}B + \bar{B}B)(\bar{A}B)$$

$$\text{But } A \cdot A = A \text{ and } \bar{B} \cdot \bar{B} = \bar{B} \text{ and}$$

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

$$\therefore \text{LHS} = (A + \bar{A}B + \bar{B})(\bar{A}B)$$

$$= [A(1 + \bar{B}) + \bar{B}](\bar{A}B)$$

$$\text{But } (1 + \bar{B}) = 1 \quad \therefore \text{LHS} = [(A \cdot 1) + \bar{B}](\bar{A}B)$$

$$\therefore \text{LHS} = (A + \bar{B})(\bar{A}B) \dots \text{since } A \cdot 1 = A$$

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

$$\text{But } A\bar{A} = 0 \text{ and } B\bar{B} = 0$$

$$\therefore \text{LHS} = 0 + 0 = 0 \quad \dots \text{Proved.}$$

Ex. 3.12.3 : Prove that $A + \bar{A}B + AB = A + B$.

Soln. :

$$\text{LHS} = A + \bar{A}B + AB = A + B(\bar{A} + A)$$

$$\text{But } \bar{A} + A = 1$$

$$\therefore \text{LHS} = A + B = \text{RHS} \quad \dots \text{Proved.}$$

Ex. 3.12.4 : Simplify : $ABCD + A\bar{B}CD$.

$$\text{Soln. : } Y = ABCD + A\bar{B}CD = ACD(B + \bar{B})$$

$$\text{But } (B + \bar{B}) = 1$$

$$\therefore Y = ACD \quad \dots \text{Ans.}$$

Ex. 3.12.5 : Simplify the following expression :

$$Y = (\overline{AB} + \bar{A} + AB)$$

$$\text{Soln. : } Y = (\overline{AB} + \bar{A} + AB)$$

$$\text{But } \overline{AB} = \bar{A} + \bar{B} \dots \text{De-Morgan's first theorem}$$

$$\therefore Y = (\overline{A} + \bar{B} + \bar{A} + AB)$$

$$\text{But } \bar{A} + \bar{A} = \bar{A}$$

$$\therefore Y = (\bar{A} + \bar{B} + AB)$$

Now use De-Morgan's second theorem which states that,

$$\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

$$\therefore Y = \bar{A} \cdot \bar{B} \cdot \bar{AB}$$

$$\text{But } \bar{A} = A \text{ and } \bar{B} = B$$

$$\therefore Y = A \cdot B \cdot AB$$

$$\text{But } \overline{AB} = (\bar{A} + \bar{B}) \dots \text{De-Morgan's second theorem.}$$

$$\therefore Y = A \cdot B (\bar{A} + \bar{B}) = A\bar{A}B + A\bar{B}$$

$$\text{But } A\bar{A} = 0 \text{ and } B\bar{B} = 0$$

$$\therefore Y = 0 \cdot B + A \cdot 0 = 0 + 0$$

...since $0 \cdot B = 0$ and $A \cdot 0 = 0$

$$\therefore Y = 0$$

3.12.3 Complement of a Function :

- The complement of a function F is denoted by \bar{F} . We can obtain \bar{F} by replacing 0's by 1's and 1's by 0's while calculating the value of F.

- It is possible to derive the complement of a function algebraically using De Morgan's theorems. We can extend De Morgan's theorems to three or more variables as well.

- The three variable form of De Morgan's first theorem is derived as follows :

$$\text{LHS} = (\overline{A + B + C})$$

$$\text{Let } B + C = D$$

$$\therefore \text{LHS} = (\overline{A + D}) = \bar{A} \cdot \bar{D}$$

....As per De Morgan's theorem

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

$$\therefore (\overline{A + B + C}) = \bar{A} \cdot \bar{B} \cdot \bar{C} \quad \dots \text{Proved.}$$

- On the same lines we can derive the De Morgan's theorems for any number of variables. These theorems in the generalized form can be stated as follows :

$$1. (\overline{A + B + C + D + \dots + G}) = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} \cdot \dots \cdot \bar{G}$$



$$2. \overline{(A \cdot B \cdot C \cdot D \cdot \dots \cdot G)} = \overline{A} + \overline{B} + \overline{C} + \overline{D} + \dots + \overline{G}$$

Ex. 3.12.6 : Find the complement of the following functions.

$$F_1 = \overline{A} B \overline{C} + \overline{A} \overline{B} C \quad \text{and} \quad F_2 = A (\overline{B} \overline{C} + BC)$$

Soln. :

$$1. \quad F_1 = \overline{A} B \overline{C} + \overline{A} \overline{B} C$$

$$\therefore \overline{F_1} = (\overline{A} B C + \overline{A} \overline{B} C) = (\overline{A} B C) \cdot (\overline{A} \overline{B} C)$$

$$\therefore \overline{F_1} = (A + \overline{B} + C) \cdot (A + B + \overline{C}) \quad \dots \text{Ans.}$$

$$2. \quad F_2 = A (\overline{B} \overline{C} + BC)$$

$$\therefore \overline{F_2} = [A (\overline{B} \overline{C} + BC)]$$

$$= [\overline{A} + (\overline{B} \overline{C} + BC)]$$

$$= \overline{A} + (\overline{B} \overline{C}) \cdot (BC)$$

$$\therefore \overline{F_2} = \overline{A} + (B + C) (\overline{B} + \overline{C}) \quad \dots \text{Ans.}$$

Review Questions

- Q. 1 Explain with example the binary addition and subtraction.
- Q. 2 Give and explain the advantages of One's complement representation.
- Q. 3 Explain with example 2's complement method used for subtraction.

- Q. 4 What is two's complement number ?
- Q. 5 Explain the binary multiplication and division with example.
- Q. 6 What is two's complement number ?
- Q. 7 Find out 2's complement number of following decimal number :
 - 1. 17
 - 2. 20
 - 3. 107
- Q. 8 Subtract using 2's complement $(11011011)_2$ from $(0101010)_2$.
- Q. 9 Subtract using 2's complement 11001 from 01101.
- Q. 10 Subtract $1101101 - 11010$ using 2's complement.
- Q. 11 State AND laws.
- Q. 12 State OR laws.
- Q. 13 State De Morgan's theorems.
- Q. 14 What is AND-OR-NOT logic ?
- Q. 15 Define truth table.
- Q. 16 Write the Boolean expressions for the following :
 - 1. OR gate
 - 2. EX-NOR gate
- Q. 17 State the rules of Boolean algebra.
- Q. 18 State and explain the commutative law.
- Q. 19 State and explain the associative law.
- Q. 20 State and explain the distributive law.
- Q. 21 Write a note on : Duality theorem.
- Q. 22 State and prove De-Morgan's first and second theorem.



Unit 1

Chapter

4

Logic Minimization

Syllabus

Representation of logic function : Logic statement, Truth-table, SOP form, POS form; Simplification of logical functions using K-Maps upto 4 variables.

Chapter Contents

4.1 System or Circuit	4.5 Karnaugh-Map Simplification (The Map Method)
4.2 Standard Representations for Logical Functions	4.6 Simplification of Boolean Expressions using K-map
4.3 Concepts of Minterm and Maxterm	4.7 Minimization of SOP Expressions (K Map Simplification)
4.4 Methods to Simplify the Boolean Functions	4.8 Product of Sum (POS) Simplification



4.1 System or Circuit :

- A system or circuit is defined as the physical device or group of devices or algorithm which performs the required operations on the signal applied at its input which can be either analog or digital.
- Depending on the type of input signal (analog or digital) the systems or circuits can be classified into two types :
 1. Analog systems
 2. Digital systems.

4.1.1 Digital Systems :

- We may define the digital system as the system which processes or works upon the digital signals to produce another digital signal.



(C-215) Fig. 4.1.1 : A digital system

- Thus the input signal to a digital system is digital and its output signal is also digital.

Input Output Relation :

- In Fig. 4.1.1, A, B, C ... are the inputs whereas Y or $f(A, B, \dots)$ represents the output of the system. The system may have a single output or it can have multiple outputs.
- The system output is a function of inputs. Hence it is denoted by $f(A, B, C, \dots)$.
- The relation between input and output can be represented in various different ways as given below :
 1. Truth table.
 2. Switching equations.
 3. Logic diagram.

1. Truth table :

- Here the relation between the input variables (A, B, C ...) with the output Y is presented in a tabular form as shown in Table 4.1.1.
- The state of output (0 or 1) is written for all the possible combinations of inputs.

(C-8064) Table 4.1.1

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
⋮	⋮	⋮	⋮
1	1	1	1

2. Switching equations :

- The relation between inputs and output(s) can be presented in the form of equation(s) called as switching equations.
- The input variables are called as switching variables. The output (Y) is written on the left hand side of the equation whereas the terms containing the input variables are written on the right hand side of the switching equation as shown below :

$$Y = \overline{ABC} + \overline{ABC} + \overline{ABC} \quad \dots(4.1.1)$$

$$\text{Or } f(A, B, C) = \overline{ABC} + \overline{ABC} + \overline{ABC} \quad \dots(4.1.2)$$

$$\text{Or } f(A, B, C) = (\overline{A} + \overline{B} + C) \cdot (\overline{A} + B + \overline{C}) \quad \dots(4.1.3)$$

- The switching equations are also called as the Boolean equations or the **system equations**.
- The system equations or switching equations can be of two different types :
 1. Sum of Products (SOP) format.
 2. Product of Sums (POS) format.

3. Logic diagram :

This is a diagrammatic way of expressing the input-output relationship of a digital circuit.

4.1.2 Types of Digital Systems :

- The digital systems in general are classified into two categories namely :
 1. Combinational logic circuits.
 2. Sequential logic circuits.

Combinational circuits :

- It is a logic circuit, whose output at any instant of time, depends only on the levels present at input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied also has no effect on the output of a combinational circuit.



- The block diagram of a combinational circuit is shown in Fig. 4.1.2.



(C-332) Fig. 4.1.2 : Block diagram of a combinational circuit

- A combinational circuit can have a number of inputs and a number of outputs.
- The circuit of Fig. 4.1.2 has "n" inputs and "m" outputs.
- Between the inputs and outputs, logic gates are connected so combinational circuit basically consists of logic gates.
- A combinational circuit operates in three steps :
 - It accepts n-different inputs.
 - The combination of gates operates on the inputs.
 - "m" different outputs are produced as per requirement.

Examples of combinational circuits :

- Following are the examples of some combinational circuits :
 - Adders, subtractors
 - Comparator
 - Code converters
 - Encoders, decoders
 - Multiplexers, demultiplexers

4.1.3 Combinational Circuit Design :

- In this chapter we are going to discuss the design of combinational circuits.
- To design a combinational circuit we have been given the specifications or the requirements of combinational circuits. Such specifications or requirements can be specified in the following ways :
 - A list of statements
 - Boolean expressions
 - Truth table.
- From the specified requirements we have to design a combinational circuit using a combination of gates which will fulfill all the requirements.
- We can adopt one of the following two approaches to the combinational circuit design :
 - Traditional methods.

- Use of Medium Scale Integration (MSI).
- In this chapter we will discuss the traditional method.

Steps followed in traditional circuit design method :

- A truth table is given.
- Obtain the Boolean expression from the truth table.
- Simplify the Boolean equation using standard methods.
- Realize the simplified equation using gates i.e. build the logic circuit.

Methods to simplify the boolean equations :

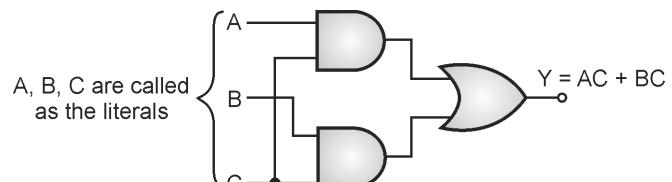
- The methods used for simplifying the Boolean functions are as follows :
 - Algebraic method.
 - Karnaugh-map (K-map) simplification.
 - Quine-Mc Cluskey method and
 - Variable Entered Mapping (VEM) technique.
- The Boolean theorems and De-Morgan's theorems are useful in simplifying the given Boolean expressions. We can then realize the logical expressions using either the conventional gates or universal gates.
- We should use the minimum number of logic gates for the realization of a logical expression.

4.2 Standard Representations for Logical Functions :

- Consider that the logic expression given to us is as follows :

$$Y = AC + BC$$

- Then it can be realized using basic gates as shown in Fig. 4.2.1.
- In this Boolean expression, Y is the result or output and A, B, C are called as **literals**.



(C-1217) Fig. 4.2.1 : Realization of given logic expression

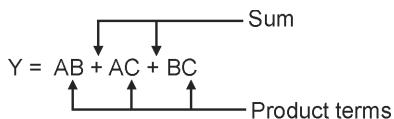
- When we realize the Boolean equation by using gates, each literal acts as an input as shown in Fig. 4.2.1.
- Any logic expression can be expressed in the following two standard forms :
 - Sum-of-Products form (SOP) and
 - Product-of-Sums form (POS)



- These two forms are suitable for reducing the given logic expression to its simplest form.

4.2.1 Sum-of-Products (SOP) Form :

- Refer to logic expression shown in Fig. 4.2.2. It is in the form of sum of three terms AB, AC and BC with each individual term is product of two variables. Say A-B or A-C etc.



(C-1217(a)) Fig. 4.2.2 : Sum-of-Products (SOP) form

- Therefore such expressions are known as expression in **SOP** form.
- The sums and products in the SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions.
- A, B and C are the **literals** or the inputs of the combinational circuit.
- Some more examples of the SOP expressions are as follows :

$$Y = ABC + BCD + ABD,$$

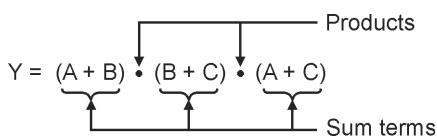
$$A = XY + \bar{X}Y + X\bar{Y},$$

$$Y = \bar{P}\bar{Q} + PQR + P\bar{Q}\bar{R}$$

- Thus in each product term there can be one or more than one literals ANDed together. The literals can be in their complemented or uncomplemented form.

4.2.2 Product of the Sums Form (POS) :

- Refer to the logic expression shown in Fig. 4.2.3. It is in the form of product of three terms $(A + B)$, $(B + C)$ and $(A + C)$, with each term is in the form of sum of two variables.
- Such expressions are said to be in the Product of Sums (**POS**) form.
- Some other examples of POS expressions form are as follows :



(C-1218) Fig. 4.2.3 : Product of Sums (POS) form

$$Y = (A + \bar{B} + \bar{C}) \cdot (A + B) \cdot (\bar{A} + C)$$

$$A = (\bar{X} + \bar{Y}) \cdot (X + Y + Z)$$

$$Y = (P + R) \cdot (P + \bar{Q}) \cdot (\bar{P} + R)$$

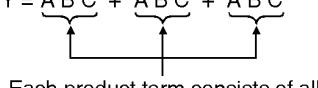
- The literals are ORed together to form the sum terms and the sum terms are ANDed to get the expression in the POS form.

4.2.3 Standard or Canonical SOP and POS Forms :

Canonical or standard forms :

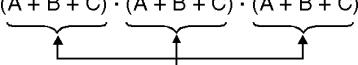
- The word standard is used in order to describe a condition of switching equation. The meaning of the word standard is conforming to a general rule.
- This rule states that each term used in a switching equation must contain all the available input variables.
- The two formats of a switching equation in the standard form are :
 1. Sum of Product (SOP) format.
 2. Product of Sum (POS) format.
- When we simplify a Boolean equation, sometimes an input variable is eliminated to simplify the equation.
- But standard expressions are not simplified. It is said that the standard expression is the opposite of simplification. So it contains redundancies.
- Many times, switching equations written in the SOP or POS form are not standard. That means each term may not contain all the input variables.
- Consider the SOP and POS expressions shown in Fig. 4.2.4.

Standard SOP form : $Y = A B C + A \bar{B} \bar{C} + \bar{A} B C$



Each product term consists of all the literals in the complemented or uncomplemented form

Standard POS form : $Y = (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + \bar{B} + \bar{C})$



Each sum term consists of all the literals in the complemented or uncomplemented form

(C-1219(a)) Fig. 4.2.4 : Standard SOP and POS forms

- Referring to Fig. 4.2.4 we can say that a logic expression is said to be in the **standard** SOP or POS form if each product term (for SOP); and sum term (for POS) consists of all the literals in their complemented or uncomplemented form.

**Soln. :****Step 1 : Find the missing literal for each term :**

$$Y = \underbrace{(A + B)}_{\text{Missing literal}} \underbrace{(A + C)}_{B} \underbrace{(B + \bar{C})}_{A}$$

(C-2384)

Step 2 : OR each term with (Missing literal. Its complement) :

$$Y = (A + B + C\bar{C}) \cdot (A + C + B\bar{B}) \cdot (B + \bar{C} + A\bar{A})$$

Missing literal ANDed with its complement

This term is ORed with the term formed by ANDing the missing literal with its complement

(C-6153)

Step 3 : Simplify the expression to get standard POS :

$$Y = (A + B + C\bar{C})(A + C + B\bar{B})(B + \bar{C} + A\bar{A})$$

But $A + BC = (A + B)(A + C)$

$$\therefore Y = (A + B + C)(A + B + \bar{C})(A + C + B)$$

$$(A + C + \bar{B})(B + \bar{C} + A)(B + \bar{C} + \bar{A})$$

But $A \cdot A = A$

$$\therefore (A + B + C)(A + C + B) = (A + B + C)$$

and $(A + B + \bar{C})(B + \bar{C} + A) = (A + B + \bar{C})$

(C-6154)

$$\therefore Y = \underbrace{(A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})}_{\text{Each term contains all the literals}} \text{ Standard POS form}$$

Ex. 4.2.3 : Convert the following expressions into their standard SOP or POS forms :

1. $Y = AB + AC + BC$
2. $Y = (A + B)(\bar{B} + C)$
3. $Y = A + BC + ABC$

Soln. : Solve it yourself.**Ans. :**

1. $Y = ABC + AB\bar{C} + A\bar{B}C + \bar{A}BC$
2. $Y = (A + B + C)(A + B + \bar{C})(\bar{B} + C + A)(\bar{B} + C + \bar{A})$
3. $Y = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + BCA$

4.3 Concepts of Minterm and Maxterm :

Minterm :

- Each individual term in the standard SOP form is called as **minterm**. This is shown in Fig. 4.3.1.

Maxterm :

- Each individual term in the standard POS form is called as **maxterm**. This is shown in Fig. 4.3.1.

Standard SOP $Y = \underbrace{ABC}_{\text{Each individual term is called minterm}} + \underbrace{A\bar{B}\bar{C}}_{\text{Each individual term is called minterm}} + \underbrace{\bar{A}B\bar{C}}_{\text{Each individual term is called minterm}}$

Standard POS $Y = \underbrace{(A + B)}_{\text{Each individual term is called maxterm}} \cdot \underbrace{(A + \bar{B})}_{\text{Each individual term is called maxterm}}$

(C-1220(a)) Fig. 4.3.1 : Concept of maxterm and minterm

- Table 4.3.1 gives the minterms and maxterms for a three variable/literal logic function. Let Y be the output and A, B, C be the inputs.

(C-8067) Table 4.3.1 : Minterms and maxterms for three variables

Variables			Minterms	Maxterms
A	B	C	m_i	M_i
0	0	0	$\bar{A} \bar{B} \bar{C} = m_0$	$A + B + C = M_0$
0	0	1	$\bar{A} \bar{B} C = m_1$	$A + B + \bar{C} = M_1$
0	1	0	$\bar{A} B \bar{C} = m_2$	$A + \bar{B} + C = M_2$
0	1	1	$\bar{A} B C = m_3$	$A + \bar{B} + \bar{C} = M_3$
1	0	0	$A \bar{B} \bar{C} = m_4$	$\bar{A} + B + C = M_4$
1	0	1	$A \bar{B} C = m_5$	$\bar{A} + B + \bar{C} = M_5$
1	1	0	$A B \bar{C} = m_6$	$\bar{A} + \bar{B} + C = M_6$
1	1	1	$A B C = m_7$	$\bar{A} + \bar{B} + \bar{C} = M_7$

- The number of minterms and maxterms is $2^3 = 8$. In general for "n" number of variables the number of minterms or maxterms will be $2^n - 1$ and each maxterm is represented by M_i .

Writing minterm for a particular combination of ABC :

- Consider ABC = 011. Assume that A, B, C are input to an AND gate.
- We want the output of the AND gate to be 1. For that all its inputs should be 1.
- So take the complement of that input which is 0 i.e. A in this case and write the minterm.

\therefore Minterm = $\bar{A} B C$ corresponding to ABC = 011

Similarly other minterms in Table 4.3.1 can be obtained.

Writing maxterm for a particular combination of ABC :

- Let ABC = 011.
- Assume that ABC are inputs to an OR gate.



- We want the output of the OR gate to be 0. So all the inputs to the OR gate should be 0s.
 - So invert the inputs which are 1s. i.e. B and C in this case and write the maxterm.
- $\therefore \text{Maxterm} = (A + \bar{B} + \bar{C})$ corresponding to $ABC = 011$
Similarly other maxterms in Table 4.3.1 can be obtained.

Ex. 4.3.1 : For the truth table of two variables write the minterms and maxterms.

Soln. :

Refer Table P. 4.3.1 for solution.

(C-8068) **Table P. 4.3.1 : Solution to Ex. 4.3.1**

Variables/literals		Minterms	Maxterms
A	B	m_i	M_i
0	0	$m_0 = \bar{A} \bar{B}$	$M_0 = A + B$
0	1	$m_1 = \bar{A} B$	$M_1 = A + \bar{B}$
1	0	$m_2 = A \bar{B}$	$M_2 = \bar{A} + B$
1	1	$m_3 = AB$	$M_3 = \bar{A} + \bar{B}$

4.3.1 Representation of Logical Expressions using Minterms and Maxterms :

- We can represent the logical expression using the minterms and maxterms as follows :

$$1. Y = ABC + \bar{A}BC + \bar{A}\bar{B}C \leftarrow \text{Given logic expression}$$

$m_7 \quad m_3 \quad m_4 \leftarrow \text{Corresponding minterms}$

$$\therefore Y = m_7 + m_3 + m_4 = \Sigma m (3, 4, 7) \leftarrow \text{Other way of representation}$$

(C-6155)

where Σ denotes sum of products.

$$2. Y = (A+\bar{B}+C)(A+B+C)(\bar{A}+\bar{B}+C) \leftarrow \text{Given expression}$$

$M_2 \quad M_0 \quad M_6 \leftarrow \text{Corresponding maxterms}$

$$\therefore Y = M_2 M_0 M_6$$

$$\therefore Y = \Pi M (0, 2, 6) \leftarrow \text{Other way of representation}$$

where Π denotes product of sums.

(C-6156)

4.3.2 Writing SOP and POS Forms for a Given Truth Table :

- We know that a logic expression can be represented in the truth table form. For example, the expression $Y = AB + \bar{A}\bar{B}$ which is the Boolean expression for an EX-NOR gate can also be represented using a truth table.

- Hence it is possible to obtain the logic expression in the standard SOP or POS form if a truth table is given to us.

4.3.3 To Write Standard SOP Expression for a Given Truth Table :

- The procedure to be followed for writing the standard SOP expression from a given truth table is as follows :

Step 1 : From the given truth table, consider only those combinations of inputs which produce an output $Y = 1$.

Step 2 : Write down a product term interms of input variables for each such combination.

Step 3 : OR all these product terms produced in step 2 to get the standard SOP.

Ex. 4.3.2 : From the truth table P. 4.3.2, obtain the logical expression in the standard SOP form.

(C-8114) **Table P. 4.3.2 : Given truth table**

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Soln. :

Step 1 : Consider only those combinations of inputs which correspond to $Y = 1$.

Steps 2 and 3 :

- For the second and the third entries in Table P. 4.3.2(a) write the product terms.

(C-6157) **Table P. 4.3.2(a)**

A	B	Y
1	0	0
2	0	1
3	1	0
4	1	1

$Y_1 = \bar{A}B$ $Y_2 = A\bar{B}$ Boolean expressions in the product forms

- Now OR (Add) all the product terms to write the final expression in standard SOP form as follows :

$$\begin{aligned} \therefore Y &= Y_1 + Y_2 \\ &= \bar{A}B + A\bar{B} = m_1 + m_2 \\ &= \Sigma m (1, 2) \end{aligned}$$

Ex. 4.3.3 : For the truth table shown in Table P. 4.3.3 write the logic expression in the standard SOP form.



(C-6412) Table P. 4.3.3 : Given truth table

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

 $\bar{A}\bar{B}\bar{C}$ (m_1) $\bar{A}\bar{B}\bar{C}$ (m_4) $\bar{A}\bar{B}C$ (m_7)**Soln. :**

Step 1 : Product terms corresponding to combinations of inputs for which $Y = 1$.

Step 2 : OR (Add) all the product terms :

- ORing all the product terms we get,

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C \quad \dots \text{Ans.}$$

$$\therefore Y = m_1 + m_4 + m_7 = \Sigma m (1, 4, 7)$$

4.3.4 To Write a Standard POS Expression for a Given Truth Table :

- Follow the procedure given below to get the expression in standard POS expression from a given truth table.

Step 1 : From the given truth table, consider only those combinations of inputs which produce a low output ($Y = 0$).

Step 2 : Write the maxterms only for such combinations.

Step 3 : AND these maxterms to obtain the expression in standard POS form.

Ex. 4.3.4 : Write the logic expression in standard POS form for the truth table shown in Table P. 4.3.4.

(C-6413) Table P. 4.3.4 : Given truth table

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

 $A + B + C$ (M_0) $A + \bar{B} + \bar{C}$ (M_3) $\bar{A} + B + \bar{C}$ (M_5) $\bar{A} + \bar{B} + C$ (M_6)**Soln. :**

Step 1 : Write maxterms for the combinations of input which produce $Y = 0$.

Step 2 : AND (take product of) all the maxterms to get standard POS form :

- ANDing (taking product of) all the maxterms written in step 1 we get,

$$Y = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C)$$

$$\therefore Y = M_0 \cdot M_3 \cdot M_5 \cdot M_6 = \Pi M (0, 3, 5, 6)$$

4.3.5 Conversion from SOP to POS and Vice Versa :

- It is important to note that the SOP and POS forms written for the same truth table are always logically equivalent.
- This point can be proved by solving the following example.

Ex. 4.3.5 : For the given truth table write the logical expressions in the standard SOP and POS forms and prove their equivalence.

(C-6414) Table P. 4.3.5 : Given truth table

Minterms	A	B	C	Y	Maxterms
0	0	0	0	0	$A + B + C$
0	0	0	1	1	$A + \bar{B} + C$
0	1	0	0	0	$A + \bar{B} + \bar{C}$
0	1	1	0	0	$\bar{A} + B + \bar{C}$
1	0	0	0	1	$\bar{A} + B + C$
1	0	1	0	0	$\bar{A} + \bar{B} + C$
1	1	0	0	0	$\bar{A} + \bar{B} + \bar{C}$
1	1	1	1	1	$A + \bar{B} + \bar{C}$

Soln. :

Step 1 : Minterms and maxterms for the combinations of inputs producing $Y = 1$ and 0 respectively.

Step 2 : Write the standard SOP and POS expressions :

Standard SOP form :

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC$$

Standard POS form :

$$Y = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})$$

$$(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$$

Step 3 : To prove the equivalence between SOP and POS forms :

- Consider the standard POS form.

$$Y = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})$$

$$(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$$



$$\begin{aligned} \text{But } (A + B + C)(A + \bar{B} + C) &= (A + C + B \cdot \bar{B}) \\ &= (A + C) \\ \therefore Y = (A + C)(A + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C) & \\ (1) & (2) & (3) & (4) \quad (\text{C-6354}) \end{aligned}$$

- Multiply the terms (1×2) and (3×4) to get,

$$\begin{aligned} Y &= [AA + A\bar{B} + A\bar{C} + AC + \bar{B}C + C\bar{C}] \\ &= [\bar{A}\bar{A} + \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{A}B + B\bar{B} + BC + \bar{A}\bar{C}] \\ &\quad + \bar{B}\bar{C} + C\bar{C}] \end{aligned}$$

$$\text{But } AA = A, C\bar{C} = 0, \bar{A}\bar{A} = \bar{A}, B\bar{B} = 0, \bar{C}\bar{C} = 0$$

$$\begin{aligned} \therefore Y &= [A + A\bar{B} + A\bar{C} + AC + \bar{B}C] \\ &= [\bar{A} + \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{A}B + BC + \bar{A}\bar{C} + \bar{B}\bar{C}] \\ &= [A(1 + \bar{B}) + A(\bar{C} + C) + \bar{B}C] \\ &= [(1 + \bar{B})\bar{A} + \bar{A}(C + \bar{C}) + \bar{A}B + BC + \bar{B}\bar{C}] \end{aligned}$$

$$\text{But } (1 + \bar{B}) = 1, (\bar{C} + C) = 1, (1 + \bar{B}) = 1$$

$$\therefore Y = [A + A + \bar{B}C] [\bar{A} + \bar{A} + \bar{A}B + BC + \bar{B}\bar{C}]$$

$$\text{But } A + A = A$$

$$\text{And } \bar{A} + \bar{A} = \bar{A}$$

$$\begin{aligned} \therefore Y &= (A + \bar{B}C)[\bar{A} + \bar{A}B + BC + \bar{B}\bar{C}] \\ &= (A + \bar{B}C)[\bar{A}(1 + B) + BC + \bar{B}\bar{C}] \end{aligned}$$

$$\text{But } (1 + B) = 1$$

$$\begin{aligned} \therefore Y &= (A + \bar{B}C)(\bar{A} + BC + \bar{B}\bar{C}) \\ &= A\bar{A} + ABC + A\bar{B}\bar{C} + \bar{A}B\bar{C} + (\bar{B}C \cdot BC) \\ &\quad + (\bar{B}C \cdot \bar{B}\bar{C}) \end{aligned}$$

$$\text{But } A\bar{A} = 0, (\bar{B}C \cdot BC) = 0 \text{ and } (\bar{B}C \cdot \bar{B}\bar{C}) = 0$$

$\therefore Y = ABC + A\bar{B}\bar{C} + \bar{B}C$ ← This is same as standard SOP form.
Thus the equivalence SOP and POS forms is proved.

(C-6363)

Step 4 : Express output in terms of minterms and maxterms :

- The output can be expressed in terms of minterms and maxterms as follows :

$$\begin{aligned} Y &= m_1 + m_4 + m_7 \quad \dots \text{In terms of minterms.} \\ &= \sum m(1, 4, 7) \end{aligned}$$

$$\text{and } Y = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \dots \text{In terms of maxterms.}$$

$$= \prod M(0, 2, 3, 5, 6)$$

- Due to equivalence between SOP and POS forms we can write,

$$\sum m(1, 4, 7) = \prod M(0, 2, 3, 5, 6) \quad \dots(1)$$

Complementary relationship between minterms and maxterms :

- From Equation (1) we can conclude that the relationship between the expressions expressed using minterms and maxterms is complementary relationship.
- We can exploit this complementary relationship to write the expression in terms of maxterms if the expression in terms of minterms is known and vice versa.
- For example, if a SOP expression for 4-variables is given by,

$$Y = \sum m(0, 1, 3, 5, 6, 7, 11, 12, 15)$$

- Then we can get the equivalent POS expression using the complementary relationship as follows :

$$Y = \prod M(2, 4, 8, 9, 10, 13, 14)$$

4.4 Methods to Simplify the Boolean Functions :

- The methods used for simplifying the Boolean expressions are as follows :
 1. Algebraic method.
 2. Karnaugh-map simplification.
 3. Quine Mc-Cluskey method and
 4. Variable Entered Mapping (VEM) technique.
- The Boolean theorems and De-Morgan's theorems are useful in simplifying the given Boolean expressions.
- We can then realize the logical expressions using either the standard gates or universal gates.
- We should use the minimum possible number of logic gates for the realization of a logical expression.
- This is possible if we can simplify the logical expressions.
- In this chapter we will discuss one of the simplification techniques called **Karnaugh map or K-map** and the **Quine Mc-Cluskey Method** or the **Tabular Method**.

4.4.1 Algebraic Simplification :

- We have studied the Boolean laws and De-Morgan's theorems.
- We can use them to simplify the given Boolean expression in the following way.
- The most important thing is to convert the given expression into SOP form.

**Standard procedure for algebraic simplification :**

- Step 1 :** Bring the given expression into the SOP form by using the Boolean laws and De-Morgan's theorems.
- Step 2 :** Simplify this SOP expression by checking the product terms for common factors.

Ex. 4.4.1 : Simplify the expression given below :

$$Y = AB + (A + B)(\bar{A} + B).$$

Soln. :

Step 1 : Bring the given expression in SOP form :

Given expression :

$$\begin{aligned} Y &= AB + (A + B)(\bar{A} + B) \\ &= AB + (\bar{A}\bar{B} + AB + \bar{A}B + BB) \end{aligned}$$

Step 2 : Search for common factors and simplify :

$$\begin{aligned} Y &= AB + A\bar{A} + AB + \bar{A}B + BB \\ &= AB + AB + BB + A\bar{A} + \bar{A}B \end{aligned}$$

But $AB + AB = AB$, $BB = B$ and $A\bar{A} = 0$

$$\therefore Y = AB + B + \bar{A}B = B(A + 1) + \bar{A}B$$

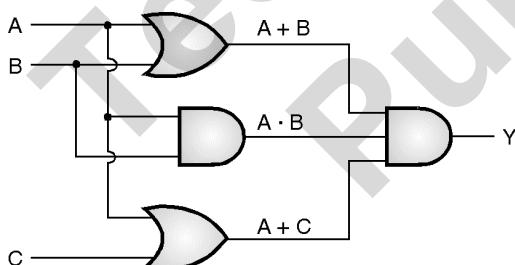
But $(A + 1) = 1$

$$\therefore Y = B + \bar{A}B = B(1 + \bar{A}) = B \dots \text{since } (1 + \bar{A}) = 1$$

$$\therefore Y = B \quad \dots \text{Ans.}$$

- This is simplified expression.

Ex. 4.4.2 : For the logic circuit shown in Fig. P. 4.4.2 write the Boolean expression and simplify it.



(C-1221) Fig. P. 4.4.2 : Given logic circuit

Soln. :

Step 1 : Write the Boolean expression :

- The expression for output of the given logic circuit is,
- $$Y = (A + B)(AB)(A + C)$$

Step 2 : Bring this expression in SOP form :

- Multiply the terms to get the expression into SOP form.

$$\therefore Y = (A + B)(AAB + ABC)$$

But $AA = A$

$$\therefore Y = (A + B)(AB + ABC)$$

$$= AAB + AABC + BAB + BABC$$

$$= AB + ABC + AB + ABC$$

$$= AB + AB + ABC + ABC$$

But $AB + AB = AB$ and $ABC + ABC = ABC$

$$\therefore Y = AB + ABC = AB(1 + C)$$

$$Y = AB$$

...since $1 + C = 1$.

- This is the simplified expression.

4.5 Karnaugh-Map Simplification (The Map Method) :

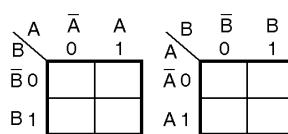
- This is another simplification technique to reduce the Boolean equation.
- It overcomes all the disadvantages of the algebraic simplification technique.
- K-map (short form of Karnaugh map) is a graphical method of simplifying a Boolean equation.
- K-map is a graphical chart made up of rectangular boxes.
- The information contained in a truth table or available in the SOP or POS form can be represented on a K-map.
- The K-map can be used for systematic simplification of Boolean expression.
- K-maps can be written for 2, 3, 4 ... upto 6 variables. Beyond that the K-map technique becomes very cumbersome.

K-map is ideally suitable for designing the combinational logic circuits using either a SOP method or a POS method.

- The K-map is drawn for output Y and the input variables (A, B, C... etc.) are used for making the entries in the boxes.

4.5.1 K-map Structure :

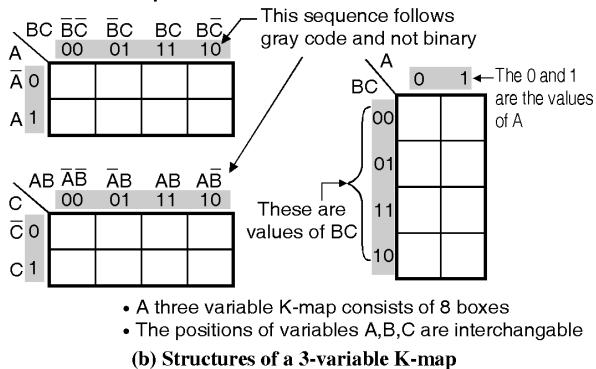
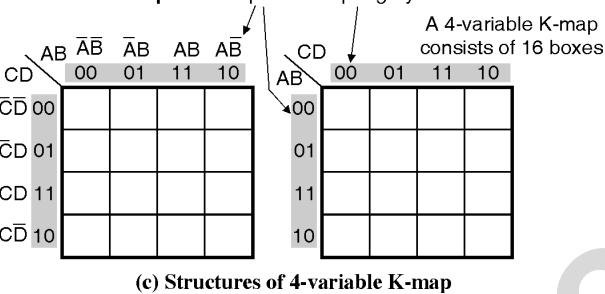
- The structure of a 2 input (variable) Karnaugh is shown in Fig. 4.5.1(a).
- This K-map is drawn for output Y of any two input combinational circuit such as a logic gate with inputs A and B. i.e. AB = 00, 01, 10 and 11.

2 Variable K-map

- A and B are the inputs or variables
- 0 and 1 are the value of A or B
- Inside 4 boxes we have to enter the values of Y i.e. output

(a) Structures of a 2-variable K-map

Fig. 4.5.1(Contd...)

**3 Variable K-map****4 Variable K-map**

(C-217) Fig. 4.5.1 : Structures of 2, 3, 4 variable K-map

- A 2 variable K-map consists of $2^2 = 4$ rectangular boxes. Inside these boxes we have to enter the values of output Y for different combinations of inputs A and B.
- The K-map comprises a box for every line in the truth table. For a 2-input combinational circuit there are 4-lines in the truth table, so there will be 4-boxes in the 2 variable K-map.
- For a 3-variable K-map there will be 8 boxes, for 4-variable map there will be 16 boxes and so on.
- The 0's and 1's written on top or sides of the boxes represent the values of the corresponding variables.

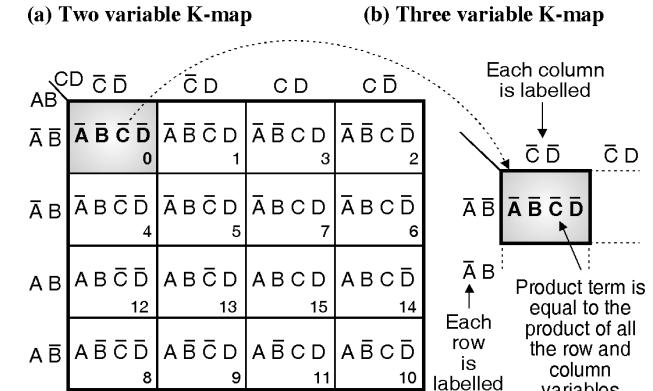
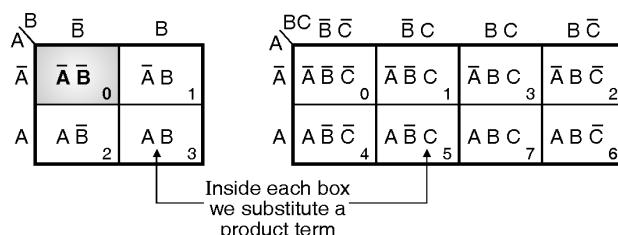
The sequence is funny (It is gray code) :

- In truth tables, the values of inputs follow a standard binary sequence (00, 01, 10, 11).
- But in K-maps the input values are ordered in a different sequence (00, 01, 11, 10).
- This is as per the **gray code** and not binary code. So that as we move along a row or column only one variable will change its value at a time.

- If the labeling is done as per the gray code, only then the elimination of variables and therefore the simplification will take place.
- When pairs, quads or octets are identified with the normal labeling, this elimination will not take place.
- Hence gray code is used in labeling the cells of a K-map.

4.5.2 K-map Boxes and Associated Product Terms :

- The rectangular boxes in a K-map are to be filled with the values of output Y corresponding to different combinations of inputs, as shown in Fig. 4.5.2.
- Each row and column of a K-map is labelled with a variable, or a group of variables or their complements. For example, see the shaded box in Fig. 4.5.2(a).
- This box corresponds to the first row which is labelled by \bar{A} and first column labelled by \bar{B} .
- Hence the product term written inside this box is $\bar{A} \bar{B}$ as shown in Fig. 4.5.2(a).
- Another example is the shaded box shown in Fig. 4.5.2(c). The first row is labelled with $\bar{A} \bar{B}$ and the first column with $\bar{C} \bar{D}$. Hence the product term written in this box is $\bar{A} \bar{B} \bar{C} \bar{D}$.
- Similarly the other product terms have been inserted, in the remaining boxes.

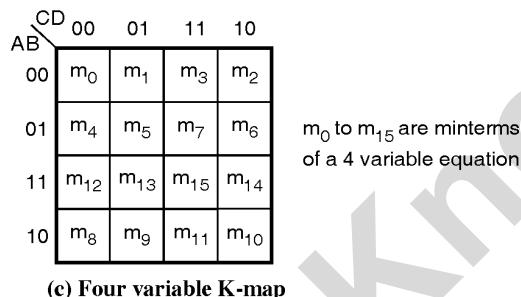
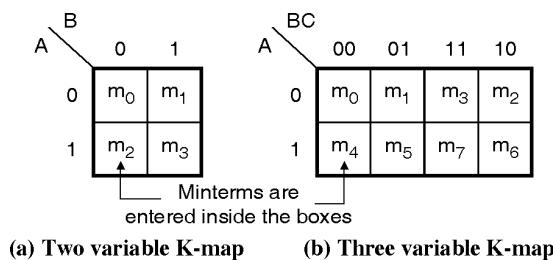


(C-218) Fig. 4.5.2 : K-map boxes and associated product terms



4.5.3 Alternative Way to Label the K-map :

- We can label the rows and columns of a K-map in a different way as shown in Fig. 4.5.3.
- Instead of labelling the rows and columns with the inputs and their complements (A , \bar{A} , $A\bar{B}$ etc.), their values in terms of 0s and 1s is used for labelling.
- And inside the boxes, instead of writing the actual product term, the corresponding shorthand minterm notations m_0 , m_1 are entered.



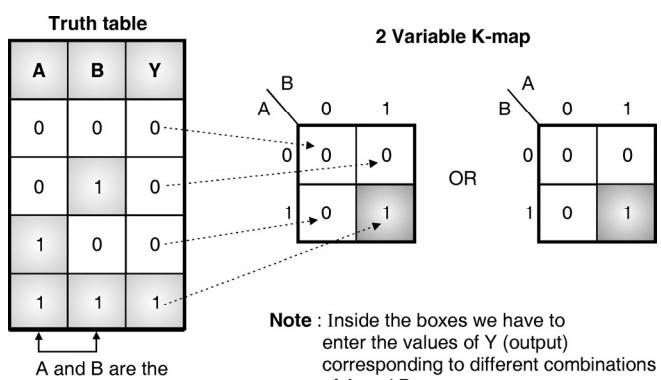
(C-219) Fig. 4.5.3 : Alternative way to label K-map

4.5.4 Truth Table to K-map :

- Whenever the K-map is to be practically used for simplification, the entries inside the boxes are to be written by referring to the given truth table.
- In this section we are going to learn how to represent the given truth table on a Karnaugh map.

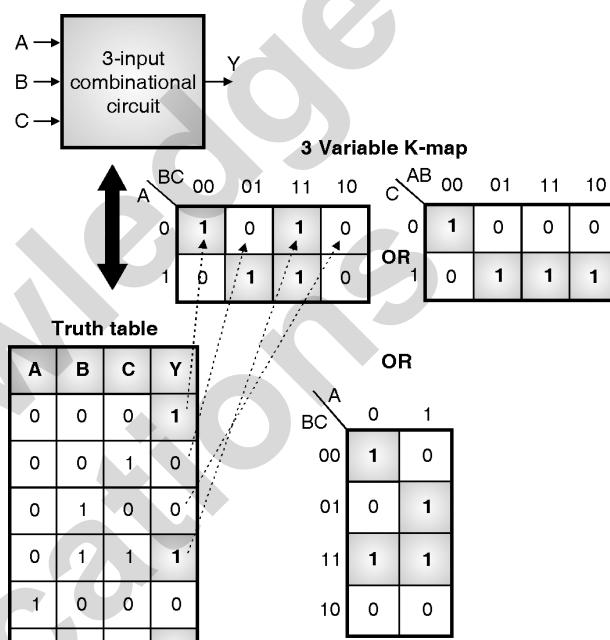
Relation between a truth table and K-map entries :

- Fig. 4.5.4(a) illustrates the mapping of a truth table on the K-map.



(C-220) Fig. 4.5.4(a) : Relation between a truth table and K-maps for 2 variables

- Referring to Fig. 4.5.4(a) we conclude that inside the boxes of the K-map we have to enter the values of output Y corresponding to various combinations of A and B .
- Fig. 4.5.4(b) shows the representation of a truth table using a 3-variable K-map and Fig. 4.5.4(c) shows the representation of a truth table using a 4-variable K-map.

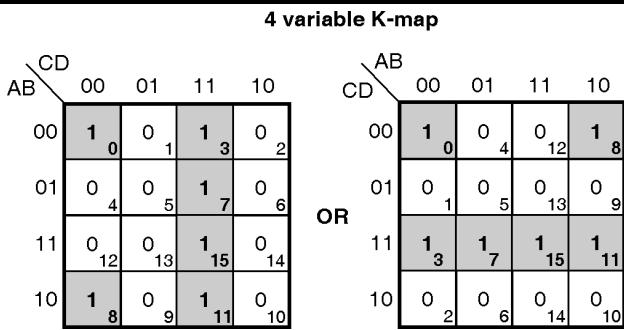


Note : Inside the boxes we enter the values of Y corresponding to different combinations of A and B

(C-221) Fig. 4.5.4(b) : Relation between a truth table and K-map for 3-variables

(C-6693) Truth table

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



(C-222) Fig. 4.5.4(c) : Relation between the truth table and 4-variable K-map

4.5.5 Representation of Standard SOP Form on K-map :

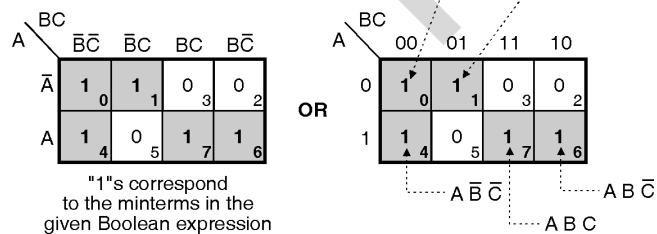
- The logical expression in standard SOP form can be represented with the help of a K-map by simply entering 1's in the cells (boxes) of the K-map corresponding to each minterm present in the equation.
- The remaining cells (boxes) are filled with zeros.
- Ex. 4.5.1 illustrates the concept of transferring a standard SOP expression on K-map.

Ex. 4.5.1 : Represent the equation given below on Karnaugh map.

$$Y = \bar{A} \bar{B} \bar{C} + \bar{A} \bar{B} C + A \bar{B} \bar{C} + A B \bar{C} + A B C.$$

Soln. :

- The given expression is in the standard SOP form. Each term represents a minterm.
- We have to enter 1's in the boxes corresponding to each minterm, as shown in Fig. P. 4.5.1.



(C-223) Fig. P. 4.5.1 : Representation of standard SOP on K-map

Ex. 4.5.2 : Plot the following Boolean expression on K-map.

$$Y = \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C D + \bar{A} B C \bar{D} + A B \bar{C} \bar{D}$$

Soln. : Refer Fig. P. 4.5.2.

CD	00	01	11	10	
AB	00	1 ₀	0 ₁	1 ₃	0 ₂
00	0 ₄	0 ₅	0 ₇	1 ₆	
01	0 ₁₂	0 ₁₃	0 ₁₅	0 ₁₄	
11	0 ₈	0 ₉	0 ₁₁	0 ₁₀	
10	0 ₂	0 ₆	0 ₁₄	0 ₁₀	

"1"s correspond to the minterms in the given Boolean expression

(C-224) Fig. P. 4.5.2 : Representation of canonical SOP on Karnaugh map

CD	00	01	11	10	
AB	00	1 ₀	0 ₁	1 ₃	0 ₂
00	0 ₄	0 ₅	0 ₇	1 ₆	
01	0 ₁₂	0 ₁₃	0 ₁₅	0 ₁₄	
11	0 ₈	0 ₉	0 ₁₁	0 ₁₀	
10	0 ₂	0 ₆	0 ₁₄	0 ₁₀	

(C-224) Fig. P. 4.5.2 : Representation of canonical SOP on Karnaugh map

4.6 Simplification of Boolean Expressions using K-map :

- Simplification of Boolean expressions using K-map is based on combining or grouping the terms in the adjacent cells (or boxes) of a K-map.
- Two cells of a K-map are said to be adjacent if they differ in only one variable as shown in Fig. 4.6.1.

BC	00	01	11	10	
A	0	1	2	3	4
0	5	6	7	8	
1	9	10	11	12	

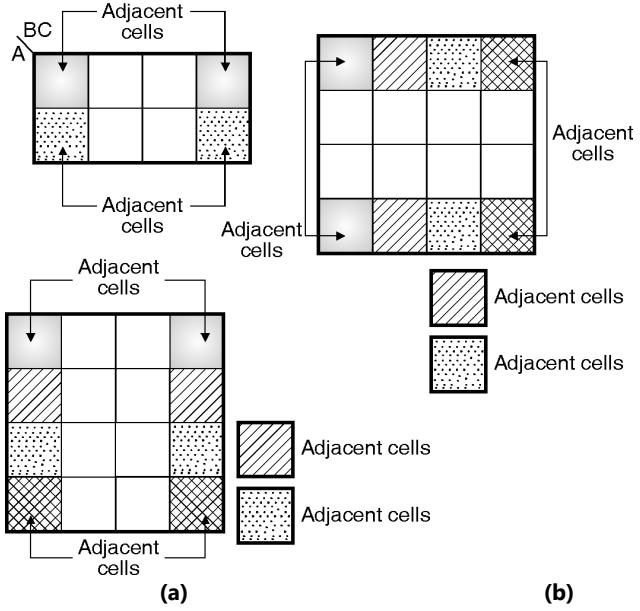
- 1 and 2 are adjacent
- 1 and 5 are adjacent
- But 1-6 or 2-5 are not adjacent

(C-225) Fig. 4.6.1 : Adjacent or non-adjacent cells

- Note the cells on left or right side or at the bottom and top of cells are adjacent cells. But the cells connected diagonally are not the adjacent ones.
- The left most cells are adjacent to their corresponding right most cells as shown in Fig. 4.6.2(a).

Left most and Right most cells are adjacent

Top and corresponding bottom cells are adjacent



(C-226) Fig. 4.6.2 : Illustration of adjacent cells

- And the top cells are adjacent to their corresponding bottom cells.

4.6.1 How does Simplification Takes Place ?

- Once we transfer the logic function or truth table on a Karnaugh map, we have to use the grouping technique for simplifying the logic function.
- Grouping means combining the terms in the adjacent cells.
- The grouping of either adjacent 1's or adjacent 0's results in the simplification of Boolean expression in the SOP or POS forms respectively.
- If we group the adjacent 1's then the result of simplification is in SOP (Sum Of Products) form.
- And if the adjacent 0's are grouped, then the result of simplification is in the POS (Product Of Sums) form.

4.6.2 Way of Grouping (Pairs, Quads and Octets) :

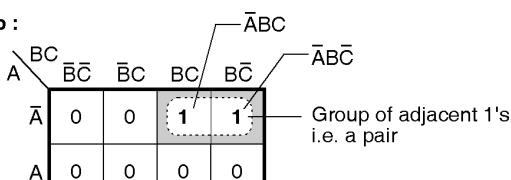
- While grouping, we should group most number of 1's (or 0's).
 - The grouping follows the binary rule i.e. we can group 1, 2, 4, 8, 16, 32 number of 1's or 0's. We cannot group 3, 5, 7, number of 1's or 0's.
- Pairs** : A group of two adjacent 1's or 0's is called as a pair.
 - Quad** : A group of four adjacent 1's or 0's is called as a quad.
 - Octet** : A group of eight adjacent 1's or 0's is called as octet.

4.6.3 Grouping Two Adjacent One's (Pairs) :

If we group two adjacent 1's on a K-map, to form a pair, then the resulting term will have one less literal (variable) than the original term. **That means by pairing two adjacent 1's we can eliminate one variable.**

- The grouping of two adjacent 1's and elimination of one variable due to pairing is illustrated in Figs. 4.6.3(a) to (d).

Given K-map :



(C-228) Fig. 4.6.3(a)

Simplification :

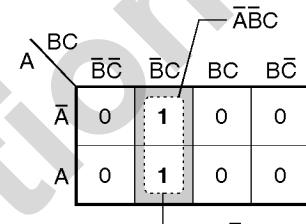
$$\begin{aligned} Y &= \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} \\ &= \bar{A}B(C + \bar{C}) \\ &= \bar{A}B \dots \text{since } C + \bar{C} = 1 \end{aligned}$$

- Thus C is eliminated.

Conclusion :

- Due to formation of pair the variable C is eliminated. So henceforth just by looking at the pair we should be able to identify the variable that will be eliminated.
- The other types of pairs and corresponding simplifications are as shown in Fig. 4.6.4(a) and Fig. 4.6.4(b).

Given K-map :



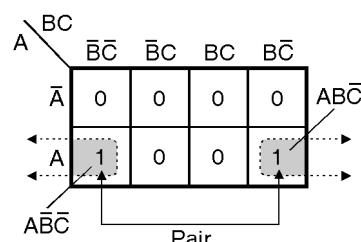
(C-228) Fig. 4.6.3(b)

Simplification :

$$\begin{aligned} Y &= \bar{A}\bar{B}C + A\bar{B}C \\ &= \bar{B}C(\bar{A} + A) \\ Y &= \bar{B}C \quad \dots \text{since } (\bar{A} + A) = 1 \end{aligned}$$

- Thus A is eliminated.

Given K-map :



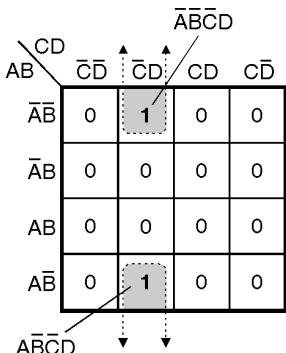
(C-228) Fig. 4.6.3(c)

Simplification :

$$\begin{aligned} Y &= A\bar{B}\bar{C} + A\bar{B}\bar{C} = A\bar{C} (B + \bar{B}) \\ Y &= A\bar{C} \quad \dots \text{since } (B + \bar{B}) = 1 \end{aligned}$$

- Thus B is eliminated.

Given K-map :



(C-228) Fig. 4.6.3(d)

Simplification :

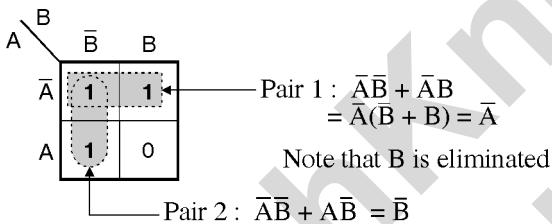
$$Y = \bar{A} \bar{B} C D + A \bar{B} C D = \bar{B} C D (\bar{A} + A)$$

$$\therefore Y = \bar{B} C D$$

- Thus A is eliminated.

Example of overlap :

Given K-map :



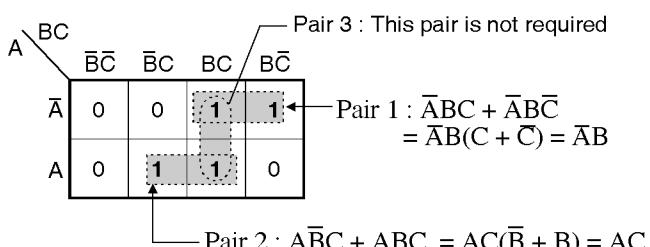
(C-229) Fig. 4.6.4(a)

Final expression

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

- Note that in order to cover all the 1's, we have to overlap two pairs as shown.

Given K-map :



(C-229) Fig. 4.6.4(b) : Different types of pairs and the corresponding simplifications

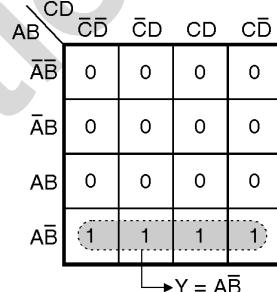
- Add the two product terms to get, $Y = \bar{A}B + AC$

Note : In this K-map three pairs were possible to be formed. However only two pairs are sufficient to include all the 1's present in the K-map. Then the third pair is not required.

4.6.4 Grouping Four Adjacent Ones (Quad) :

- If we group four 1's from the adjacent cells of a K-map, then the group is called as a **Quad**.
- In such a quad two variables associated with the minterms are same and the other two are not the same.
- After forming a Quad, the simplification takes place in such a way that the two variables which are not same will be eliminated.
- Thus a quad eliminates two variables.**
- Fig. 4.6.5(a) to (f) shows various types of Quads and the corresponding mathematical simplification.
- Note that overlapping is possible in Quads.

Given K-map :



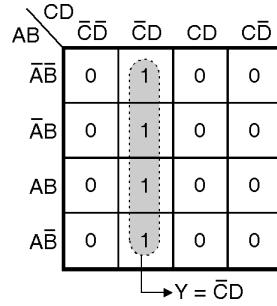
(C-230) Fig. 4.6.5(a)

Simplification :

$$\begin{aligned} Y &= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + A\bar{B}\bar{C}\bar{D} \\ &= \bar{A}\bar{B} [\bar{C}\bar{D} + \bar{C}D + CD + C\bar{D}] \\ &= \bar{A}\bar{B} [\bar{C}(\bar{D} + D) + C(D + \bar{D})] \\ \therefore Y &= \bar{A}\bar{B} [\bar{C} + C] = \bar{A}\bar{B} \end{aligned}$$

Thus C and D are eliminated.

Given K-map :



(C-231) Fig. 4.6.5(b)

**Simplification :**

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} D + \bar{A} \bar{B} C D + A \bar{B} \bar{C} D + A \bar{B} C \bar{D} \\
 &= \bar{C} D [\bar{A} \bar{B} + \bar{A} B + A B + A \bar{B}] \\
 &= \bar{C} D [\bar{A} (\bar{B} + B) + A (B + \bar{B})] \\
 &= \bar{C} D [\bar{A} + A] = \bar{C} D
 \end{aligned}$$

Thus A and B are eliminated.

Four adjacent ones forming a square :

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	0	0	0	0
		$\bar{A}B$	1	1	0	0
		AB	1	1	0	0
		$A\bar{B}$	0	0	0	0

(C-231(a)) Fig. 4.6.5(c)

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} \bar{D} + A \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D} + A \bar{B} C \bar{D} \\
 &= \bar{B} \bar{C} \bar{D} (\bar{A} + A) + B \bar{C} D (\bar{A} + A) \\
 &= \bar{B} \bar{C} \bar{D} + B \bar{C} D = \bar{B} \bar{C} (\bar{D} + D) \\
 \therefore Y &= \bar{B} \bar{C} \text{ Thus A and D are eliminated.}
 \end{aligned}$$

Top and bottom 1's forming a Quad :

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	0	1	1	0
		$\bar{A}B$	0	0	0	0
		AB	0	0	0	0
		$A\bar{B}$	0	1	1	0

(C-231(b)) Fig. 4.6.5(d)

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} D + \bar{A} \bar{B} C D + A \bar{B} \bar{C} D + A \bar{B} C \bar{D} \\
 &= \bar{A} \bar{B} D (\bar{C} + C) + A \bar{B} D (\bar{C} + C) \\
 &= \bar{A} \bar{B} D + A \bar{B} D \\
 \therefore Y &= \bar{B} D (\bar{A} + A) = \bar{B} D
 \end{aligned}$$

Thus A and C are eliminated.

Leftmost and rightmost 1's forming a Quad :

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	0	0	0	0
		$\bar{A}B$	1	0	0	1
		AB	1	0	0	1
		$A\bar{B}$	0	0	0	0

$$Y = B \bar{D} \text{ (A and C are eliminated)}$$

(C-231(c)) Fig. 4.6.5(e)

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} \bar{D} + A \bar{B} \bar{C} \bar{D} + \bar{A} B C \bar{D} + A B C \bar{D} \\
 &= \bar{B} \bar{C} \bar{D} (\bar{A} + A) + B C \bar{D} (\bar{A} + A) \\
 &= B \bar{C} \bar{D} + B C \bar{D}
 \end{aligned}$$

Thus A and C are eliminated.

1's corresponding to corners forming a Quad :

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	1	0	0	1
		$\bar{A}B$	0	0	0	0
		AB	0	0	0	0
		$A\bar{B}$	1	0	0	1

$$Y = \bar{B} \bar{D} \text{ (A and C are eliminated)}$$

(C-232) Fig. 4.6.5(f)

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} \bar{D} + A \bar{B} \bar{C} \bar{D} + \bar{A} B C \bar{D} + A B C \bar{D} \\
 &= \bar{B} \bar{C} \bar{D} (\bar{A} + A) + B C \bar{D} (\bar{A} + A) \\
 &= \bar{B} \bar{C} \bar{D} + B C \bar{D} = \bar{B} \bar{D} (\bar{C} + C)
 \end{aligned}$$

$$\therefore Y = \bar{B} \bar{D}$$

Thus A and C are eliminated.

Overlapping of Quads and pairs :

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	0	0	0	0
		$\bar{A}B$	1	1	1	1
		AB	0	1	1	1
		$A\bar{B}$	0	0	0	0

There are two quads (overlapping) and one pair
 $\therefore Y = BC + BD + \bar{ABC}$

(C-233) Fig. 4.6.5(g)



4.6.5 Grouping Eight Adjacent Ones (Octet) :

- It is possible to form a group of eight adjacent ones. Such a group is called as octet.
- When an octet is formed three variables will change and only one variable will remain same in all the minterms.
- The three variables that change will be eliminated and the variable which does not change will appear as output.
- Thus octet eliminates three variables.
- Fig. 4.6.6(a) to (d) shows various types of octets and the corresponding output.

Given K-map :

		CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	1	1	1	1
		$\bar{A}B$	1	1	1	1
		AB	0	0	0	0
		$A\bar{B}$	0	0	0	0

B,C and D are eliminated
(C-234) Fig. 4.6.6(a)

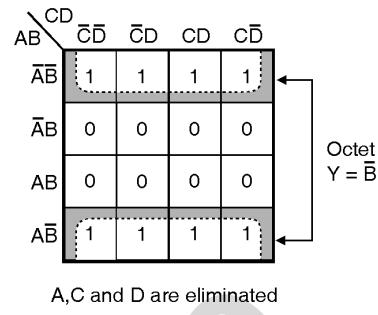
Simplification :

$$\begin{aligned}
 Y &= \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} \bar{C} D + \bar{A} \bar{B} C \bar{D} + \bar{A} \bar{B} C \bar{D} \\
 &\quad + \bar{A} B \bar{C} \bar{D} + \bar{A} B \bar{C} D + \bar{A} B C \bar{D} + \bar{A} B C \bar{D} \\
 \therefore Y &= \bar{A} \bar{B} \bar{C} (\bar{D} + D) + \bar{A} \bar{B} C (D + \bar{D}) \\
 &\quad + \bar{A} B \bar{C} (\bar{D} + D) + \bar{A} B C (D + \bar{D}) \\
 \therefore Y &= \bar{A} \bar{B} (\bar{C} + C) + \bar{A} B (\bar{C} + C) \\
 \therefore Y &= \bar{A} (\bar{B} + B) = \bar{A}
 \end{aligned}$$

- The only variable that remains same is \bar{A} . So it appears as output.

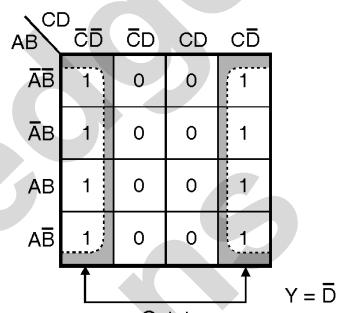
		CD	$\bar{C}D$	$\bar{C}D$	CD	$C\bar{D}$
		$\bar{A}B$	0	0	1	1
		$\bar{A}B$	0	0	1	1
		AB	0	0	1	1
		$A\bar{B}$	0	0	1	1

A,B and D are eliminated
(C-234(a)) (b)



A,C and D are eliminated

(C-234(b)) (c)



Octet
A,B and C are eliminated

(C-235) (d)

Fig. 4.6.6

4.6.6 Summary of Rules Followed for K-Map Simplification :

Summary :

- No zeros allowed.
- No diagonals.
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping allowed.
- Wrap around allowed.
- Fewest number of groups possible.

4.7 Minimization of SOP Expressions (K Map Simplification) :

- The K-map can be used to simplify the logical expression to a level beyond which it cannot be further simplified.
- After such a simplification, it will require minimum number of gates with minimum number of inputs to the gates.
- Such an expression is called as a **minimized** expression.
- For minimizing the logical expression, follow the procedure given below.

Minimization procedure :

- Step 1 :** Prepare the K-map and place 1's according to the given truth table or logical expression. Fill the remaining cells by 0's.
- Step 2 :** Locate the isolated 1's i.e. the 1's which cannot be combined with any other 1. Encircle such 1's.
- Step 3 :** Identify the 1's which can be combined to form a pair in only one way and encircle them.
- Step 4 :** Identify the 1's which can form a quad in only one way and encircle them.
- Step 5 :** Identify the 1's which can form an octet in only one way and encircle them.
- Step 6 :** After identifying the pairs, quads and octets, check if any 1 is yet to be encircled. If yes then encircle them with each other or with the already encircled 1's (by means of overlapping).

- Note that the number of groups should be minimum.
- Also note that any 1 can be included any number of times without affecting the expression.
- Let us solve some examples on this to make the concept clear.

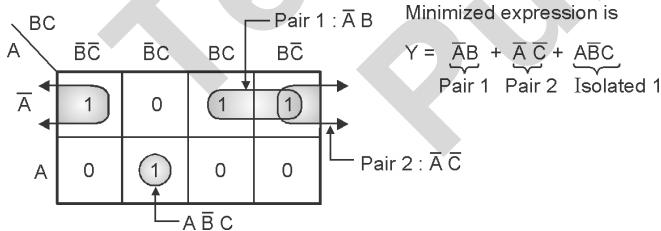
Ex. 4.7.1 : A logical expression in the standard SOP form is as follows :

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

Minimize it using the K-map technique.

Soln. : $Y = \sum m(0, 2, 3, 5)$

- The required K-map is as shown in Fig. P. 4.7.1.



(C-250) Fig. P. 4.7.1

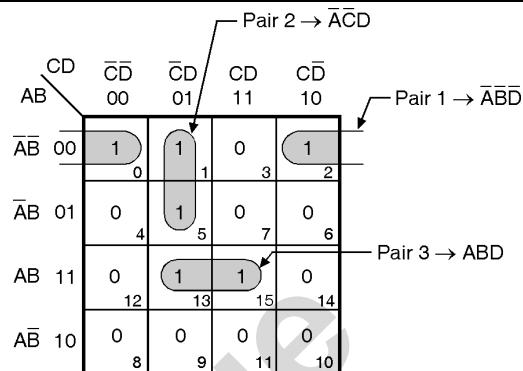
Ex. 4.7.2 : The logical expression representing a logic circuit is $Y = \sum m(0, 1, 2, 5, 13, 15)$. Draw the K-map and find the minimized logical expression.

Soln. :

- From the given expression, it is clear that the number of variables is 4.

$$Y = m_0 + m_1 + m_2 + m_5 + m_{13} + m_{15}$$

- The required K-map is as shown in Fig. P. 4.7.2.



(C-251) Fig. P. 4.7.2

Minimized expression is,

$$(C-6158) \quad Y = \underbrace{\bar{A} \bar{B} \bar{D}}_{\text{Pair 1}} + \underbrace{\bar{A} \bar{C} D}_{\text{Pair 2}} + \underbrace{A B D}_{\text{Pair 3}}$$

Ex. 4.7.3 : For the logical expression given below draw the K-map and obtain the simplified logical expression. $Y = \sum m(1, 5, 7, 9, 11, 13, 15)$.

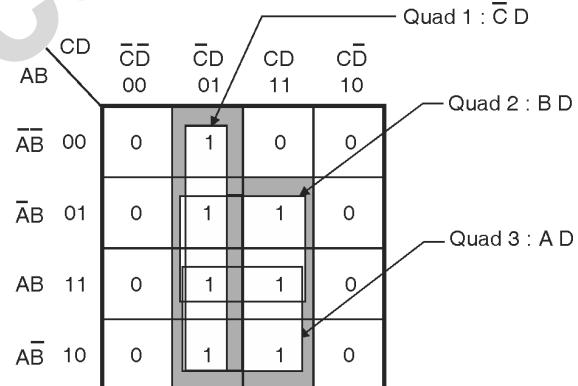
Realize the minimized expression using the basic gates.

Soln. :

- The given expression is,

$$Y = m_1 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$$

- It can be expressed on K-map as shown in Fig. P. 4.7.3(a).



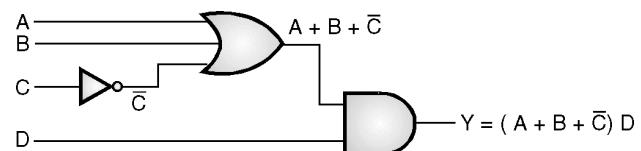
The simplified expression is as follows :

$$Y = \underbrace{\bar{C} D}_{\text{Quad 1}} + \underbrace{B \bar{D}}_{\text{Quad 2}} + \underbrace{A D}_{\text{Quad 3}} \quad \dots(1)$$

(C-252) Fig. P. 4.7.3(a)

Realization :

- Equation (1) can be realized as shown in Fig. P. 4.7.3(b).



(C-253) Fig. P. 4.7.3(b) : Realization with minimum number of gates



Ex. 4.7.4 : Minimize the following Boolean expression using K-map and realize it using the basic gates. $Y = \Sigma m(1, 3, 5, 9, 11, 13)$

Soln. :

- The given expression can be expressed in terms of the minterms as,

$$Y = m_1 + m_3 + m_5 + m_9 + m_{11} + m_{13}$$

- The corresponding K-map is shown in Fig. P. 4.7.4(a).

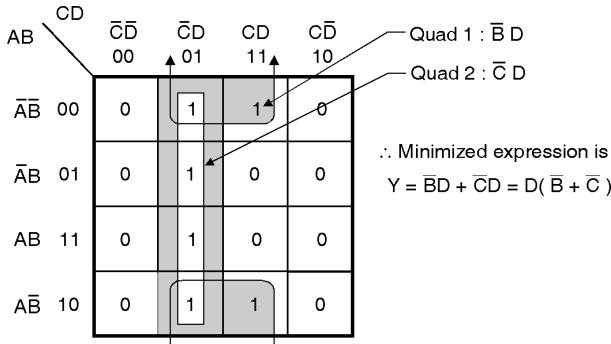
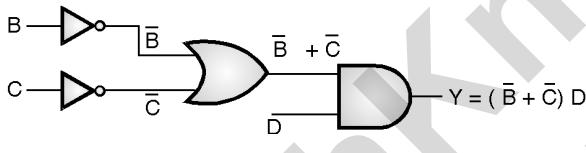


Fig. P. 4.7.4(a)

Realization :

- Fig. P. 4.7.4(b) shows realization using gates..



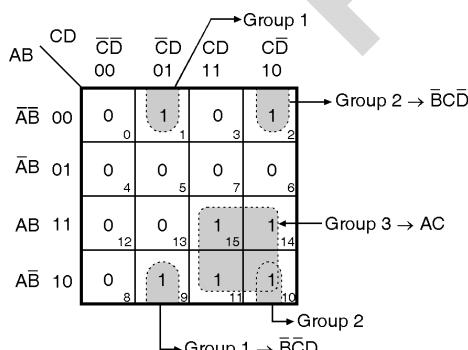
(C-254) Fig. P. 4.7.4(b)

Ex. 4.7.5 : Minimize the following expression using K-map and realize using the basic gates : $Y = \Sigma m(1, 2, 9, 10, 11, 14, 15)$

May 19, 6 Marks

Soln. :

Step 1 : K-map simplification :



(C-256) Fig. P. 4.7.5(a) : K-map simplification

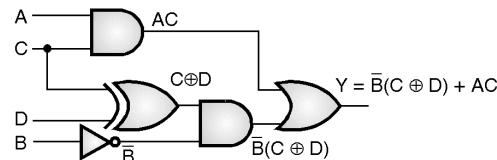
Minimized expression

$$Y = \bar{B}\bar{C}D + \bar{B}C\bar{D} + AC$$

$$= \bar{B}(CD + \bar{C}D) + AC$$

EX OR gate
 $\therefore Y = \bar{B}(C \oplus D) + AC$

Step 2 : Realization using gates :



(C-256) Fig. P. 4.7.5(b) : Realization with minimum number of gates

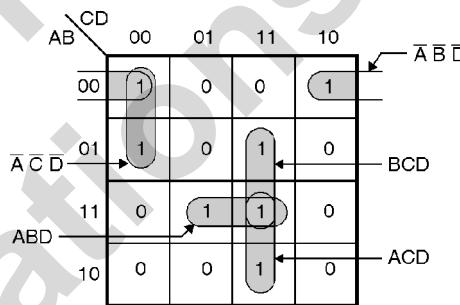
Ex. 4.7.6 : Solve the following using minimization technique :

$$z = f(A, B, C, D) = \Sigma (0, 2, 4, 7, 11, 13, 15)$$

Dec. 09, 10 Marks

Soln. :

Simplification using K-map :



(C-1692) Fig. P. 4.7.6

$$z = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{C}\bar{D} + ABD + BCD + ACD$$

Ex. 4.7.7 : Solve the following equations using corresponding minimization techniques, also draw MSI design for the minimized output equation :

$$Z = f(A, B, C, D) = \Sigma (0, 3, 4, 9, 10, 12, 14).$$

May 10, 12 Marks

Soln. : Solve it yourself.

Ans. :

$$Z = f(A, B, C, D) = \bar{A}\bar{C}\bar{D} + B\bar{C}\bar{D} + A\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{B}\bar{C}D$$

Ex. 4.7.8 : Using K-map realize the following expression using minimum number of gates.

$$Y = \Sigma m(1, 3, 4, 5, 7, 9, 11, 13, 15)$$

Soln. : Solve it yourself.

Ans. : $Y = A'BC' + D$

Ex. 4.7.9 : Minimize the following expression and realize using basic gates.

$$Y = \Sigma m(0, 2, 5, 6, 7, 8, 10, 13, 15)$$



Soln. : Solve it yourself.

Ans. : $Y = \bar{B}\bar{D} + BD + \bar{A}\bar{B}C$

4.7.1 Elimination of a Redundant Group :

- If all the 1's in a group are already being used in some other groups, then that group is called as a redundant group.
- A redundant group has to be eliminated, because it increases the number of gates required.
- Effect of a redundant group and its elimination is illustrated in the Ex. 4.7.8.

Ex. 4.7.10 : Minimize the following expression using the K-map.

$$Y = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$$

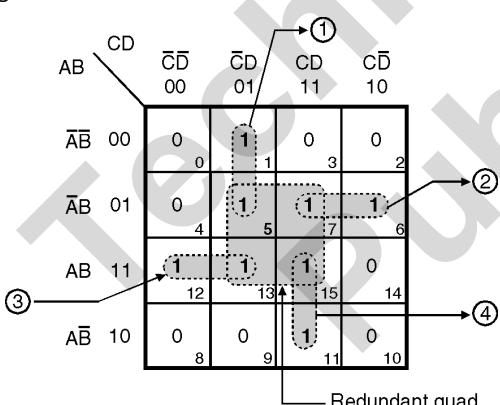
Soln. :

- The given expression can be expressed in the standard SOP form as follows :

$$Y = m_1 + m_5 + m_6 + m_7 + m_{11} + m_{12} + m_{13} + m_{15}$$

where m_1, m_5, \dots, m_{15} are the minterms.

- The required K-map is shown in Fig. P. 4.7.10(a).
- There are no isolated 1's. So encircle the separate pairs as shown in Fig. P. 4.7.10(a).
- We would visualize a quad shown by dotted lines in Fig. P. 4.7.10(a).



(C-258) Fig. P. 4.7.10(a)

- The question is should we encircle this quad ?
- The answer will be obtained by writing the expression of Y without and with this quad as follows :

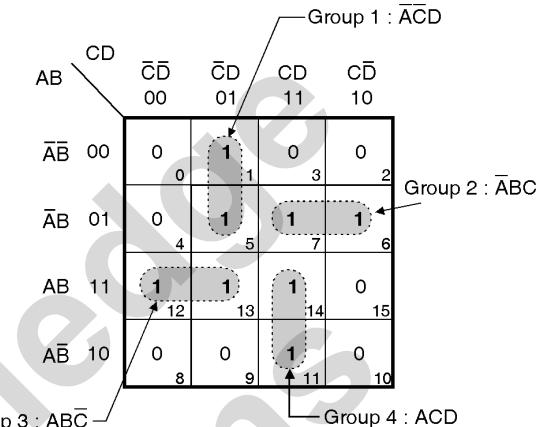
Expression without quad : $Y = \bar{A}\bar{C}D + \bar{A}BC + ABC + ACD$
(1) (2) (3) (4)

Expression with quad : $Y = \bar{A}\bar{C}D + \bar{A}BC + ABC + ACD + BD$
(1) (2) (3) (4)
Redundant quad

(C-6159)

Conclusion :

- If we encircle the quad, then the expression for output consists of an additional term. So quad should not be encircled. It is called as the **redundant group**.
- Hence the correct K-map is shown in Fig. P. 4.7.10(b).



(C-259) Fig. P. 4.7.10(b)

$$\begin{aligned} \therefore Y &= \bar{A}\bar{C}D + \bar{A}BC + A\bar{B}C + ACD \\ &= \bar{A}\bar{C}D + ACD + \bar{A}BC + A\bar{B}C \\ &= D \underbrace{(\bar{A}C + AC)}_{\text{EX-NOR}} + B \underbrace{(\bar{A}C + A\bar{C})}_{\text{EX-OR}} \end{aligned}$$

(C-6160)

$$\therefore Y = D(A \oplus C) + B(A \oplus C)$$

4.7.2 Minimization of Logic Functions not Specified in Standard SOP Form :

- Till now we have seen how to use the K-map to minimize an expression which is given in standard SOP form.
- Now let us see the use of K-map for minimization when the given expression is not in the standard form. The procedure to be followed under such conditions is as follows :
- The given expression,

$$Y = \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{D} + A\bar{C} + \bar{B}$$

Procedure :

- | |
|---|
| Step 1 : Enter 1 for minterms (i.e. $\bar{A}\bar{B}CD$ and $\bar{A}BC\bar{D}$) present in the given expression. |
| Step 2 : Enter a pair of 1's for each term with one less variable than total (i.e. for $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}\bar{D}$ because these terms have one less variable) |
| Step 3 : Enter four adjacent 1's for each term with two less variables than total (i.e. $A\bar{C}$). |



Step 4 : Repeat for the other terms in a similar way.

Step 5 : Once the K-map is obtained, the minimization procedure is same as the one discussed earlier.

- This procedure will be clear as you solve the example given below.

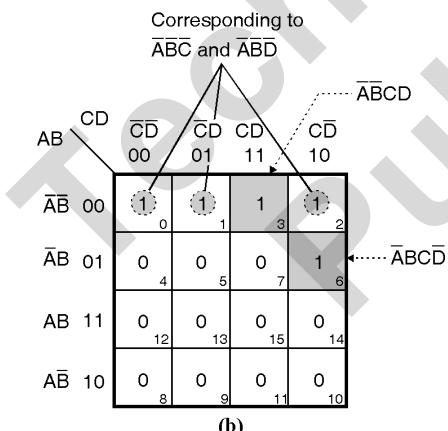
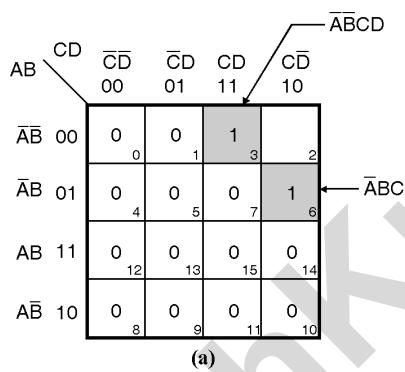
Ex. 4.7.11 : Minimize the logic equation given below :

$$\therefore Y = \overline{ABC}D + \overline{AB}\overline{CD} + \overline{ABC} + \overline{ABD} + \overline{AC} + \overline{B}$$

(1) (2) (3) (4) (5) (6) (C-6357)

Soln. :

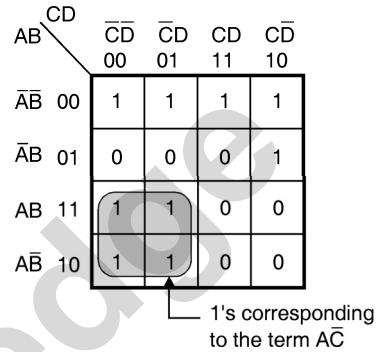
Step 1 : Enter 1 in the cell with $A = 0, B = 0, C = 1, D = 1$ for the first term $\overline{A}\overline{B}\overline{C}D$ and in the cell with $A = 0, B = 1, C = 1, D = 0$ for the second term $\overline{A}\overline{B}\overline{C}\overline{D}$ as shown in Fig. P. 4.7.11(a).



(C-261) Fig. P. 4.7.11

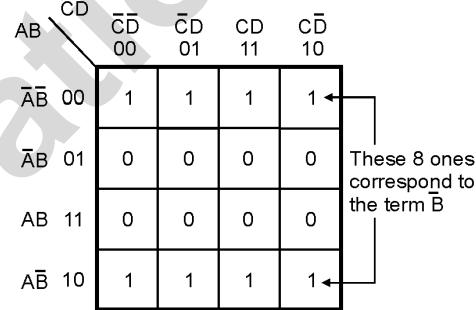
Step 2 : Consider the third term $\overline{A}\overline{B}\overline{C}$. Enter 1 in two cells which correspond to $A = 0, B = 0$ and $C = 0$, as shown in Fig. P. 4.7.11(b). Now consider the fourth term $\overline{A}\overline{B}\overline{D}$ and enter 1 in two cells which correspond to $A = 0, B = 0$ and $D = 0$ as shown in Fig. P. 4.7.11(b).

Step 3 : Now consider the fifth term \overline{AC} . Enter 1 in four cells corresponding to $A = 1$ and $C = 0$, as shown in Fig. P. 4.7.11(c).



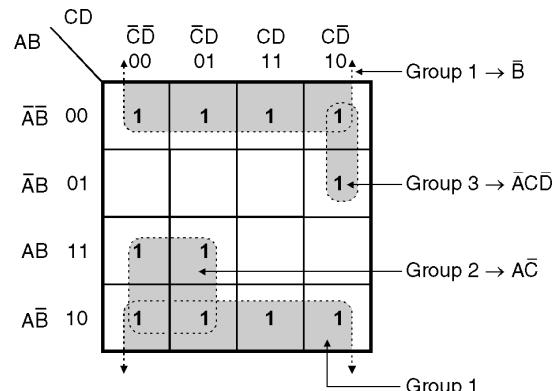
(C-262) Fig. P. 4.7.11(c)

Step 4 : Finally corresponding to the sixth term, enter eight 1's corresponding to $B = 0$ as shown in Fig. P. 4.7.11(d).



(C-262) Fig. P. 4.7.11(d)

- The final K-map is as shown in Fig. P. 4.7.11(e). Use the normal simplification techniques discussed earlier to simplify this K-map and get the minimized expression.



(C-263) Fig. P. 4.7.11(e) : Final K-map

- The simplified equation is,

$$Y = \overline{B} + \overline{AC} + \overline{ACD}$$



4.7.3 Don't Care Conditions :

- For SOP form, we enter 1's corresponding to the combinations of input variables which produce a high output. And we enter 0's in the remaining cells of the K-map.
- For the POS form we enter 0's corresponding to the combinations of inputs which produce a low output and enter 1's in the remaining cells of the K-map.
- But it is not always true that the cells not containing 1's (in SOP) will contain 0's, because some combinations of input variable do not occur.
- Take the example of a 4 bit BCD counter. It will have valid outputs from 0000 to 1001 only.
- Also for some functions the outputs corresponding to certain combinations of input variables **do not matter**.
- That means for such input combinations it does not matter whether the value of output is 0 or 1.
- In such situation we are free to assume a 0 or 1 as output for each of such input combinations.
- These conditions are known as the "**Don't care conditions**" and in the K-map it is represented as \times (cross) mark in the corresponding cell.

Important :

- The don't care condition (\times) may be assumed to be 0 or 1 as per the need for simplification.

Ex. 4.7.12 : Simplify the expression given below using K-map. The don't care conditions are indicated by d ().

$$Y = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5).$$

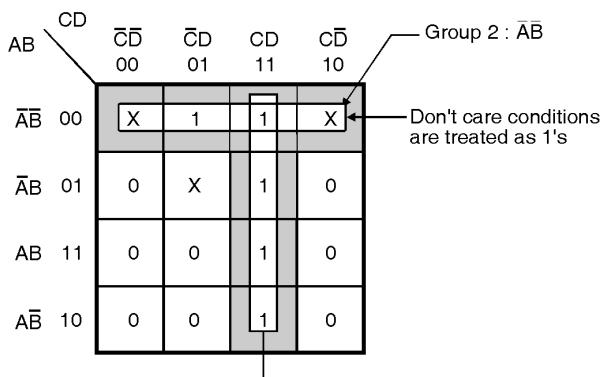
Dec. 12, 5 Marks

Soln. : The given equation is,

$$Y = \underbrace{m_1 + m_3 + m_7 + m_{11} + m_{15}}_{\text{Regular minterms so enter 1's}} + \underbrace{d(0, 2, 5)}_{\text{Don't care conditions so enter } \times \text{ marks}} \quad (\text{C-6161})$$

Regular minterms so enter 1's Don't care conditions so enter \times marks

- The required K-map is shown in Fig. P. 4.7.12.



(C-264) Fig. P. 4.7.12

- Simplified expression $Y = CD + \bar{A}\bar{B}$

Note : Every don't care mark need not be considered while grouping.

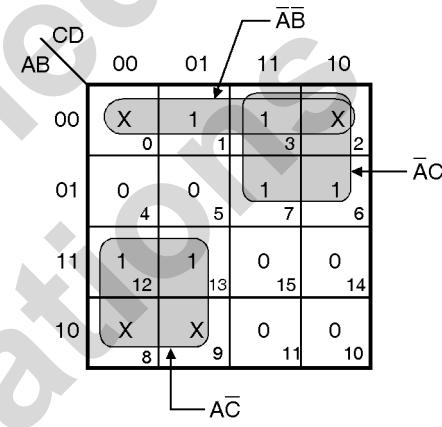
Ex. 4.7.13 : Solve the following equation using K map minimization technique. Draw the MSI design for the minimized output :

$$Z = f(A, B, C, D) = \Sigma m(1, 3, 6, 7, 12, 13) + d(0, 2, 8, 9)$$

May 12, 6 Marks

Soln. :

Step 1 : Reduction using K-map :

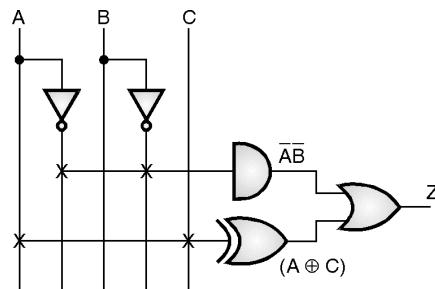


$$\therefore Z = \bar{A}\bar{B} + \bar{A}C + A\bar{C} \\ = \bar{A}\bar{B} + (A \oplus B)$$

(C-3224) Fig. P. 4.7.13(a)

Step 2 : MSI design :

- Implementation using logic gates is as shown in Fig. P. 4.7.13(b).



(C-3225) Fig. P. 4.7.13(b)

Ex. 4.7.14 : Minimize the following function using K-map and implement using basic logic gates.

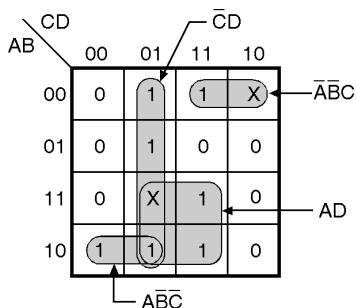
$$f(A, B, C, D) = \Sigma m(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$$

Dec. 17, 6 Marks



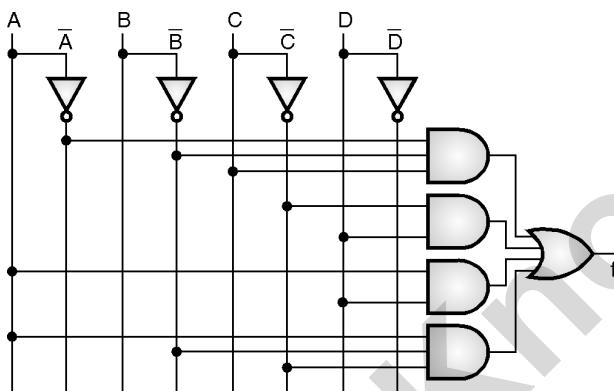
Soln. :

Step 1 : Minimization using k-map : (C-6329)



$$\therefore f(A, B, C, D) = \bar{A}\bar{B}C + \bar{C}D + AD + AB\bar{C}$$

Step 2 : Implementation using basic gates :



(C-6330) Fig. P. 4.7.14(a) : Implementation using basic gates

Ex. 4.7.15 : Solve the following reduction using K-map, also draw MSI circuit for the output :

1. $Z = f(A, B, C, D) = \sum(1, 2, 7, 8, 10, 12, 15) + d(0, 5, 6)$
2. $Z = f(A, B, C, D) = \sum(1, 3, 4, 6, 8, 11, 15) + d(0, 5, 7)$

May 11, 12 Marks

Soln. : Solve it yourself.

Ex. 4.7.16 : Solve the following equation using corresponding minimization technique. Draw the diagram for the output :

$$Z = f(A, B, C, D) = \sum m(2, 4, 6, 11, 12, 14) + d(3, 10).$$

Dec. 11, 6 Marks

Soln. : Solve it yourself.

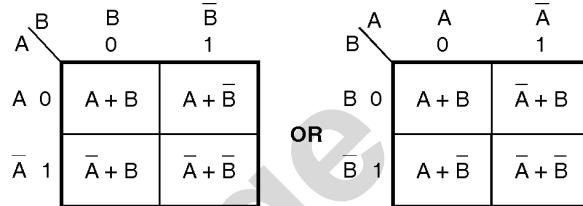
4.8 Product of Sum (POS) Simplification :

4.8.1 K-map Representation of POS Form :

- The POS form equations consist of maxterms.

- In the K-map corresponding to POS form we have to enter a 0 corresponding to each maxterm.

- The K-maps corresponding to the POS form are shown in Fig. 4.8.1.



(C-286) Fig. 4.8.1(a) : Two variable K-map for POS form

	BC	00	01	11	10
0	$A+B+C$	$A+B+\bar{C}$	$A+\bar{B}+\bar{C}$	$A+\bar{B}+C$	
1	$\bar{A}+B+C$	$\bar{A}+B+\bar{C}$	$\bar{A}+\bar{B}+C$	$\bar{A}+\bar{B}+\bar{C}$	

OR

	AB	00	01	11	10
0	$A+B+C$	$A+\bar{B}+C$	$\bar{A}+\bar{B}+C$	$\bar{A}+B+C$	
1	$\bar{A}+B+\bar{C}$	$A+\bar{B}+\bar{C}$	$\bar{A}+\bar{B}+\bar{C}$	$\bar{A}+B+\bar{C}$	

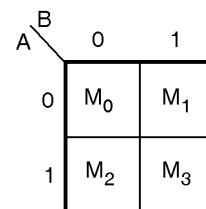
(C-286) (b) Three variable K-map for POS form

	CD	00	01	11	10
00	$A+B+C+D$	$A+B+C+\bar{D}$	$A+B+\bar{C}+\bar{D}$	$A+B+\bar{C}+D$	
01	$A+\bar{B}+C+D$	$A+\bar{B}+C+\bar{D}$	$A+\bar{B}+\bar{C}+\bar{D}$	$A+\bar{B}+\bar{C}+D$	
11	$\bar{A}+\bar{B}+C+D$	$\bar{A}+\bar{B}+C+\bar{D}$	$\bar{A}+\bar{B}+\bar{C}+\bar{D}$	$\bar{A}+\bar{B}+\bar{C}+D$	
10	$\bar{A}+B+C+D$	$\bar{A}+B+C+\bar{D}$	$\bar{A}+B+\bar{C}+\bar{D}$	$\bar{A}+B+\bar{C}+D$	

(c) Four variable K-map for POS form

(C-286) Fig. 4.8.1

- The entries in the K-map can be shown in terms of the maxterms M_0, M_1, \dots etc. as shown in Fig. 4.8.2.



(a)

Fig. 4.8.2(Contd...)



		BC	00	01	11	10
		A	M ₀	M ₁	M ₃	M ₂
		0	0	1	1	0
		1	M ₄	M ₅	M ₇	M ₆
		1	1	1	1	0

(b)

		CD	00	01	11	10
		AB	M ₀	M ₁	M ₃	M ₂
		00	0	1	1	0
		01	M ₄	M ₅	M ₇	M ₆
		11	M ₁₂	M ₁₃	M ₁₅	M ₁₄
		10	M ₈	M ₉	M ₁₁	M ₁₀

(c)

(C-287) Fig. 4.8.2 : K-map in terms of maxterms

4.8.2 Representation of Standard POS form on K-map :

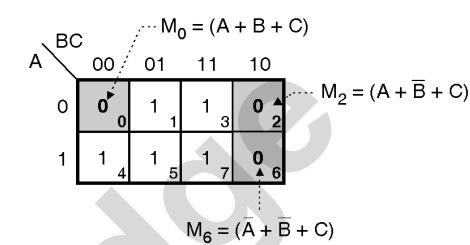
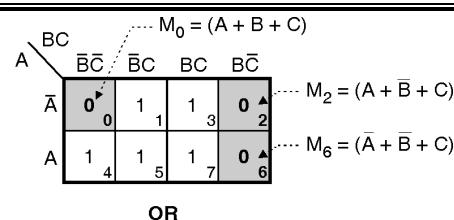
- Logical expressions in the standard POS form can be represented on K-map by entering 0's in the cells of K-map corresponding to each maxterm present in the given equation.
- The remaining cells are filled with 1's.
- This technique is illustrated in Ex. 4.8.1.

Ex. 4.8.1 : Represent the following standard POS expression on Karnaugh map.

$$Y = (A + B + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + C).$$

Soln. :

- Each term in the given logical equation is a maxterm.
- Enter a 0 corresponding to each maxterm as shown in Fig. P. 4.8.1.
- The given expression has three maxterms as follows,
 $(A + B + C) = M_0,$
 $(A + \bar{B} + C) = M_2,$
 $(\bar{A} + \bar{B} + C) = M_6$
- Hence we have to write the structure of 3 variable K-map as usual and enter 0's at M_0, M_2 and M_6 as shown in Fig. P. 4.8.1.



(C-288) Fig. P. 4.8.1 : Representation of standard POS on K-map

Ex. 4.8.2 : Represent the following standard POS equation on the Karnaugh map.

$$Y = (A + B + C + \bar{D})(A + B + \bar{C} + \bar{D}) \\ (A + \bar{B} + \bar{C} + \bar{D})(\bar{A} + \bar{B} + C + D)$$

Soln. :

- The given expression contains four maxterms as follows :

$$(A + B + C + \bar{D}) = M_1$$

$$(A + \bar{B} + \bar{C} + \bar{D}) = M_7$$

$$(A + B + \bar{C} + \bar{D}) = M_3$$

$$(\bar{A} + \bar{B} + C + D) = M_{12}$$

- Enter a 0 corresponding to each maxterm as shown in Fig. P. 4.8.2.

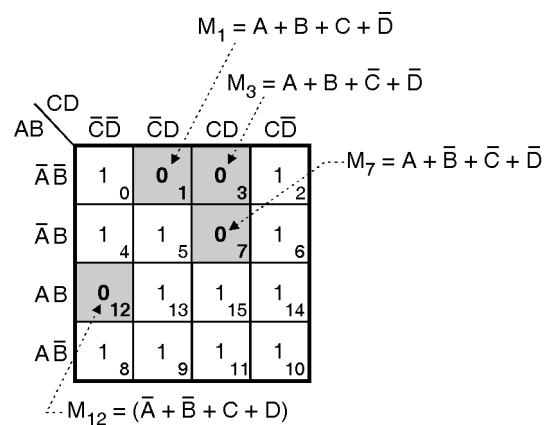
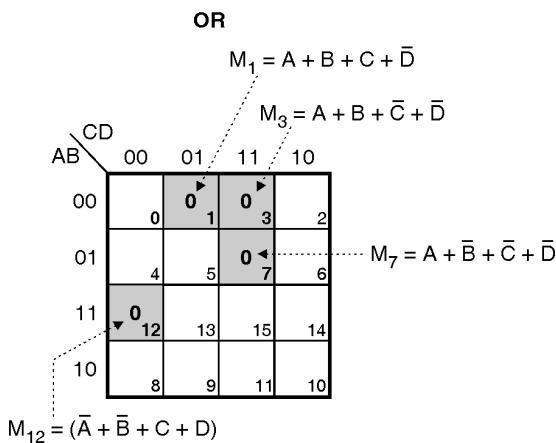


Fig. P. 4.8.2(Contd...)



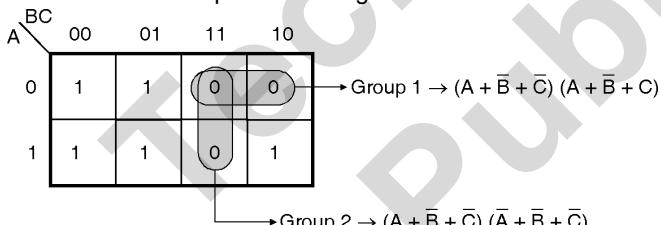
(C-289) Fig. P. 4.8.2 : Representation of standard POS on K-map

4.8.3 Simplification of Standard POS Form using K-map :

Minimization procedure :

1. The given POS expression consists of maxterms.
2. Corresponding to every maxterm enter a 0 in the K-map.
3. Enter 1's in the remaining cells of K-map.
4. Encircle/group 0's instead of 1's for carrying out the simplification.
5. Rules of simplification are exactly same as those used for the SOP form.

Ex. 4.8.3 : Find the expression in the POS form for the K-map shown in Fig. P. 4.8.3.



(C-290) Fig. P. 4.8.3 : Given K-map

Soln. :

Simplification :

$$\begin{aligned}
 \text{Group 1} &\rightarrow (A + \bar{B} + \bar{C}) (A + \bar{B} + C) \\
 &= AA + A\bar{B} + AC + AB + \bar{B}\bar{B} + \bar{B}C + A\bar{C} + \bar{B}\bar{C} + CC \\
 \text{But } AA &= A, \bar{B}\bar{B} = \bar{B}, CC = 0, AB + A\bar{B} = AB \\
 \therefore \text{Group 1} &= A + A\bar{B} + AC + \bar{B} + \bar{B}C + A\bar{C} + \bar{B}\bar{C} \\
 &= A(1 + \bar{B}) + A(C + \bar{C}) + \bar{B}(1 + C) + \bar{B}\bar{C} \\
 &= A + A + \bar{B} + \bar{B}\bar{C} \quad (\text{C-6358}) \\
 &= A + \bar{B}(1 + \bar{C})
 \end{aligned}$$

$$\text{Group 1} = A + \bar{B}$$

Conclusion :

- When a pair of 0s is formed, the variable which changes will get eliminated e.g. C changes for the pair of 0's in group 1. Hence it is eliminated.

$$\text{Group 2} \rightarrow (A + \bar{B} + \bar{C}) (\bar{A} + \bar{B} + \bar{C})$$

..... Remain unchanged

..... Variable that changes its value hence it gets eliminated

(C-291)

$$\therefore \text{Group 2} \rightarrow \bar{B} + \bar{C}$$

\therefore Final minimized equation is as follows :

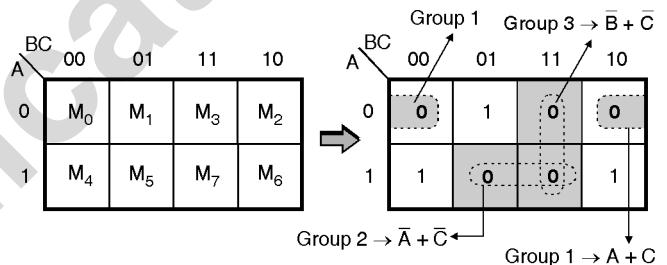
$$Y = (A + \bar{B})(\bar{B} + \bar{C})$$

(1) (2) (C-291(a))

Ex. 4.8.4 : Minimize the following standard POS expression using K-map :

$$Y = \prod M(0, 2, 3, 5, 7)$$

Soln. :



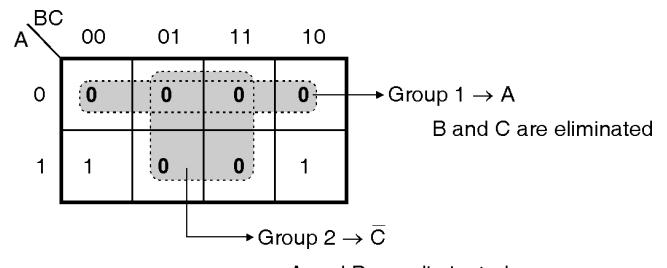
(C-292) Fig. P. 4.8.4

\therefore Minimized expression is as follows :

$$Y = (A + C)(\bar{A} + \bar{C})(\bar{B} + \bar{C})$$

...Ans.

Ex. 4.8.5 : Find the expression for the output in the POS form for the K-map shown in Fig. P. 4.8.5.



(C-293) Fig. P. 4.8.5 : Given K-map

Soln. :

- Minimized expression is as follows :



$$Y = A \cdot \bar{C}$$

Group 2
Group 1

(C-293(a))

Ex. 4.8.6 : Write the expression for output in the POS form for the K-map shown in Fig. P. 4.8.6.

		CD	00	01	11	10
AB		00	0	1	1	0
		01	1	1	1	1
		11	1	1	1	1
		10	0	1	1	0

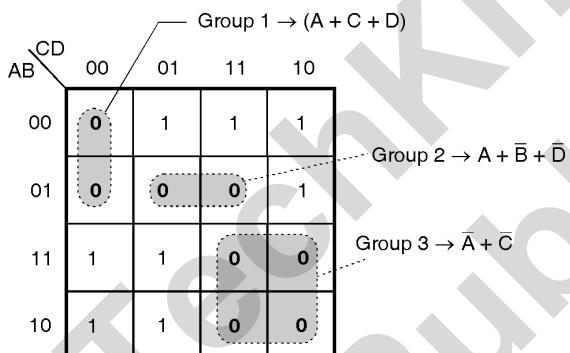
The 0s in four corners form a quad. Hence 2-variables which change will get eliminated

(C-295) Fig. P. 4.8.6

Soln. :

$$\text{Expression for output } Y = B + D$$

Ex. 4.8.7 : For the K-map shown in Fig. P. 4.8.7 write the expression for output in the POS form.



(C-296) Fig. P. 4.8.7

Soln. :

- The expression for output is,

$$Y = (A + C + D) (A + \bar{B} + \bar{D}) (\bar{A} + \bar{C})$$

(1) (2) (3) (C-6361)

Note : Sometimes there are more than one correct ways of encircling the 0's, and the answers obtained as a result of all of them are correct. This is illustrated in Ex. 4.8.8.

Ex. 4.8.8 : Group the 0s given in the K-map of the Fig. P. 4.8.8(a) in different ways to obtain the expression for output in the POS form.

		CD	00	01	11	10
AB		00	1	0	1	1
		01	1	0	0	0
		11	1	1	1	0
		10	0	0	1	0

(C-297) Fig. P. 4.8.8(a) : Given K-map

Soln. :

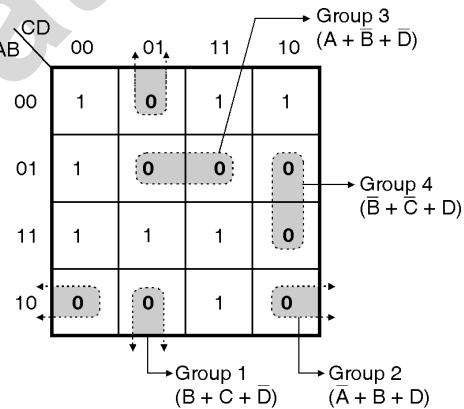
- Figs. P. 4.8.8(b) and (c) shows two different but valid ways of grouping 0's.
- The corresponding equations for the output in the POS form are obtained after that.
- Referring to the grouping shown in Fig. P. 4.8.8(b) we get,

$$Y = (B + C + \bar{D}) (\bar{A} + B + D) (A + \bar{B} + \bar{D}) (\bar{B} + \bar{C} + D)$$

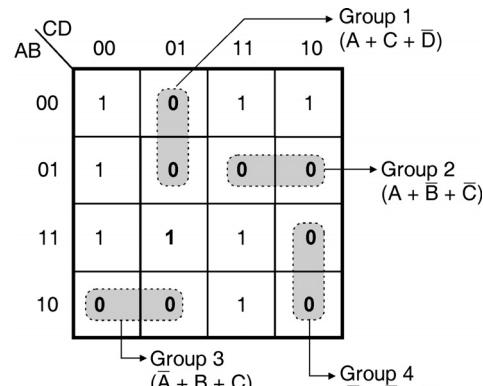
(1) (2) (3) (4)

(C-6359)

- This shows that for the same K-map we can get completely different answers and all of them are correct.



(b) One way of grouping

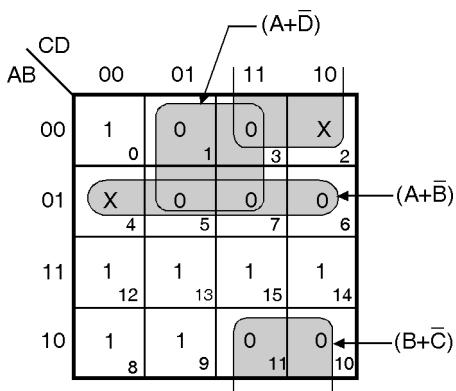


(c) Another way of grouping

(C-298) Fig. P. 4.8.8



Ex. 4.8.9 : Solve the following equation using corresponding minimization technique. Draw the MSI design for the minimized output :
 $Z = f(A, B, C, D) = \prod M(1, 3, 5, 6, 7, 10, 11) + d(2, 4)$

May 12, 6 Marks**Soln. :****Step 1 : Simplification using K-map :**

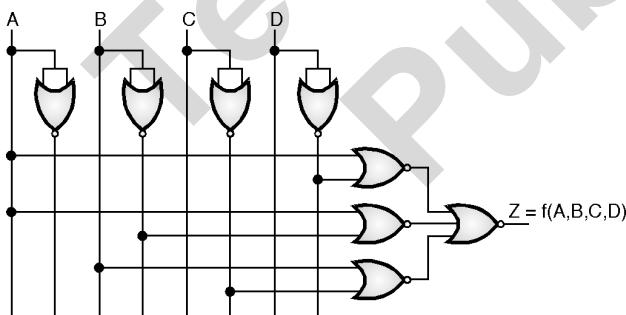
(C-3226) Fig. P. 4.8.9(a)

$$\therefore Z = (A + \bar{D}) \cdot (A + \bar{B}) \cdot (B + \bar{C})$$

Step 2 : Realization using NOR gates :**Taking double inversion of R.H.S**

$$Z = (\overline{A + \bar{D}}) \cdot (\overline{A + \bar{B}}) \cdot (\overline{B + \bar{C}})$$

$$Z = (\overline{A + \bar{D}}) + (\overline{A + \bar{B}}) + (\overline{B + \bar{C}})$$

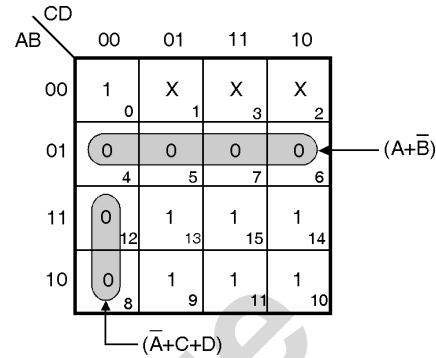


(C-3227) Fig P. 4.8.9(b) : Implementation using NOR

Ex. 4.8.10 : Minimize the following four variable functions using K-map. 'd' represents don't care conditions : $f(A, B, C, D) = \prod M(4, 5, 6, 7, 8, 12) + d(1, 2, 3)$

Dec. 12, 5 Marks**Soln. :**

- Required K-map is as follows :

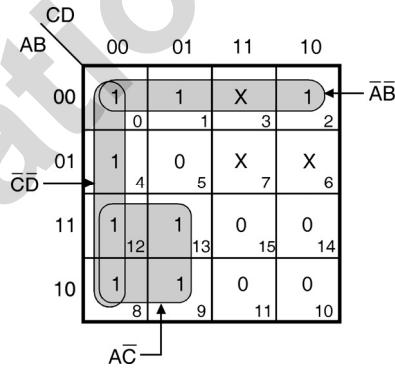


(C-3228) Fig. P. 4.8.10

$$\therefore f(A, B, C, D) = (A + \bar{B})(\bar{A} + C + D) \quad \dots \text{Ans.}$$

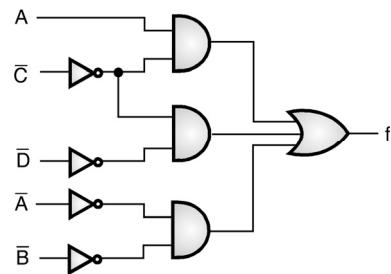
Ex. 4.8.11 : Minimize the following function using K-map and implement using basic logic gates :

$$f(A, B, C, D) = \sum m(0, 1, 2, 4, 8, 9, 12, 13) + d(3, 6, 7)$$

May 14, 6 Marks**Soln. :****Step 1 : K-map :**

(C-4805) Fig. P. 4.8.11(a)

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

Step 2 : Implementation using basic gates :

(C-4805) Fig. P. 4.8.11(b)

Ex. 4.8.12 : What is De Morgan's theorem ? Solve the following using minimization technique :

$$z = f(A, B, C, D) = \prod(1, 2, 3, 6, 8, 11, 14, 15).$$

Dec. 09, 10 Marks



Soln. : Solve it yourself.

Ex. 4.8.13 : Solve the following equations using corresponding minimization techniques, also draw MSI design for the minimized output equation : $Z = f(A, B, C, D) = \prod(2, 7, 8, 10, 11, 13, 15)$ **May 10, 12 Marks**

Soln. : Solve it yourself.

Ex. 4.8.14 : Solve the following equation using K map minimization technique. Draw the diagram for the output : $Z = f(A, B, C, D) = \sum m(0, 1, 6, 7, 8, 9)$ **Dec. 11, 6 Marks**

Soln. : Solve it yourself.

Review Questions

Q. 1 Explain the following terms :

1. Product term
2. Sum term

Q. 2 Explain the term SOP and POS related to Boolean function.

Q. 3 Convert the equation into standard POS form
$$Y = (A + B)(A + C)(B + C)$$

Q. 4 State the disadvantages of algebraic method of simplification.

Q. 5 Write the standard SOP equation for the truth table shown in Table 1.

Table 1

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Q. 6 For the same truth table write the standard POS expression.
- Q. 7 Explain the K-map reduction technique.
- Q. 8 Draw the structure of four input K-map.
- Q. 9 Why is gray code followed for K-map?
- Q. 10 Define a redundant group.
- Q. 11 State the importance of don't care terms in K-map.
- Q. 12 Draw the structure of four variable K-map to represent the standard POS form.
- Q. 13 Solve the following with K-maps :
1. $f(A,B,C) = \sum m(0,1,3,4,5)$
 2. $f(A,B,C) = \sum m(0,1,2,3,6,7)$
- Q. 14 Explain the different methods used to simplify the Boolean function.

Unit 2

Chapter 5

Combinational Logic Design

Syllabus

Design using SSI chips : Code converters, Half- adder, Full adder, Half subtractor, Full subtractor, n bit binary adder.

Introduction to MSI chips : Multiplexer (IC 74153), Demultiplexer (IC 74138), Decoder (74238) Encoder (IC 74147), Binary adder (IC 7483).

Design using MSI chips : BCD adder & subtractor using IC 7483, Implementation of logic functions using IC 74153 & 74138.

Case Study : Use of combinational logic design in 7 segment display interface.

Chapter Contents

5.1	Introduction to Combinational Circuits	5.11	Study of Different Multiplexer ICs
5.2	Design of Combinational Logic using SSI chips	5.12	Multiplexer Tree/Cascading of multiplexer
5.3	Binary Adders and Subtractors	5.13	Use of Multiplexers in Combinational Logic Design
5.4	The n-Bit Parallel Adder	5.14	Demultiplexers
5.5	n-bit Parallel Subtractor	5.15	Types of Demultiplexers
5.6	BCD Addition	5.16	Demultiplexer Tree
5.7	BCD Subtractor using MSI IC 7483	5.17	Encoders
5.8	Magnitude Comparators	5.18	Priority Encoder
5.9	Multiplexer (Data Selector)	5.19	Decoder
5.10	Types of Multiplexers	5.20	Case Study : Combinational Logic Design of BCD to 7 Segment Display Controller

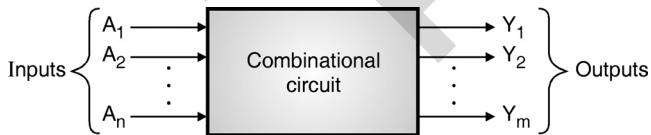
5.1 Introduction to Combinational Circuits :

Types of digital circuits :

- The digital systems in general are classified into two categories namely :
 1. Combinational logic circuits.
 2. Sequential logic circuits.

Combinational circuits :

- The combinational circuit is a digital system the output of which at any instant of time, depends only on the levels present at its input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied also does not have any effect on the output of a combinational circuit.
- A combinational circuit is a logic circuit the output of which depends only on the combination of the inputs.
- The output does not depend on the past value of inputs or outputs.
- Hence combinational circuits do not require any memory (to store the past values of inputs or outputs).
- The block diagram of a combinational circuit is shown in Fig. 5.1.1.
- A combinational circuit can have a number of inputs and a number of outputs. The circuit of Fig. 5.1.1 has "n" inputs and "m" outputs.



(C-332) Fig. 5.1.1 : Block diagram of a combinational circuit

- A combinational circuit is made up of logic gates. These gates are connected suitably between the inputs and outputs of the combinational circuit.
- A combinational circuit operates in three steps :
 1. It accepts n-different inputs.
 2. The combination of gates operates on the inputs.
 3. "m" different outputs are produced as per requirement.

Examples of combinational circuits :

- Following are some of the examples of combinational circuits :
- 1. Adders, subtractors. 2. Comparator.
- 3. Code converters. 4. Encoders, decoders.
- 5. Multiplexers, demultiplexers.

5.1.1 Analysis of a Combinational Circuit :

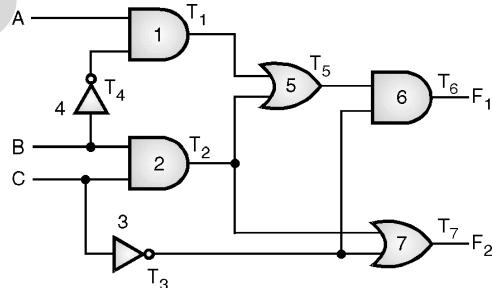
- In the analysis problem, the logic diagram of a combinational circuit is given to us.
- We have to obtain the Boolean expression for its output or write a truth table or explain the operation of circuit.

Analysis Procedure :

- The general procedure for the analysis is as follows :
- 1. Write down the Boolean function for the output of each input gate in the given circuit.
- 2. Obtain the Boolean expression at all other gates.
- 3. Write down the Boolean expression for the final output in terms of the input variables.

Illustration :

- Analyze the combinational circuit shown in Fig. 5.1.2.



(C-333) Fig. 5.1.2 : Given combinational circuit

Analysis :

Step 1 : Boolean expressions for the outputs of all the input gates :

- Input gates are 1, 2, 3 and 4. The Boolean expressions for their outputs are as follows :

$$T_1 = \bar{AB} \quad T_3 = \bar{C}$$

$$T_2 = BC \quad T_4 = \bar{B}$$

Step 2 : Boolean expressions for remaining gates :

$$T_5 = T_1 + T_2 = \bar{AB} + BC$$

$$T_6 = F_1 = T_5 \cdot T_3 = (\bar{AB} + BC) \bar{C}$$



$$= \overline{ABC} + B\overline{C} = \overline{ABC}$$

$$T_7 = F_2 = T_2 + T_3 = BC + \overline{C}$$

Step 3 : Boolean expressions for final outputs :

$$F_1 = T_6 = \overline{ABC}$$

$$F_2 = T_7 = \overline{C} + BC$$

Step 4 : Write the truth table :

(C-8244) Table 5.1.1

Inputs			Outputs						
A	B	C	F ₁	F ₂	T ₁	T ₂	T ₃	T ₄	T ₅
0	0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	0	1	0
0	1	0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	1	0	1	1	1
1	0	1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	1	0	0
1	1	1	0	1	0	1	0	0	1

- Note that the outputs of all the gates (T_1 to T_5) are first written in the truth table for various combinations of inputs and then the values of F_1 and F_2 are entered using the following equations,

$$F_1 = T_3 \cdot T_5 \text{ And } F_2 = T_2 + T_3$$

5.1.2 Design of Combinational Logic using SSI Chips :

- SSI is small signal integration. The SSI include the gate ICs. Hence this design procedure is used to implement the designed circuit using gates.
- The steps involved in designing a combinational logic are as follows :

Steps to be followed :

Step 1 : You will be given a problem.

Step 2 : Determine the number of inputs and outputs and assign letter symbols to input and output variables. For example $F_1, F_2 \dots$ for outputs and A, B, C, for the inputs.

Step 3 : Prepare a truth table relating the inputs and outputs.

Step 4 : Write K-map for each output in terms of inputs and obtain the simplified Boolean expression for each output.

Step 5 : Draw the logic diagram (combinational circuit).

Ex. 5.1.1 : A circuit has four inputs and two outputs. One of the outputs is high when majority of inputs are high. The second output is high only when all inputs are of same type. Design the combinational circuit.

OR

Design a logic circuit which has three inputs A, B, C and gives a high output when majority of inputs is high.

Soln. :

Step 1 : Assign symbols to input and output variables :

- Let the four inputs be A, B, C, D and the two outputs be Y_1 and Y_2 .

Step 2 : Write the truth table :

- The truth table is as given in Table P. 5.1.1.

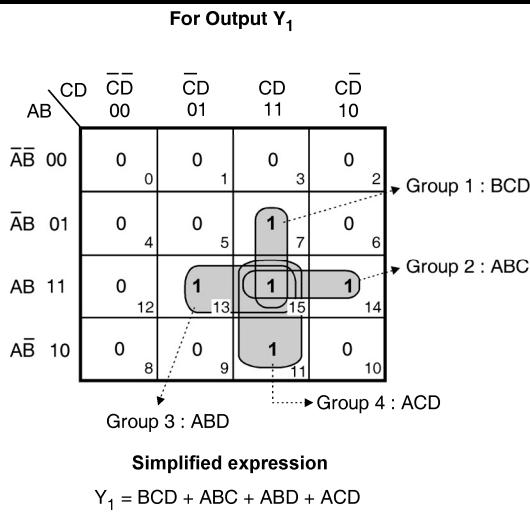
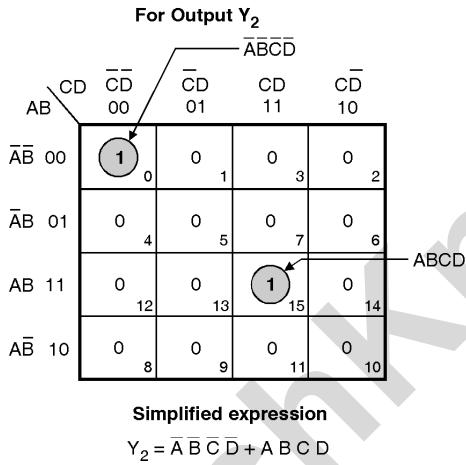
(C-8245) Table P. 5.1.1 : Truth table relating the inputs and outputs

Decimal	Inputs				Outputs	
	A	B	C	D	Y_1	Y_2
0	0	0	0	0	0	1
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	1	1	0	0
4	0	1	0	0	0	0
5	0	1	0	1	0	0
6	0	1	1	0	0	0
7	0	1	1	1	1	0
8	1	0	0	0	0	0
9	1	0	0	1	0	0
10	1	0	1	0	0	0
11	1	0	1	1	1	0
12	1	1	0	0	0	0
13	1	1	0	1	1	0
14	1	1	1	0	1	0
15	1	1	1	1	1	1

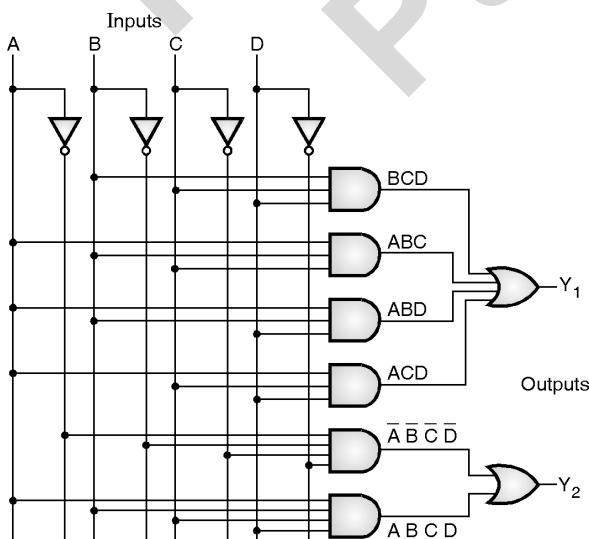
- From the truth table we note the following things.
- $Y_1 = 1$ when number of 1 inputs is higher than the number of 0 inputs.
- $Y_2 = 1$ when $A = B = C = D$.

Step 3 : Write K-map for each output and get simplified expression :

- K-maps for the two outputs and the corresponding simplified Boolean expressions are given in Figs. P. 5.1.1(a) and (b).

(C-334) Fig. P. 5.1.1(a) : K-map and simplification for Y_1 (C-334) Fig. P. 5.1.1(b) : K-map and simplification for Y_2 **Step 4 : Implement the logic diagram :**

- The logic diagram using logic gates is as shown in Fig. P. 5.1.1(c).



(C-335) Fig. P. 5.1.1(c) : Logic diagram

5.2 Design of Combinational Logic using SSI Chips :

- The combinational circuits to be designed using SSI chips i.e. logic gates are as follows :
 - Code converters
 - Adders and subtractors

5.2.1 Code Converters :

- In this section we are going to convert one type of code into another type.

5.2.1.1 BCD to Excess 3 Converter :

SPPU : May 06, Dec. 12

University Questions

- Q. 1** Design and implement BCD to Excess-3 code converter using logic gates. Starting with truth table show K-maps and circuit diagram of your design.
(May 06, 8 Marks)
- Q. 2** Design 4-bit BCD to excess-3 code converter. Use logic gates as per your design and requirement.
(Dec. 12, 8 Marks)

Principle :

- We know that Excess-3 code can be derived from the BCD code by adding 3 to each BCD number.
- For example decimal 13 is represented as 0001 0011 in BCD.
- If we add 3 i.e. (0011 0011) then the corresponding Excess 3 code is 0100 0110.

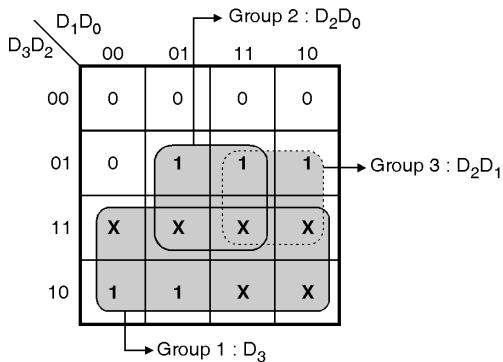
Step 1 : Write the truth table relating BCD and Excess 3 :

(C-8109) Table 5.2.1 : Truth table relating BCD and Excess-3 codes

Decimal	BCD inputs				Excess-3 output			
	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Step 2 : Write K-map for each output and obtain simplified expression :

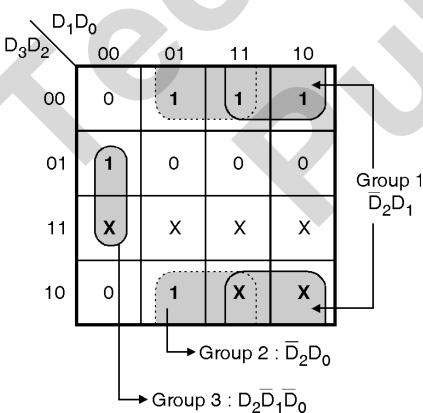
- Refer Fig. 5.2.1(a) to (d) for the K-maps and corresponding simplified equations.
- Don't care conditions (X) have been entered for $D_3 D_2 D_1 D_0$ above 1001.

For output E_3 (C-365) Fig. 5.2.1(a) : K-map for E_3

- Simplified equation :

$$E_3 = D_3 + D_2 D_0 + D_2 D_1$$

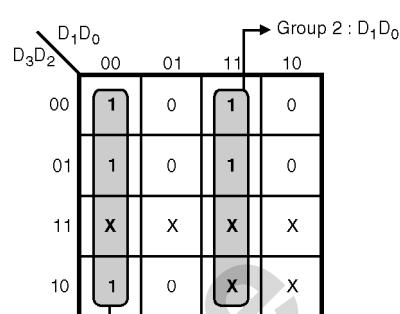
$$\therefore E_3 = D_3 + D_2 (D_0 + D_1)$$

For output E_2 (C-365) Fig. 5.2.1(b) : K-map for E_2

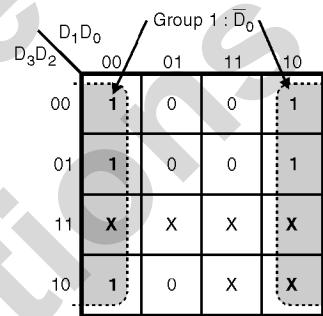
- Simplified equation :

$$E_2 = \bar{D}_2 D_1 + \bar{D}_2 D_0 + D_2 \bar{D}_1 \bar{D}_0$$

$$\therefore E_2 = \bar{D}_2 (D_1 + D_0) + D_2 \bar{D}_1 \bar{D}_0$$

For output E_1 (C-366) Fig. 5.2.1(c) : K-map for E_1

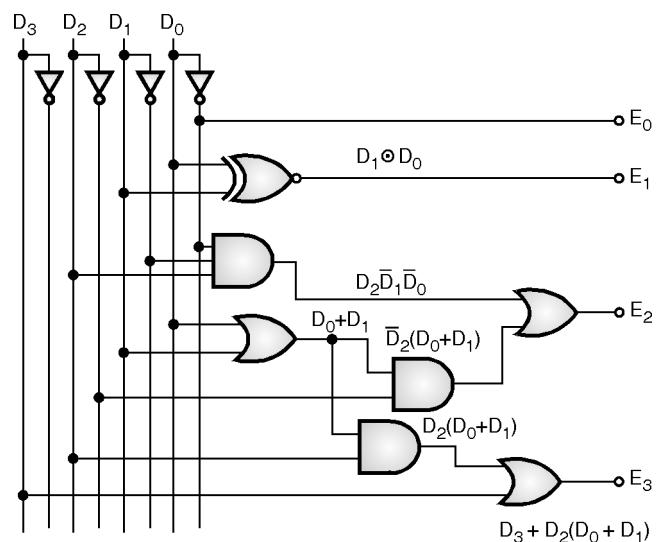
$$\therefore E_1 = \bar{D}_1 \bar{D}_0 + D_1 D_0 = (D_1 \odot D_0)$$

For output E_0 (C-366) Fig. 5.2.1(d) : K-map for E_0

$$\therefore E_0 = \bar{D}_0$$

Step 3 : Implement the BCD to Excess 3 code converter using gates :

- The circuit diagram of BCD to Excess 3 code converter is shown in Fig. 5.2.2.



(C-367) Fig. 5.2.2 : BCD to Excess - 3 code converter



5.2.2 BCD to Gray Code Converter :

SPPU : Dec. 06

University Questions

- Q. 1** Design and implement BCD to gray code converter using logic gates. Starting with truth table show K-maps and circuit diagram of your design.

(Dec. 06, 8 Marks)

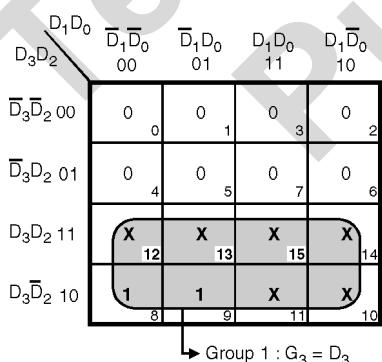
Step 1 : Write the truth table relating BCD and gray codes :

(C-8145) Table 5.2.2

Decimal	BCD inputs				Gray code			
	D ₃	D ₂	D ₁	D ₀	G ₃	G ₂	G ₁	G ₀
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

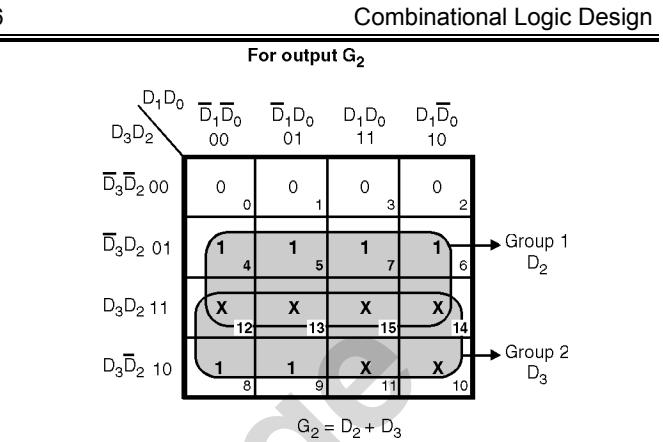
Step 2 : Write K-map for each output and get simplified equation :

For output G₃



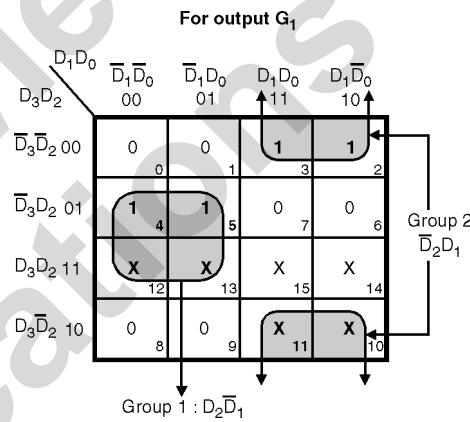
(C-371) Fig. 5.2.3(a) : K-map for G₃

$$\therefore G_3 = D_3$$



(C-371) Fig. 5.2.3(b) : K-map for G₂

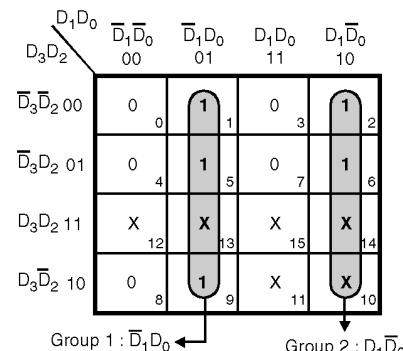
$$G_2 = D_2 + D_3$$



(C-372) Fig. 5.2.3(c) : K-map for G₁

$$\therefore G_1 = D_2D_1 + D_2D_1 = (D_2 \oplus D_1)$$

For output G₀

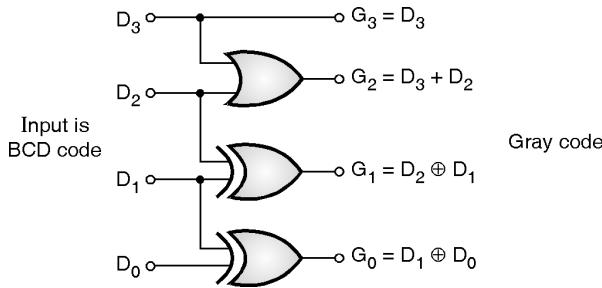


(C-372) Fig. 5.2.3(d) : K-map for G₀

$$\therefore G_0 = D_1D_0 + D_1D_0 = D_1 \oplus D_0$$

Step 3 : Realization using gates:

- The BCD to gray converter is shown in Fig. 5.2.4.



(C-373) Fig. 5.2.4 : BCD to Gray code converter

5.2.3 Binary to Gray Code Converter :

SPPU : May 12

University Questions

Q. 1 Design 4-bit binary to gray code converter. State the applications of gray code. **(May 12, 8 Marks)**

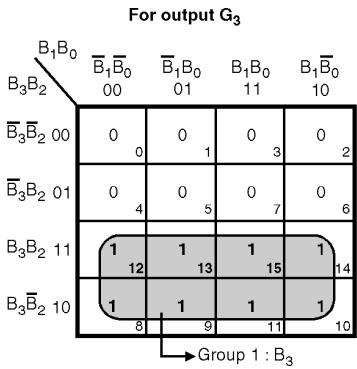
- In gray code only one bit changes at a time.

Step 1 : Write the truth table relating binary inputs and gray outputs :

(C-8252) Table 5.2.3 : Truth table relating binary and gray codes

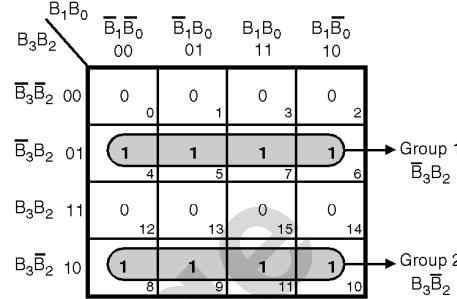
Decimal	Binary inputs				Gray outputs			
	B ₃	B ₂	B ₁	B ₀	G ₃	G ₂	G ₁	G ₀
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	1	0	0

Step 2 : Write K-map for each gray output and obtain the simplified expression :

(C-374) Fig. 5.2.5(a) : K-map for G₃

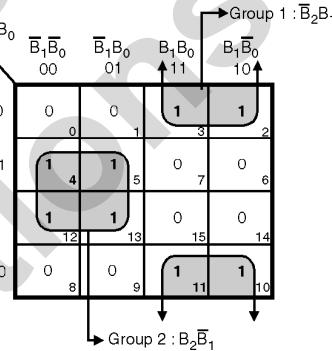
∴ Simplified equation : G₃ = B₃

For output G₂

(C-374) Fig. 5.2.5(b) : K-map for G₂

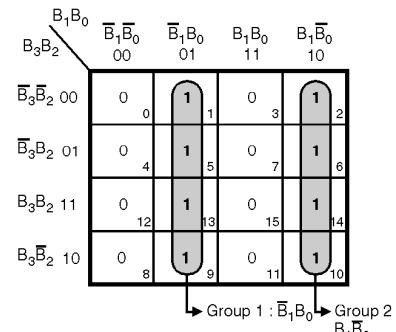
∴ Simplified equation : G₂ = B₃B₂ + B₃B₂ = B₃ ⊕ B₂

For output G₁

(C-375) Fig. 5.2.5(c) : K-map for G₁

∴ Simplified equation : G₁ = B₂B₁ + B₂B₁ = B₂ ⊕ B₁

For output G₀

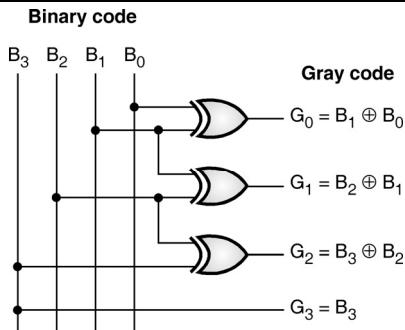
(C-375) Fig. 5.2.5(d) : K-map for G₀

∴ Simplified equation :

$$G_0 = B_1 B_0 + B_1 B_0 = B_1 \oplus B_0$$

Step 3 : Realize Binary to gray code converter using gates :

- Binary to gray code converter is shown in Fig. 5.2.5(e).



(C-376) Fig. 5.2.5(e) : Binary to gray code converter

5.2.4 Gray to BCD Converter :

Ex. 5.2.1 : Convert 4-bit gray code into corresponding BCD code. Show truth table and MSI circuit.

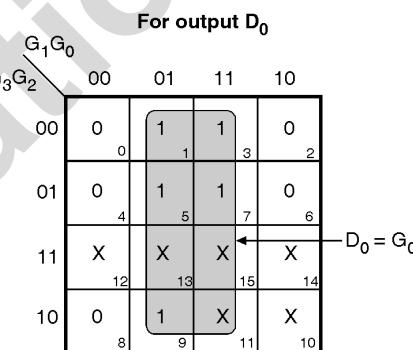
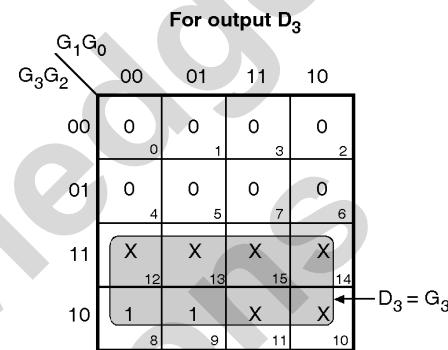
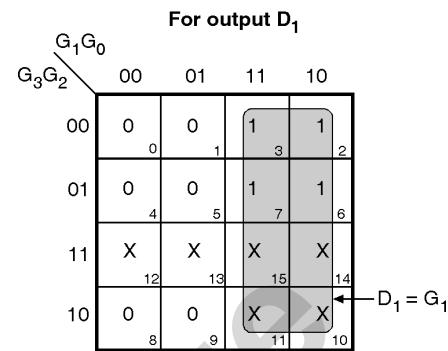
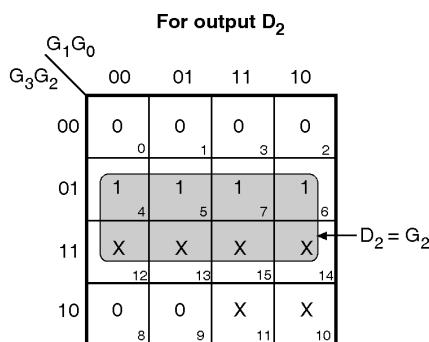
May 10, 6 Marks

Soln. :

Step 1 : Write the truth table relating gray and BCD codes : (C-8279)

Decimal	Gray inputs				BCD outputs			
	G ₃	G ₂	G ₁	G ₀	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	1
2	0	0	1	1	1	0	0	1
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	0	1	0
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1

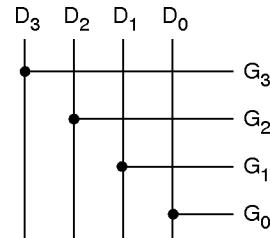
Step 2 : Write K-maps for each output and get simplified equation :



(C-1709) Fig. P. 5.2.1

Step 3 : Realization :

- The gray to BCD converter is shown in Fig. P. 5.2.1(a).



(C-1710) Fig. P. 5.2.1(a)

Ex. 5.2.2 : Design and explain in detail 4-bit gray code to 5-bit BCD code conversion. For this design use K-map reduction and MSI circuit using basic gates. **May 11, 16 Marks**



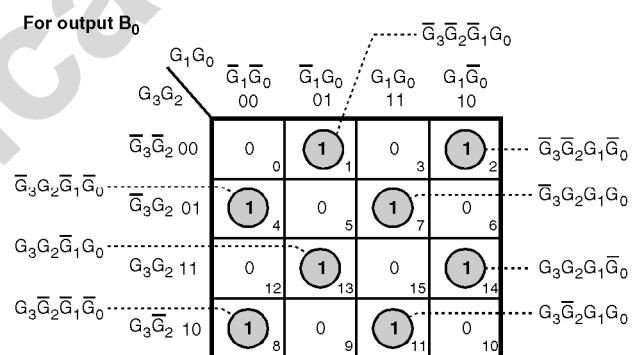
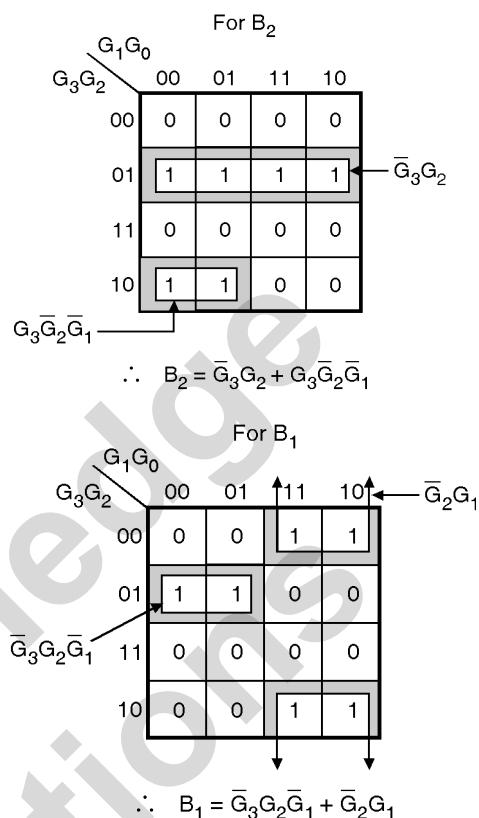
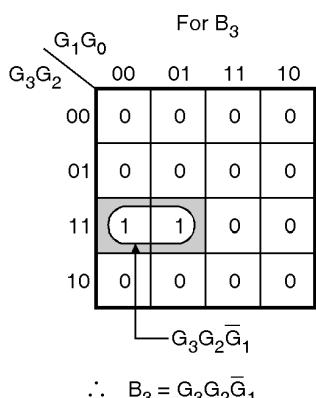
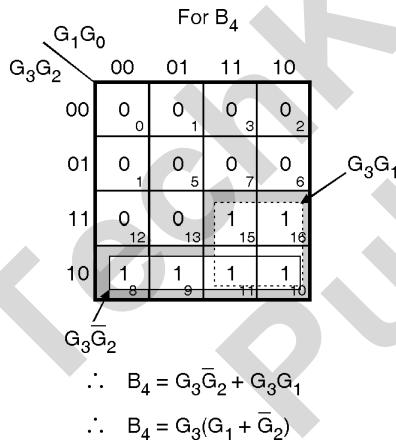
Soln. :

Step 1 : Write the truth table 4 bit gray to 5 bit BCD code converter :

(C-8280) Table P. 5.2.2

Decimal	Gray inputs				BCD outputs				
	G ₃	G ₂	G ₁	G ₀	B ₄	B ₃	B ₂	B ₁	B ₀
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	1	0	0	0	1	0
3	0	0	1	0	0	0	0	1	1
4	0	1	1	0	0	0	1	0	0
5	0	1	1	1	0	0	1	0	1
6	0	1	0	1	0	0	1	1	0
7	0	1	0	0	0	0	1	1	1
8	1	1	0	0	0	0	1	0	0
9	1	1	0	1	0	1	0	0	1
10	1	1	1	1	1	0	0	0	0
11	1	1	1	0	1	0	0	0	1
12	1	0	1	0	1	0	0	1	0
13	1	0	1	1	1	0	0	1	1
14	1	0	0	1	1	0	1	0	0
15	1	0	0	0	1	0	1	0	1

Step 2 : Write the K-maps and simplify :



(C-1964) Fig. P. 5.2.2 : K-maps

Simplified expression for B₀ :

$$\begin{aligned}
 B_0 &= \overline{G_3} \overline{G_2} G_1 G_0 + \overline{G_3} G_2 G_1 G_0 + \overline{G_3} G_2 G_1 G_0 \\
 &\quad + \overline{G_3} G_2 G_1 G_0 + \overline{G_3} G_2 G_1 G_0 + \overline{G_3} G_2 G_1 G_0 \\
 &\quad + \overline{G_3} G_2 G_1 G_0 + \overline{G_3} G_2 G_1 G_0 \\
 \therefore B_0 &= \overline{G_1} \overline{G_0} (\overline{G_3} G_2 + G_3 \overline{G}_2) + \overline{G_1} G_0 (\overline{G_3} \overline{G}_2 + G_3 G_2) \\
 &\quad \text{EX-OR} \qquad \qquad \qquad \text{EX-NOR} \\
 &\quad + G_1 G_0 (\overline{G_3} G_2 + G_3 \overline{G}_2) + G_1 \overline{G}_0 (\overline{G_3} \overline{G}_2 + G_3 G_2) \\
 &\quad \text{EX-OR} \qquad \qquad \qquad \text{EX-NOR}
 \end{aligned}$$

(C-6178)



$$\therefore B_0 = \overline{G_1} \overline{G_0} (G_3 \oplus G_2) + \overline{G_1} G_0 (\overline{G_3} \oplus \overline{G_2}) \\ + G_1 G_0 (G_3 \oplus G_2) + G_1 \overline{G_0} (\overline{G_3} \oplus \overline{G_2})$$

$$\therefore B_0 = (G_3 \oplus G_2) (\overline{G_1} \overline{G_0} + G_1 G_0) \\ + (\overline{G_3} \oplus \overline{G_2}) (\overline{G_1} G_0 + G_1 \overline{G_0})$$

$$\therefore B_0 = (G_3 \oplus G_2) (\overline{G_1} \oplus \overline{G_0}) \\ + (\overline{G_3} \oplus \overline{G_2}) (G_1 \oplus G_0)$$

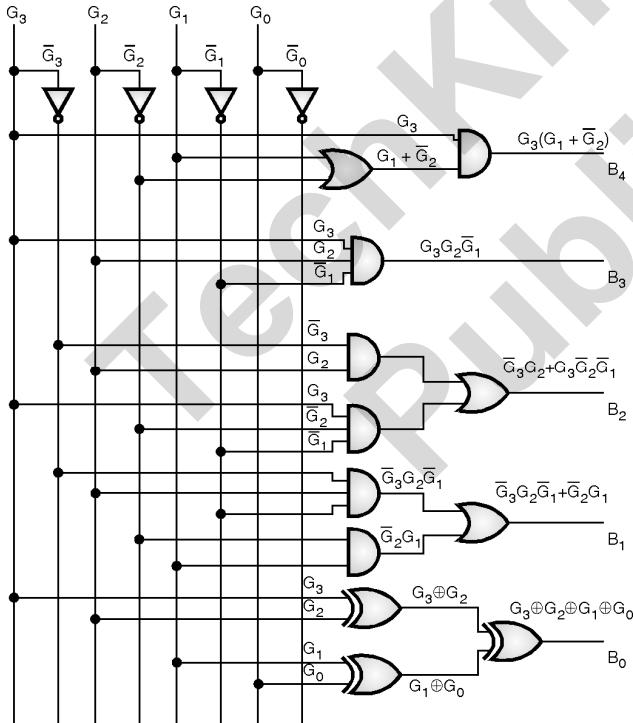
$$\text{Let } X = G_3 \oplus G_2$$

$$\text{And } Y = (G_1 \oplus G_0)$$

$$\therefore B_0 = \overline{XY} + \overline{X}Y = X \oplus Y$$

\therefore Substituting for X and Y we get,

$$B_0 = (G_3 \oplus G_2) \oplus (G_1 \oplus G_0) = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$



(C-1965) Fig. P. 5.2.2(a) : Gray to 5 bit BCD code converter

5.2.5 Excess 3 to BCD Converter :

Ex. 5.2.3 : Design a 3-bit excess 3 to 3-bit BCD code converter using logic gate. **May 15, 6 Marks**

Soln. :

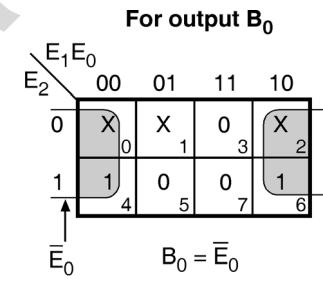
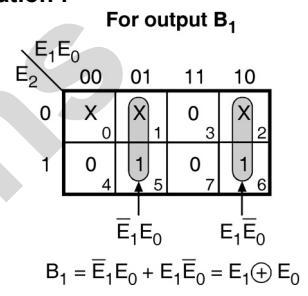
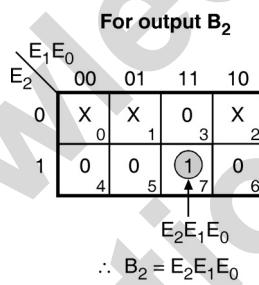
Step 1 : Write the truth table :

Input Excess 3			Output BCD		
E ₂	E ₁	E ₀	B ₂	B ₁	B ₀
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	0	0
0	0	0	X	X	X
0	0	1	X	X	X
0	1	0	X	X	X

↑
Valid states
↓
Invalid states

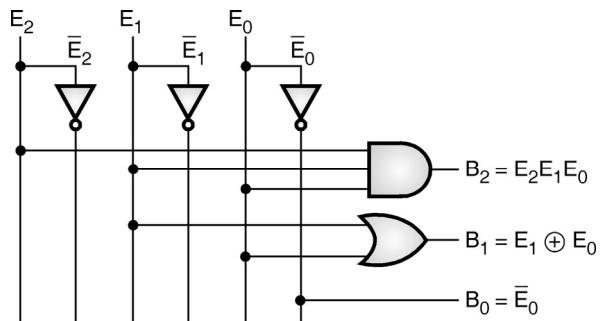
(C-6179)

Step 2 : K-map and simplification :



(C-5103) Fig. P. 5.2.3(a)

Step 3 : Logic gate diagram :



(C-5104) Fig. P. 5.2.3(b)

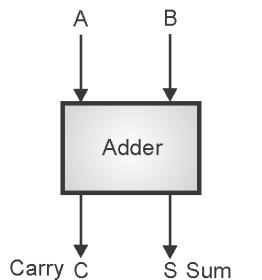
5.3 Binary Adders and Subtractors :

- Addition of two binary digits is most basic operation performed by the digital computers.

- The rules of binary addition are summarised in Table 5.3.1(A). A and B are the single bit digits to be added whereas C and S are the carry and sum outputs respectively.

Table 5.3.1(A) : Rules of binary addition

A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

**(C-2588) Fig. 5.3.1(A) : Block diagram of adder**

5.3.1 Types of Binary Adders :

- In this section we are going to learn digital circuits which are used to "add" two "binary" numbers.
- The binary adders are of two types :
 - Half adder and
 - Full adder

5.3.2 Half Adder :

SPPU : May 07

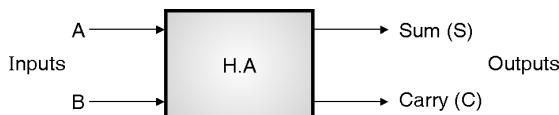
University Questions

Q. 1 What do you mean by half adder ?

(May 07, 2 Marks)

Definition and block diagram :

- Half adder is a combinational logic circuit with two inputs and two outputs, which carries out addition of two "single" bit numbers.
- This circuit has two outputs namely "carry" and "sum". The block diagram of half adder is as shown in Fig. 5.3.1(a).

**(C-344) Fig. 5.3.1(a) : Block diagram**

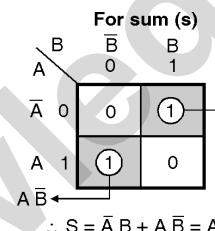
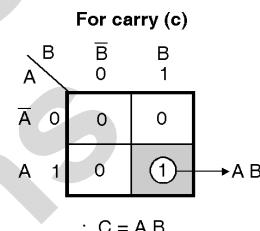
- The half adder circuit is supposed to add two single bit binary numbers A and B.
- Therefore the truth table of a half adder is as shown in Fig. 5.3.1(b).

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(C-344) Fig. 5.3.1(b) : Truth table

Design using K-maps :

- K-maps for carry and sum outputs are as shown in Figs. 5.3.2(a) and (b).

**(a) K – map for sum output****(b) K – map for carry output****(C-345) Fig. 5.3.2**

- Boolean expressions for the sum (S) and carry (C) output are obtained from the K-maps as follows :

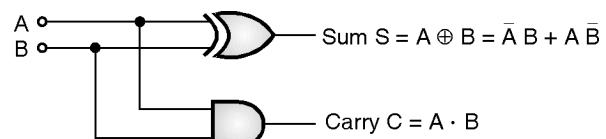
$$\begin{aligned} S &= \bar{A}\bar{B} + \bar{A}B = A \oplus B \\ C &= AB \end{aligned} \quad \left. \right\} \quad \dots(5.3.1)$$

(C-346)

- The disadvantage of half adder is that addition of three bits is not possible to perform.

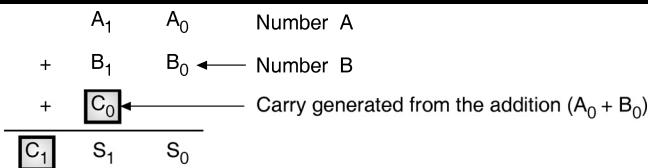
Logic diagram using gates :

- The half adder circuit is as shown in Fig. 5.3.3.

**(C-346) Fig. 5.3.3 : Half adder circuit**

Disadvantage of half adder :

- The principle of adding two 2-bit numbers A and B is as shown in Fig. 5.3.4(a).
- Let Number A = $A_1 A_0$
And Number B = $B_1 B_0$
- Then the addition should take place as shown in Fig. 5.3.4(a).



(C-347) Fig. 5.3.4(a) : Two bit binary addition

- A half adder can add A₀ and B₀ to produce S₀ and C₀. But the addition of next bits requires the addition of A₁, B₁ and C₀.
- The addition of three bits is not possible to perform by using a half adder. Hence we cannot use a half adder in practice.

5.3.3 Full Adder :

SPPU : May 07

University Questions

Q. 1 What do you mean by full adder ?

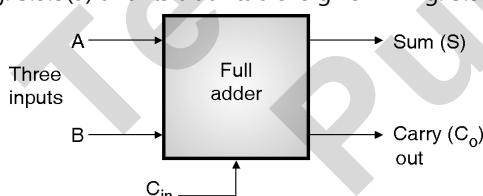
(May 07, 6 Marks)

Definition :

- Full adder is a three input two output combinational logic circuit which can add three single bits applied at its input to produce Sum and Carry outputs.
- It can add two one-bit numbers A and B, and carry C_{in}. The full adder is a three input and two output combinational circuit.
- To overcome the drawback of Half Adder circuit, a 3 single bit adder circuit called Full Adder is developed.
- It can add two one-bit numbers A and B, and carry C_{in}.

Block diagram and Truth Table :

- The block diagram of a full adder is as shown in Fig. 5.3.5(a) and its truth table is given in Fig. 5.3.5(b).



(C-350) Fig. 5.3.5(a) : Block diagram

Inputs			Outputs	
A	B	C _{in}	S	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

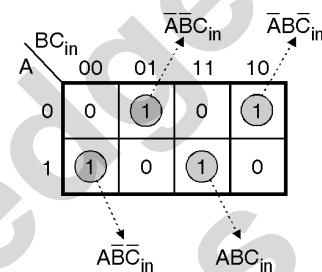
(b) Truth table

(C-350) Fig. 5.3.5 : Full adder

The K-maps :

- The K-maps for the sum (S) and carry out (C_o) outputs and the corresponding Boolean expressions are as shown in Fig. 5.3.6.
- Note that the K-maps have been written from the truth table of Fig. 5.3.5(b).

For the sum output :



(C-351) Fig. 5.3.6(a) : K-map for sum output

Expression for sum output :

$$S = \overline{ABC}_{in} + \overline{ABC}_{in} + ABC_{in} + \overline{ABC}_{in}$$

$$S = C_{in} \underbrace{(AB + AB)}_{\text{EX-NOR}} + \overline{C_{in}} \underbrace{(\overline{AB} + \overline{AB})}_{\text{EX-OR}}$$

(C-6375)

$$\therefore S = C_{in} (\overline{AB} + AB) + \overline{C_{in}} (AB + \overline{AB})$$

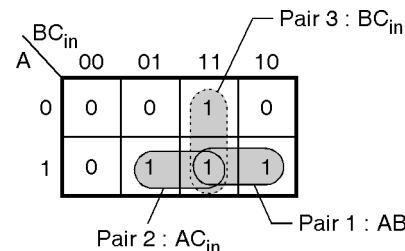
$$\text{Let } X = \overline{AB} + AB$$

$$\therefore S = C_{in} \overline{X} + \overline{C_{in}} X = C_{in} \oplus X = C_{in} \oplus (AB + \overline{AB})$$

$$\text{But } AB + \overline{AB} = A \oplus B$$

$$\therefore S = C_{in} \oplus A \oplus B \quad \dots(5.3.2)$$

For carry output :

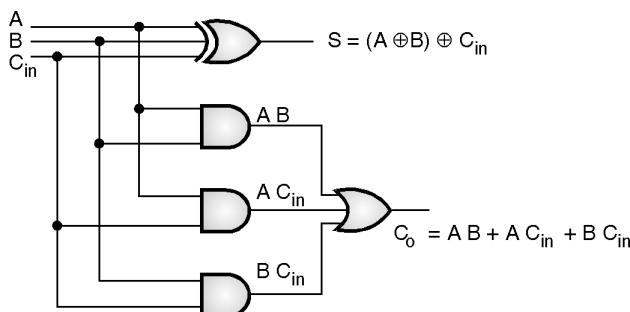


(C-352) Fig. 5.3.6(b) : K-map for carry output

Expression for carry output :

$$C_o = AB + AC_{in} + BC_{in} \quad \dots(5.3.3)$$

- We can use Equations (5.3.4) and (5.3.5) to draw the logic diagram of a full adder as shown in Fig. 5.3.6(c).

**Logic diagram for full adder :**

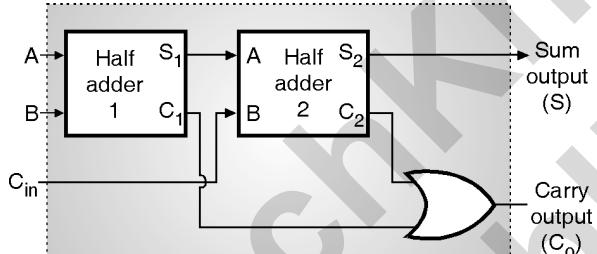
(C-353) Fig. 5.3.6(c) : Full adder circuit

5.3.4 Full Adder using Half Adder :**SPPU : May 07****University Questions**

- Q. 1** How will you implement full adder using half adder ? Explain with circuit diagram.

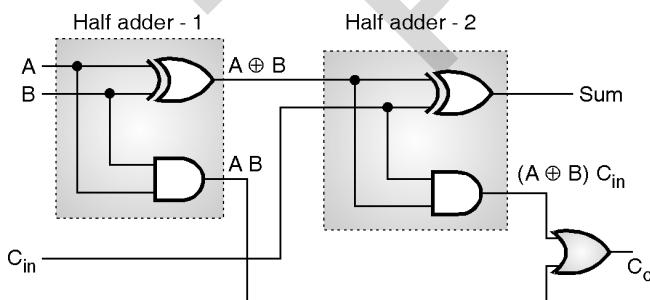
(May 07, 6 Marks)

- The full adder circuit can be constructed using two half adders as shown in Fig. 5.3.7 and the detail circuit is shown in Fig. 5.3.8.



(C-354) Fig. 5.3.7 : Full adder using half adders

- A full adder can be implemented using two half adders and an OR gate as shown in Fig. 5.3.8.



(C-355) Fig. 5.3.8 : Full adder using two half adders

- Now let us prove that this circuit acts as a full adder.

Proof :

- Refer Fig. 5.3.8 and write the expression for sum output as,

$$S = (A \oplus B) \oplus C_{in} = A \oplus B \oplus C_{in}$$

- This expression is same as that obtained for the full adder. Thus the sum output has been successfully implemented by the circuit shown in Fig. 5.3.8.
- Now write the expression for carry output C_o as,

$$C_o = (A \oplus B) C_{in} + AB$$

$$C_o = (\overline{AB} + AB) C_{in} + AB = \overline{ABC}_{in} + ABC_{in} + AB$$

$$= \overline{ABC}_{in} + ABC_{in} + AB(1 + C_{in})$$

$$= \overline{ABC}_{in} + ABC_{in} + AB + ABC_{in}$$

$$= BC_{in}(A + A) + ABC_{in} + AB$$

$$= BC_{in} + ABC_{in} + AB$$

$$= BC_{in} + ABC_{in} + AB(1 + C_{in})$$

$$= BC_{in} + ABC_{in} + AB + ABC_{in}$$

$$= BC_{in} + AB + AC_{in}(B + B)$$

$$\therefore C_o = BC_{in} + AB + AC_{in}$$

...Proved.

- This expression is same as that for a full adder. Thus we have proved that circuit shown in Fig. 5.3.8 really behaves like a full adder.

5.3.5 Applications of Full Adder :

- The full adder acts as the basic building block of the 4 bit/8 bit binary/BCD adder ICs such as 7483.

5.3.6 Binary Subtractors :

- The rules of binary subtraction are as follows :

$$0 - 0 = 0 \quad 0 - 1 = 1 \text{ with borrow 1}$$

$$1 - 0 = 1 \quad 1 - 1 = 0$$

- Note that in the second case ($0 - 1$) it is necessary to borrow a 1.

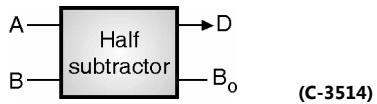
Types of binary subtractors :

- The types of binary subtractors are :

1. Half subtractor, 2. Full subtractor.

5.3.7 Half Subtractor :**Definition :**

- Half subtractor is a combinational circuit with two inputs and two outputs (difference and borrow).
- It produces the difference between the two binary bits at the input and also produces an output (borrow) to indicate if a 1 has been borrowed.



- In the subtraction $(A - B)$, A is called as **minuend bit** and B is called as **subtrahend bit**.

Truth table :

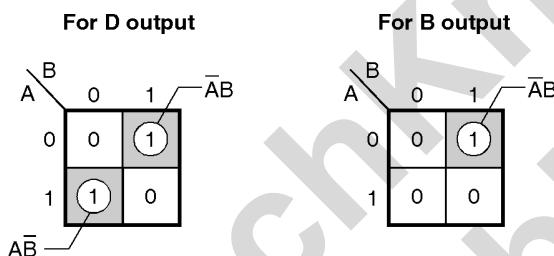
- Truth table showing the outputs of a half subtractor for all the possible combinations of input are shown in Table 5.3.1.

(C-8058) Table 5.3.1 : Truth table for half subtractor

Inputs		Outputs	
A	B	Difference D (A-B)	Borrow B _o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-maps for difference and borrow outputs :

- The K-maps for the two outputs of a half subtractor are as shown in Fig. 5.3.9.



(a) K-map and simplification for difference output

(b) K-map and simplification for the borrow output

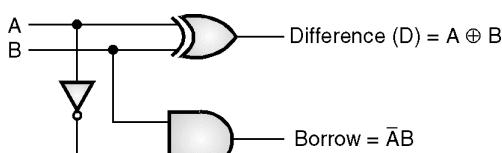
(C-356) Fig. 5.3.9

$$\therefore \text{Difference } D = \overline{AB} + AB = A \oplus B$$

$$\therefore \text{Borrow } B_o = \overline{AB}$$

Logic diagram :

- The logic diagram using these two Boolean expressions is as shown in Fig. 5.3.9(c).



(C-357) Fig. 5.3.9(c) : Half subtractor circuit

Disadvantage of the half subtractor :

- Half subtractor can only perform the subtraction of two binary bits.

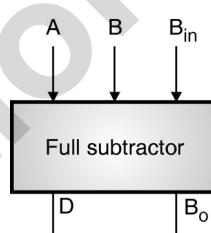
- But while performing the subtraction, it does not take into account the borrow of the lower significant stage.

5.3.8 Full Subtractor :

Definition and block diagram :

- The full subtractor is a combinational circuit with three inputs A, B and B_{in} and two outputs D and B_o.
- A is the minuend, B is subtrahend, B_{in} is the borrow produced by the previous stage, D is the difference output and B_o is the borrow output.
- The disadvantage of a half subtractor is overcome if we use the full subtractor.
- Fig. 5.3.10 shows the symbol of a full subtractor.

Inputs



(C-2591) Fig. 5.3.10 : Symbol of a full subtractor

Truth table :

- The truth table for full subtractor is shown in Table 5.3.2.

(C-8059) Table 5.3.2 : Truth table for a full subtractor

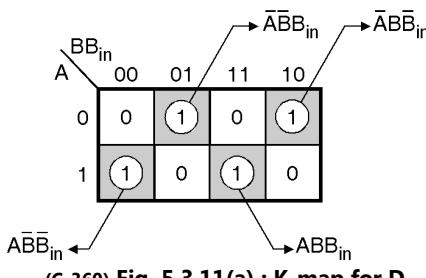
Inputs			Outputs	
A (Minuend)	B (Subtrahend)	B _{in} Previous borrow	(A-B-B _{in})	B _o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-maps and simplifications :

- K-maps for D and B_o outputs are shown in Figs. 5.3.11(a) and (b).



For difference output :



$$\therefore D = \bar{A}B\bar{B}_{in} + ABB_{in} + \bar{A}B\bar{B}_{in} + AB\bar{B}_{in}$$

Simplification for difference output :

- From Fig. 5.3.11(a),

$$\begin{aligned} D &= \bar{A}\bar{B}B_{in} + \bar{A}\bar{B}\bar{B}_{in} + ABB_{in} + AB\bar{B}_{in} \\ &= B_{in}(\bar{A}\bar{B} + AB) + \bar{B}_{in}(\bar{A}B + AB) \\ &\quad \text{EX-NOR} \qquad \text{EX-OR} \end{aligned} \quad (\text{C-6376})$$

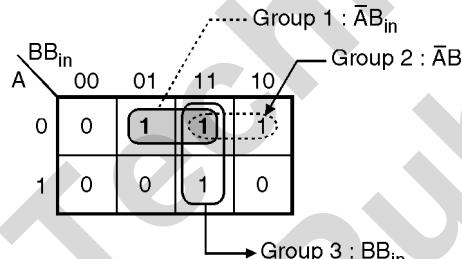
$$\therefore D = B_{in}(\overline{A \oplus B}) + \bar{B}_{in}(A \oplus B)$$

$$\text{Let } A \oplus B = C,$$

$$\therefore D = B_{in}C + \bar{B}_{in}C = B_{in} \oplus C$$

$$\therefore D = B_{in} \oplus A \oplus B \quad \dots(5.3.4)$$

For borrow output :



$$\therefore B_o = \bar{A}B_{in} + AB + BB_{in}$$

Simplification for borrow output :

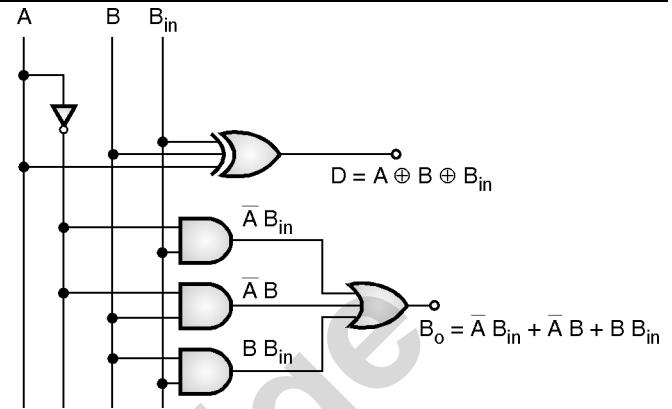
- From Fig. 5.3.13(b),

$$B_o = \bar{A}B_{in} + \bar{A}B + BB_{in} \quad \dots(5.3.5)$$

- No further simplification is possible.

Logic diagram for full subtractor :

- Logic diagram for the full subtractor is shown in Fig. 5.3.12. This has been drawn by using the Boolean equations of (5.3.8) and (5.3.9).



(C-361) Fig. 5.3.12 : Logic diagram for a full subtractor

5.3.9 Full Subtractor using Half Subtractors :

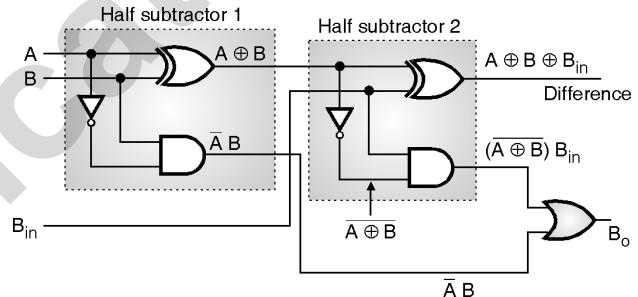
SPPU : Dec. 08

University Questions

Q. 1 With the help of a circuit diagram explain the full subtractor using half subtractor. (Dec. 08, 8 Marks)

Logic diagram :

- Fig. 5.3.13 shows the implementation of a full subtractor using two half subtractors and an OR gate.



(C-362) Fig. 5.3.13 : Full subtractor using half subtractors

- Let us prove that the circuit shown in Fig. 5.3.15 really operates as a full subtractor.

Proof :

- Refer Fig. 5.3.13 to write the expression for difference output D as,

$$D = (A \oplus B) \oplus B_{in} = A \oplus B \oplus B_{in}$$

- This is same as the expression for D output of a full subtractor.

- Now write the expression for Borrow output B_{out}:

$$\begin{aligned} B_{out} &= (\overline{A \oplus B})B_{in} + A\bar{B} = (\overline{AB} + \overline{AB})B_{in} + A\bar{B} \\ &= (\overline{AB} + AB)B_{in} + A\bar{B} = A\bar{B}B_{in} + ABB_{in} + A\bar{B} \\ &= A\bar{B}B_{in} + ABB_{in} + A\bar{B}(1 + B_{in}) \dots \text{since } (1 + B_{in}) = 1 \end{aligned}$$

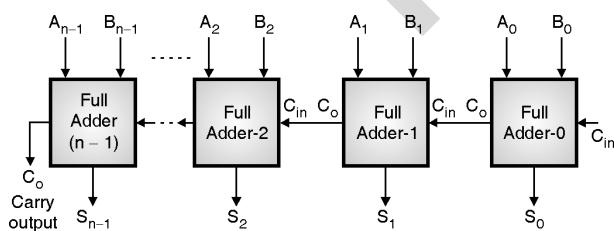


$$\begin{aligned}
 &= A B B_{in} + A B B_{in} + A B + A B B_{in} \\
 &= A B B_{in} + B B_{in} (A + A) + A B \\
 &= A B B_{in} + B B_{in} + A B \quad \dots\text{since } (A + A) = 1 \\
 &= A B B_{in} + B B_{in} + A B (1 + B_{in}) \\
 &= A B B_{in} + B B_{in} + A B + A B B_{in} \\
 &= A B_{in} (B + B) + B B_{in} + A B \\
 &= A B_{in} + B B_{in} + A B \quad \dots\text{Proved.}
 \end{aligned}$$

- Note that this expression is exactly same as that for B_0 of the full subtractor.
- Thus the circuit shown in Fig. 5.3.13 acts as a full subtractor.

5.4 The n-Bit Parallel Adder :

- The full adder is capable of adding only two single digit binary numbers along with a carry input.
- But in practice we need to add binary numbers which are much larger in size than just one bit.
- The two binary numbers to be added could be 4 bit, 8 bit, 16 bit long.
- In general we assume that both the numbers are n bit long.
- To add two n -bit binary numbers we need to use the n -bit parallel adder shown in Fig. 5.4.1.
- It uses a number of full adders which are connected in cascade.
- The carry output of the previous full adder is connected to the carry input of the next full adder as shown in Fig. 5.4.1.



(C-385) Fig. 5.4.1 : Block diagram of n-bit parallel adder

5.4.1 A Four Bit Parallel Adder Using Full Adders :

SPPU : May 06, Dec. 06

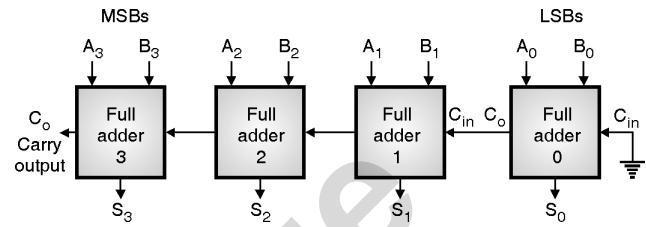
University Questions

Q. 1 Draw and explain 4-bit full adder.

(May 06, Dec. 06, 4 Marks)

Block diagram :

- The block diagram of a four bit parallel adder using full adders is shown in Fig. 5.4.2.

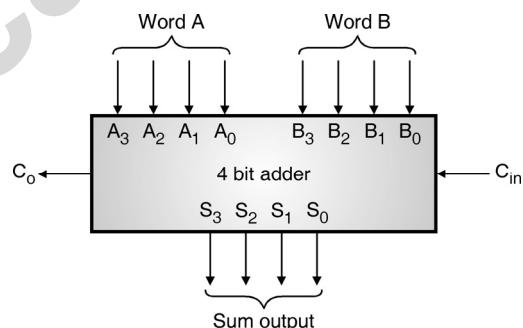


(C-386) Fig. 5.4.2 : Block diagram of a four-bit parallel adder

- Let the two four bit words that are to be added be A and B and be denoted as follows :

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$
- A_0 and B_0 represent the LSBs of the four bit words A and B . Hence full adder-0 is the lowest stage. Hence its C_{in} has been connected to 0 permanently.
- The rest of connections are exactly same as those done for the n -bit parallel adder.
- The four-bit parallel adder is a very common logic circuit. It is normally shown by a block diagram as shown in Fig. 5.4.3.



(C-387) Fig. 5.4.3 : Block diagram of 4-bit parallel adder

- It has two 4 bit inputs A_3, \dots, A_0 and B_3, \dots, B_0 , a carry input and carry output and 4-bit sum output S_3, S_2, S_1, S_0 .

5.4.2 Propagation Delay in Parallel Adder :

- In parallel adder carry out of the previous stage is connected to carry in of the next stage.
- Therefore the carry is said to be propagated like ripple from the LSB stage to the MSB stage. This phenomenon is called **ripple carry propagation**.
- Due to this ripple carry propagation **time delay** is introduced in the addition process. This time delay is called as the propagation delay.



- Consider the addition of the two 4-bit numbers.

$$\begin{array}{r}
 0110 \\
 +0011 \\
 \hline
 \text{Carry } 11 \\
 \hline
 1001 \quad (\text{C-6475})
 \end{array}$$

- From this addition it is evident that the addition of 1's in the second position produces a carry which is added to the bits in the third position and the carry produced in the third position is added to the bits in the fourth position.
- Therefore the sum bit produced in the MSB position of the result depends on the carry generated due to additions in the preceding stages.
- The real problem is created due to this.
- If the propagation delay of each full adder is say 20 nS, then sum S_3 will reach its correct value after $3 \times 20 = 60$ nS from the instant when LSB carry is generated.
- Hence the total time required to perform the addition will be $4 \times 20 = 80$ nS.
- The problem of propagation delay becomes severe as the number of bits increases.
- For example if $n = 16$ then the total time required for performing the addition is $16 \times 20 = 320$ nS.

5.4.3 Look Ahead – Carry Adder :

SPPU : May 06, Dec. 07, May 14

University Questions

Q. 1 Draw and explain 4-bit full adder. How will you generate look ahead carry for your circuit.

(May 06, 6 Marks)

Q. 2 What do you mean by carry generate and carry propagate ? Explain with suitable equations and block diagram. Write an equation for C_3 using carry generate and carry propagate. Simplify this equation to get minimum gate delay. How many gate delays are needed to generate C_3 with this principle ? Explain your calculations.

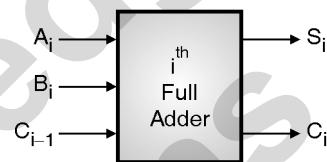
(Dec. 07, 10 Marks)

Q. 3 Draw and explain the look ahead carry generator.

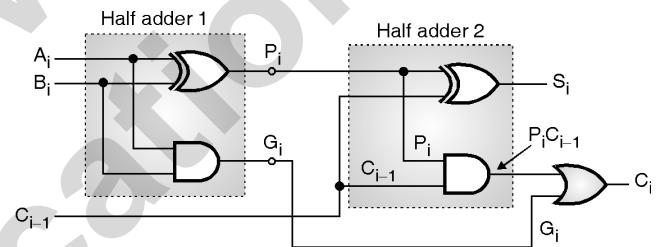
(May 14, 6 Marks)

Block diagram :

- The problem of propagation delay can be eliminated by using this addition technique.
- The look-ahead-carry addition will therefore speed up the addition process.
- The adder with look ahead carry needs additional hardware but the speed of that adder is independent of the number of bits.
- Consider the block diagram of full adder Fig. 5.4.4(a) and its AND-OR-EXOR realization shown in Fig. 5.4.4(b).



(a) Block diagram of i^{th} full adder



(b) Realization using AND-OR-EXOR gates

(C-389) Fig. 5.4.4

- Refer Fig. 5.4.4(b) to write,

$$P_i = A_i \oplus B_i \quad \dots(5.4.1)$$

$$\text{and } G_i = A_i B_i \quad \dots(5.4.2)$$

$$\text{Also } S_i = P_i \oplus C_{i-1} = A_i \oplus B_i \oplus C_{i-1} \quad \dots(5.4.3)$$

$$\text{and } C_i = G_i + P_i C_{i-1} \quad \dots(5.4.4)$$

- The carry output G_i of the first half adder is equal to 1 if $A_i = B_i = 1$ and a carry is generated at the i^{th} stage of the parallel adder. That means $C_i = 1$.
- This variable G_i is known as **Carry Generate** and its value does not depend on the input carry i.e. C_{i-1} .
- The variable P_i is called as **Carry Propagate** because this term is associated with the propagation of carry from C_{i-1} to C_i . Now consider Equation (5.4.4) i.e. $C_i = G_i + P_i C_{i-1}$.
- Using this equation, we can write the expression for the carry output of each stage in a 4 bit parallel adder as follows :



$$P_i = A_i \oplus B_i$$

Stage expression for carry output

$$0 \quad C_0 = G_0 + P_0 C_{-1} \quad \dots(5.4.5)$$

$$\begin{aligned} 1 \quad C_1 &= G_1 + P_1 C_0 \\ &= G_1 + P_1 (G_0 + P_0 C_{-1}) \\ \therefore C_1 &= G_1 + P_1 G_0 + P_0 P_1 C_{-1} \end{aligned} \quad \dots(5.4.6)$$

Stage expression for carry output

$$2 \quad C_2 = G_2 + P_2 C_1 = G_2 + P_2 (G_1 + P_1 G_0 + P_0 P_1 C_{-1})$$

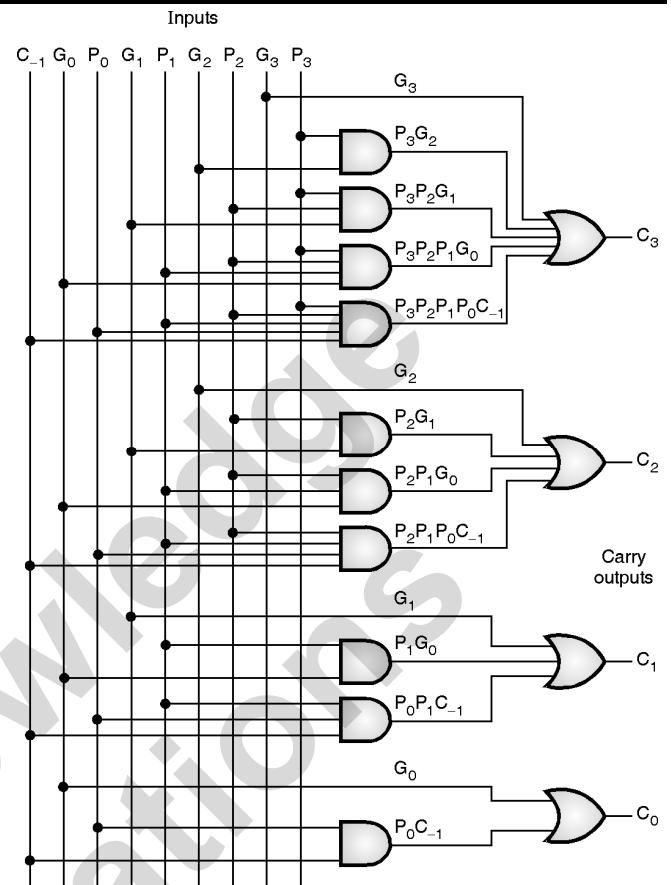
$$\therefore C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1} \quad \dots(5.4.7)$$

$$\begin{aligned} 3 \quad C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \\ &\quad + P_3 P_2 P_1 P_0 C_{-1} \end{aligned} \quad \dots(5.4.8)$$

- In the expressions stated above, the variable involved are, $G_0, G_1, G_2, G_3, P_0, P_1, P_2, P_3$ and C_{-1} .
- Out of them the G variables are generated from the A and B inputs using AND gates (as illustrated in Equation (5.4.2)).
- And the P variables are obtained again directly from A and B inputs using EX-OR gates (as illustrated in Equation (5.4.1)).
- If the G, P and C_{-1} are at a time available, then it is possible to produce the carry outputs C_0, C_1, C_2 and C_3 by using 2-level realization (AND-OR or NAND-NAND) etc.
- The advantage of generating the carry outputs using this method is that the propagation delay in this process corresponds to the propagation delay of only two gates.
- These carry outputs are then connected to the carry inputs of the succeeding stages.
- This eliminates the problem of carry getting propagated like ripples.

Logic diagram :

- The logic circuit of a look ahead carry generator is shown in Fig. 5.4.5 and it is based on Equations (5.4.1) through (5.4.8).

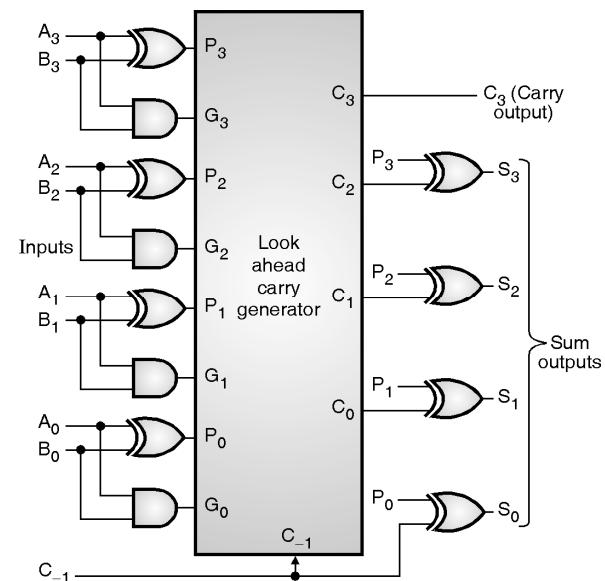


(C-390) Fig. 5.4.5 : Logic diagram of the look ahead carry generator

5.4.4 Four Bit Fast Adder with Look-Ahead Carry :

Block diagram :

- The block diagram of a four bit parallel adder using the look-ahead carry generator is shown in Fig. 5.4.6.



(C-391) Fig. 5.4.6 : A 4-bit parallel adder



- The P_i and G_i variables (i.e. P_0, P_1, \dots, P_3 and G_0, G_1, \dots, G_3) are obtained from the inputs A_i and B_i (i.e. $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$) by using the EX-OR and AND gates respectively.
- The carry outputs C_0 through C_3 are produced simultaneously by the look ahead carry generator, as explained earlier.
- These carry outputs and P_i variables are EX-ORED to produce the sum outputs S_0 through S_3 .
- For example S_3 is produced by Ex-ORing P_3 and C_2 , then S_2 is produced by Ex-ORing P_2 and C_1 and so on.

5.4.5 MSI Binary Adder IC 74 LS 83 / 74 LS 283 :

SPPU : Dec. 10, Dec. 11

University Questions

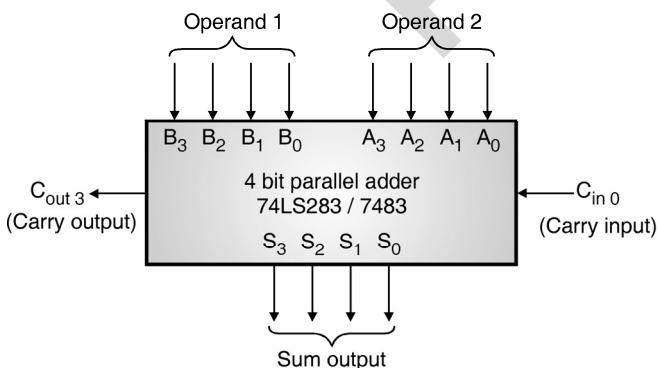
- Q. 1** Explain for IC 74LSXX various characteristics in brief. (Dec. 10, 4 Marks)
- Q. 2** What is the use of 7483 chip ? (Dec. 11, 4 Marks)

Block diagram :

- The most common binary parallel adder in the integrated circuit form is IC 74 LS 83 / 74 LS 283. This is an MSI (Medium Scale Integration) IC.
- It is a 4-Bit parallel adder, which consists of four interconnected full adders alongwith the look-ahead carry circuit.
- The IC 7483 and 74283 are TTL MSI (Medium Scale Integration) for 4-bit parallel adders and both of them have the same pin configuration.

Functional symbol :

- Fig. 5.4.7 shows the functional symbol of IC 74 LS 283. $A_3 A_2 A_1 A_0$ is a four bit word A and $B_3 B_2 B_1 B_0$ is another word B.



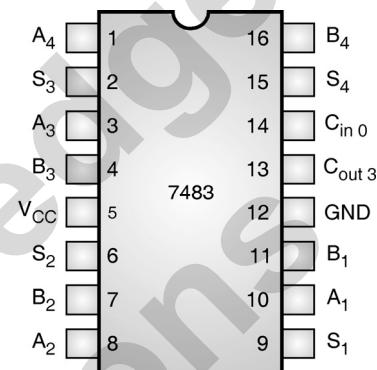
(C-392) Fig. 5.4.7 : Functional symbol for 74 LS 283

- Both these words are applied at the inputs of the I.C. 7483/74 LS 283.

- $C_{in} 0$ is the input carry and $C_{out} 3$ represents the output carry. S_3, S_2, S_1, S_0 represent the sum outputs with S_3 MSB.

Pin diagram :

- The pin diagram of IC 7483 is shown in Fig. 5.4.8.
- This IC adds the two four bit words A and B and the bit at $C_{in} 0$ and produces a four bit sum output along with carry output at $C_{out} 3$.

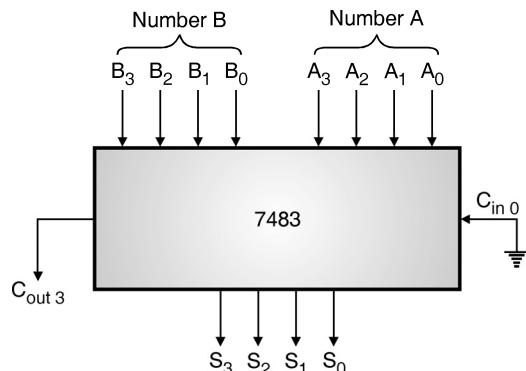


(C-392) Fig. 5.4.8 : Pin diagram of IC – 7483

5.4.6 Four Bit Binary Adder using IC 7483 :

Block diagram :

- A four bit binary adder is shown in Fig. 5.4.9. Note that the carry input $C_{in} 0$ has been connected to ground.
- So at the outputs we get the addition of two four bit numbers A and B.



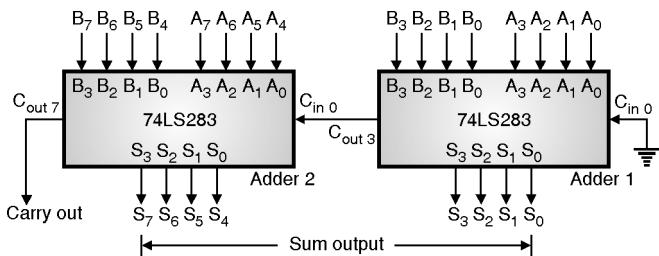
(C-393) Fig. 5.4.9 : 4 bit binary adder using IC 7483

5.4.7 Cascading of Adders :

- If we want to add two 8-bit numbers using the 4-bit parallel adder 74283, then we have to cascade two such four bit adders.
- Fig. 5.4.10 shows an 8-bit adder using two 4-bit adders. Similarly it is possible to connect a number of adders to make an n-bit adder.



- $A_7 - A_0$ and $B_7 - B_0$ are the two eight bit numbers to be added.
- Adder-1 in Fig. 5.4.10 adds the four LSB bits of the two numbers i.e. $A_3 - A_0$ and $B_3 - B_0$.
- The carry input of first adder has been connected to ground (logic 0).



(C-394) Fig. 5.4.10 : 8-bit addition using 4-bit adders

- The $C_{out\ 3}$ i.e. the carry output of adder-1 is connected to $C_{in\ 0}$ input of adder-2. The second adder adds this carry and the four MSB bits of the two numbers.
- $C_{out\ 7}$ of adder-2 acts as the final output carry and the sum output is from S_7 through S_0 .

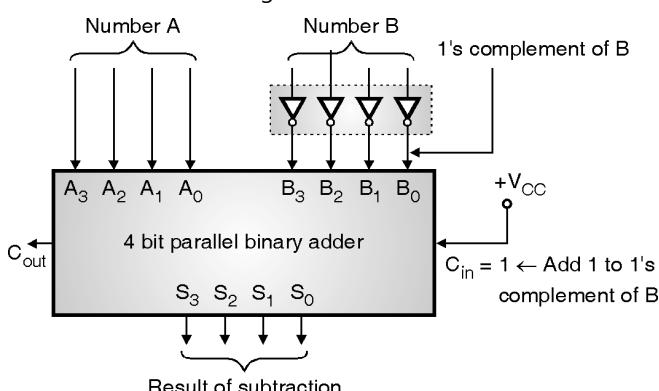
5.5 n-bit Parallel Subtractor :

- The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted.
- For example we can perform the subtraction $(A - B)$ by adding either 1's complement or 2's complement of B to A . That means we can use a binary adder to perform the binary subtraction.

5.5.1 4 Bit Parallel Subtractor using IC7483 :

Block diagram :

- A 4-bit parallel subtractor using a 4-bit parallel adder IC 7483 is shown in Fig. 5.5.1.



(C-395) Fig. 5.5.1 : 4-bit parallel binary subtractor using 2's complement

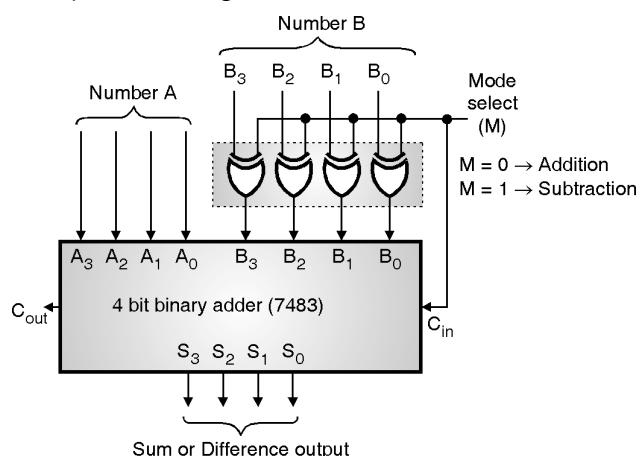
Description :

- The number to be subtracted (B) is first passed through inverters to obtain its 1's complement.
- One inverter per bit of word B is used so that all the bits of B get inverted.
- Then 1 is added to 1's complement of B , by making $C_{in} = 1$. Thus we obtain the 2's complement of B .
- The 4-bit adder then adds A and 2's complement of B to produce the subtraction at its sum outputs $S_3 S_2 S_1 S_0$.
- The word $S_3 S_2 S_1 S_0$ represent the result of binary subtraction $(A - B)$ and carry output C_{out} represents the polarity of the result.
- If $A > B$ then $C_{out} = 0$ and the result is in true binary form but if $A < B$ then $C_{out} = 1$ and the result is negative and in the 2's complement form.

5.5.2 4-Bit Binary Parallel Adder / Subtractor Using IC 7483 :

Block diagram :

- The addition or subtraction of two 4-bit binary numbers can be obtained using the same circuit shown in Fig. 5.5.2.
- It makes use of the MSI 4 bit adder IC 7483 and it is called as an adder/subtractor circuit.
- The operation performed by this circuit (addition or subtraction) depends on the state of the mode of select input i.e. M in Fig. 5.5.2.



(C-396) Fig. 5.5.2 : 4-bit binary parallel adder/subtractor

- Number B is applied to the adder through four EX-OR gates. One input of each EX-OR gate is connected to the mode-select input (M).



Operation as Adder ($M = 0$) :

- For operation as an adder, the mode select (M) input is connected to ground,
 $\therefore M = 0$.
- Since $M = 0$, the output of the EX-OR gates will be the same number B which was applied at their inputs. This is because $0 \oplus 0 = 0$ and $0 \oplus 1 = 1$.
- Hence B_3, B_2, B_1 and B_0 will pass unchanged through the EX-OR gates. The carry input C_{in} is connected to M ,
 $\therefore C_{in} = 0$.
- Therefore the adder adds $A + B + C_{in} = A + B$ since $C_{in} = 0$. Thus with $M = 0$ addition of A and B will take place.

Operation as Subtractor ($M = 1$) :

- For operation as a subtractor, the mode select (M) input is connected to V_{CC} ,
 $\therefore M = 1$.
- Since $M = 1$, one input of each EX-OR gate is now 1. Hence each EX-OR gate acts as an inverter. This is because $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$.
- Thus each bit of word B is inverted by the EX-OR inverters. Thus we get the 1's complement of number B at the output of EX-OR gates.
- The carry input terminal C_{in} is connected to M ,
 $\therefore C_{in} = 1$.
- The inverted number B adds with $C_{in} = 1$ to give the 2's complement of B . Hence the adder will add A with the 2's complement of B and the result is actually the subtraction $A - B$.
- If $C_{out} = 0$ then the subtraction $(A - B)$ is positive and in true form. But if $C_{out} = 1$ then the subtraction $(A - B)$ is negative and in the 2's complement form.
- Thus with $M = 1$, this circuit works as a 2's complement subtractor.

5.6 BCD Addition :

- In BCD addition we have to deal with three different situations. Assume that two 4 bit BCD numbers A and B are being added. Then the three cases to be considered are,

Case 1 : Sum is equal to or less than 9 and carry is 0.

Case 2 : Sum is greater than 9 and carry is 0.

Case 3 : Sum is less than or equal to 9 but carry is 1.

- The BCD addition is to be carried out in an identical manner as normal binary addition.

Case 1 : Sum equal to or less than 9 with carry 0

- The addition of $(2)_{10}$ and $(6)_{10}$ in BCD is shown in Fig. 5.6.1.

Conclusion :

Decimal	BCD	
2	0 0 1 0	
+ 6	0 1 1 0	
Carry	1 1	
Answer : 8	0	1 0 0 0
		Final carry is 0
		Sum is a valid BCD number

$$(2)_{10} + (6)_{10} = (8)_{10}$$

(C-76) Fig. 5.6.1 : Illustration of case 1 in BCD addition

Case 2 : Sum greater than 9 but carry = 0

- In this case the sum of the two BCD numbers is greater than 9 that means it is an invalid BCD number. But the final carry is 0.
- So we have to correct the sum by adding decimal 6 or BCD 0110 to it. After doing this we get the correct BCD sum.
- Case 2 of BCD addition is illustrated in Fig. 5.6.2.

Decimal	BCD	
7	0 1 1 1	
+ 6	0 1 1 0	
Carry	1 1	
Answer : 13	1 1 0 1	← Invalid BCD number with 0 carry.
		← Add 6 for correction.
		← Valid BCD number with carry = 1
Final carry →	1	
Final : answer	0 0 0 1	0 0 1 1
	1	3
		← Correct BCD result.

$$(7)_{10} + (6)_{10} = (13)_{10}$$

(C-77) Fig. 5.6.2 : Illustration of case 2 in BCD addition

Case 3 : Sum less than or equal to 9 but carry = 1

- Here the sum of two BCD numbers is less than or equal to 9 i.e. it is a valid BCD number, and final carry = 1.



- A nonzero carry indicates that the answer is wrong and needs correction.
- Add $(6)_{10}$ or $(0110)_BCD$ to the sum to correct the answer.
- Case 3 of BCD addition is illustrated in Fig. 5.6.3.

Decimal	BCD
$+ 9 \longrightarrow$	$+ 1\ 0\ 0\ 1$
$+ 8 \longrightarrow$	$+ 1\ 0\ 0\ 0$
Carry \longrightarrow	$\boxed{1}$
Sum : 17 \longrightarrow	$\boxed{1} \ 0\ 0\ 0\ 1$
	↓ ↓
	0 0 0 1 0 0 0 1
	↓ ↓
	0 0 0 1 0 0 0 1

← Sum is valid BCD and carry = 1

← Incorrect BCD result

Add 6 to incorrect BCD result

$+ 0\ 0\ 0\ 1$	$+ 0\ 0\ 0\ 1$	← Incorrect BCD result
$+ 0\ 0\ 0\ 0$	$+ 0\ 1\ 1\ 0$	← Add 6
Carry \longrightarrow	$\boxed{0\ 0\ 0\ 1}$	$0\ 1\ 1\ 1$
	↓ ↓	
	1 7	

← Correct BCD result

$$(9)_{10} + (8)_{10} = (17)_{10}$$

(C-78) Fig. 5.6.3 : Illustration of case 3 in BCD addition

Ex. 5.6.1 : Add $(83)_{10}$ and $(34)_{10}$ in BCD.

Soln. :

Decimal	BCD
$(83)_{10}$	1 0 0 0 0 0 1 1
$+ (34)_{10}$	$+ 0 0 1 1$ 0 1 0 0
Carry	$\boxed{1}$
$(117)_{10}$	Sum : $\begin{array}{c} 1 0 1 1 \\ \boxed{1} 0 1 1 \end{array}$
	Invalid BCD Valid BCD
	Sum > 9, Carry = 0
	∴ Correction required

Add 6 to the invalid BCD, for correction.

$1\ 0\ 1\ 1$	$0\ 1\ 1\ 1$	
$+ 0\ 1\ 1\ 0$	$0\ 0\ 0\ 0$	6 is added only to the invalid BCD.
Carry \longrightarrow	$\boxed{1}\ 1$	
$\boxed{1}$	$0\ 0\ 0\ 1$	
$0\ 0\ 0\ 1$	$0\ 0\ 0\ 1$	Correct BCD result.
↓ ↓	↓ ↓	
1 1	7	

(C-6578)

$$\therefore (83)_{10} + (34)_{10} = (117)_{10}$$

...Ans.

Ex. 5.6.2 : Perform addition in BCD format
 $(79)_{BCD} + (16)_{BCD}$

Soln. :

$(79)_{BCD} + (16)_{BCD}$:

Decimal	BCD
$+ 79$	0 1 1 1 1 0 0 1
$+ 16$	0 0 0 1 0 1 1 0
	Carry $\boxed{1}\ 1\ 1$
	95
	$\begin{array}{c} 1 0 0 0 \\ \boxed{1} 1 1 1 \end{array}$
	Valid BCD Invalid BCD

(C-5229)

Add $(6)_{10}$ to invalid BCD.

\therefore	$+ 0\ 0\ 0\ 0$	$1\ 1\ 1\ 1$
Carry	$\boxed{1}\ 1\ 1$	$1\ 0\ 0\ 1$
	$\boxed{1}\ 0\ 1\ 0\ 1$	
	↓ ↓	
	9 5	

(C-5230)

$$\therefore (79)_{BCD} + (16)_{BCD} = (95)_{BCD}$$

...Ans.

5.6.1 BCD Adder using MSI IC 7483 :

SPPU : Dec. 10, May 11, May 12, Dec. 12, May 18

University Questions

Q. 1 Draw and explain 4-bit BCD adder using IC 7483.

(Dec. 10, May 11, 4 Marks, May 18, 6 Marks)

Q. 2 Describe the working of BCD adder using 7483 with the help of diagram. **(May 12, 8 Marks)**

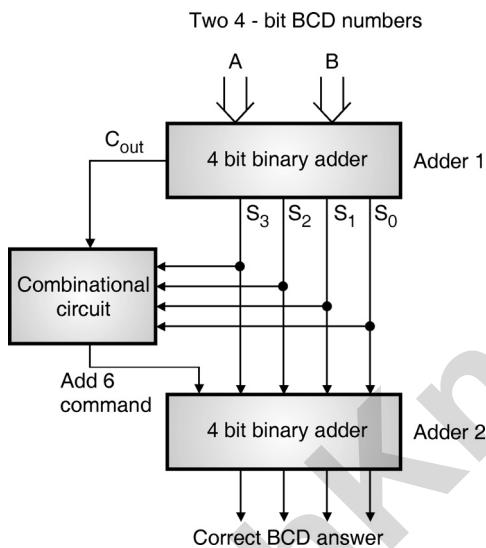
Q. 3 Draw and explain 4-bit BCD adder using IC7483. Also explain with example addition of numbers with carry. **(Dec. 12, 8 Marks)**

- BCD adder adds two BCD digits and produces a BCD digit. But remember that a BCD digit cannot be greater than 9.
- The two given 4-bit BCD numbers are to be added using the rules of binary addition.
- If sum is less than or equal to 9 and carry = 0, then no correction is necessary. The sum obtained is correct and in the true BCD form.
- But if sum is invalid BCD or carry = 1, then the result is wrong and needs correction.
- The wrong result can be corrected by adding six i.e. $(0110)_BCD$ to it.

Block Diagram of BCD Adder :

- From the points stated above, we understand that the 4-bit, BCD adder should consist of the following blocks :

1. A 4-bit binary adder to add the given two 4-bit BCD numbers A and B.
 2. A combinational circuit to check if sum is greater than 9 or carry = 1.
 3. Another 4-bit binary adder to add six (0110) to the incorrect sum if sum > 9 or carry = 1.
- The block diagram of such a BCD adder is shown in Fig. 5.6.4.
 - So we have to design the combinational circuit that finds out whether the sum is greater than 9 or carry = 1.



(C-397) Fig. 5.6.4 : Block diagram of BCD adder

Design of Combinational Circuit :

- The output of combinational circuit should be 1 if the sum produced by adder 1 is greater than 9 i.e. 1001.
- The truth table is as follows :

(C-6180) Table 5.6.1 : Truth table for combinational circuit design

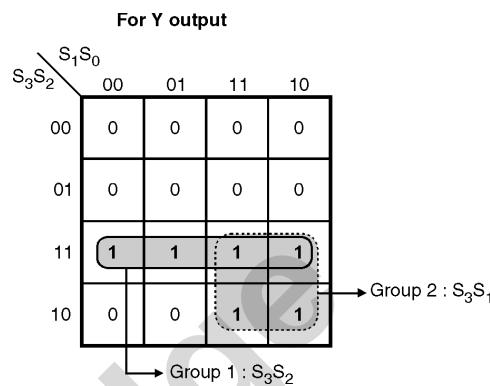
Inputs				Output
S ₃	S ₂	S ₁	S ₀	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Sum bits of adder - 1 →

Sum is valid BCD number ∴ Y = 0

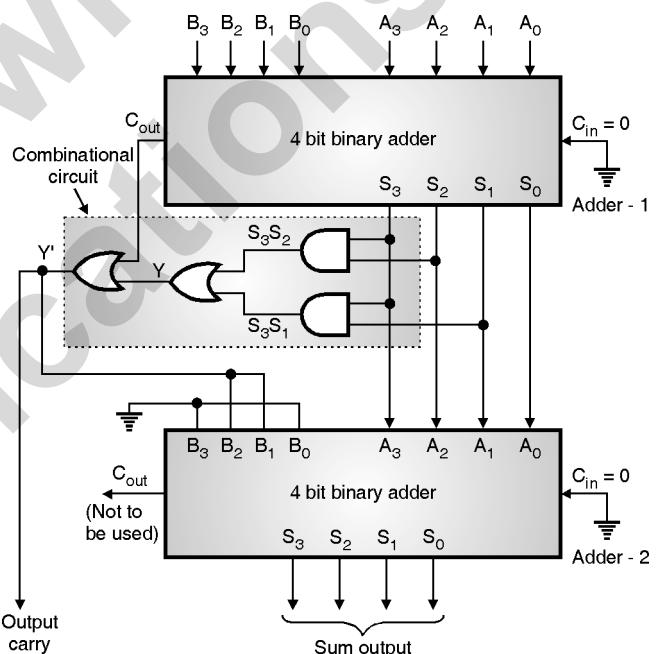
Sum is invalid BCD number ∴ Y = 1

Write K-map :



(C-398) Fig. 5.6.5 : K-map for Y output

- The boolean expression is,
- $$Y = S_3S_2 + S_3S_1$$
- The complete BCD adder is shown in Fig. 5.6.6.



(C-399) Fig. 5.6.6 : 4-bit BCD adder

- The output of the combinational circuit should be 1 if C_{out} of adder-1 is high or if the output of adder-1 is greater than 9.
- Therefore Y is ORed with C_{out} of adder 1 as shown in Fig. 5.6.6.
- The output of combinational circuit is connected to B₁B₂ inputs of adder-2 and B₃ = B₁ = 0 as they are connected to ground permanently.
- This makes B₃ B₂ B₁ B₀ = 0 1 1 0 if Y' = 1.
- The sum outputs of adder-1 are applied to A₃A₂A₁A₀ of adder-2.



- The output of combinational circuit is to be used as final output carry and the carry output of adder-2 is to be ignored.

Operation :

Case I : Sum \leq 9 and carry = 0

- The output of combinational circuit $Y' = 0$. Hence $B_3 B_2 B_1 B_0 = 0000$ for adder-2.
- Hence output of adder-2 is same as that of adder-1.

Case II : Sum $>$ 9 and carry = 0

- If $S_3 S_2 S_1 S_0$ of adder-1 is greater than 9, then output Y' of combinational circuit becomes 1.
 $\therefore B_3 B_2 B_1 B_0 = 0110$ (of adder-2)
- Hence six (0110) will be added to the sum output of adder-1.
- We get the corrected BCD result at the sum output of adder-2.

Case III : Sum \leq 9 but carry = 1

- As carry output of adder-1 is high, $Y' = 1$.
 $\therefore B_3 B_2 B_1 B_0 = 0110$ (of adder-2)
 $\therefore 0110$ will be added to the sum output of adder-1.
- We get the corrected BCD result at the sum output of adder-2.
- Thus the four bit BCD addition can be carried out using the binary adder.

5.7 BCD Subtractor using MSI IC 7483 :

BCD subtraction can be performed using two methods :

- Using 9's complement
- Using 10's complement

5.7.1 BCD Subtraction using 9's Complement :

- The 9's complement of a BCD number can be obtained by subtracting it from 9.
- For example 9's complement of 1 is 8. The 9's complement of various digits are given in Table 5.7.1.

Table 5.7.1 : 9's complement of various decimal digits

Decimal digit	0	1	2	3	4	5	6	7	8	9
9's complement	9	8	7	6	5	4	3	2	1	0

Procedure for BCD subtraction :

- The BCD subtraction using 9's complement is performed as follows :

Step 1 : Obtain the 9's complement of number B (i.e the number to be subtracted).

Step 2 : Add A and 9's complement of B.

Step 3 : If a carry is generated in step 2 then add it to the sum to obtain the final result. The carry is called as end around carry.

Step 4 : If carry is not produced then the result is negative and in the 9's complement form. So take 9's complement of the result.

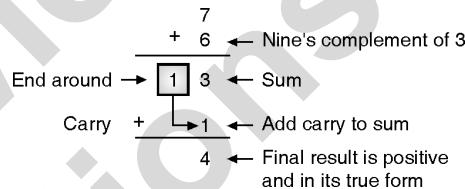
Ex. 5.7.1 : Subtract $(3)_{10}$ from $(7)_{10}$ in BCD.

Soln. :

Step 1 : Obtain 9's complement of $(3)_{10}$:

9's complement of $(3)_{10}$ is $9 - 3 = 6$.

Step 2 : Add 7 and nine's complement of 3 : (C-6258)



5.7.2 4-Bit BCD Subtractor using 9's Complement Method :

SPPU : Dec. 09, Dec. 11

University Questions

- Q. 1** Draw and explain 4-bit BCD subtractor using IC 7483. **(Dec. 09, 5 Marks)**
- Q. 2** What is the use of 7483 chip ? Draw and explain nine's complement used in BCD subtractor using 7483. **(Dec. 11, 8 Marks)**

Operation :

- The circuit diagram of a 4-bit BCD subtractor is shown in Fig. 5.7.1. It consists of four binary parallel adders (IC 7483).
- Adder – 1 obtains the 9's complement of number B.
- Adders – 2 and 3 form the normal 4-bit BCD adder with a facility to add $(6)_{10}$ i.e $(0110)_2$ for correction.
- Adder – 2 adds number A with the 9's complement of number B. The combinational circuit associated with adder – 3 will correct the sum by adding $(6)_{10}$ or $(0110)_2$ if necessary.
- The output of this combinational circuit is used further as a carry. At the output of adder – 3 we get the correct BCD sum of A and 9's complement of B.
- Adder – 4 is used to either add 1 to the output of adder – 3 or take the 9's complement of the output of adder – 3 depending on the status of carry as follows :

Adder 4 operation

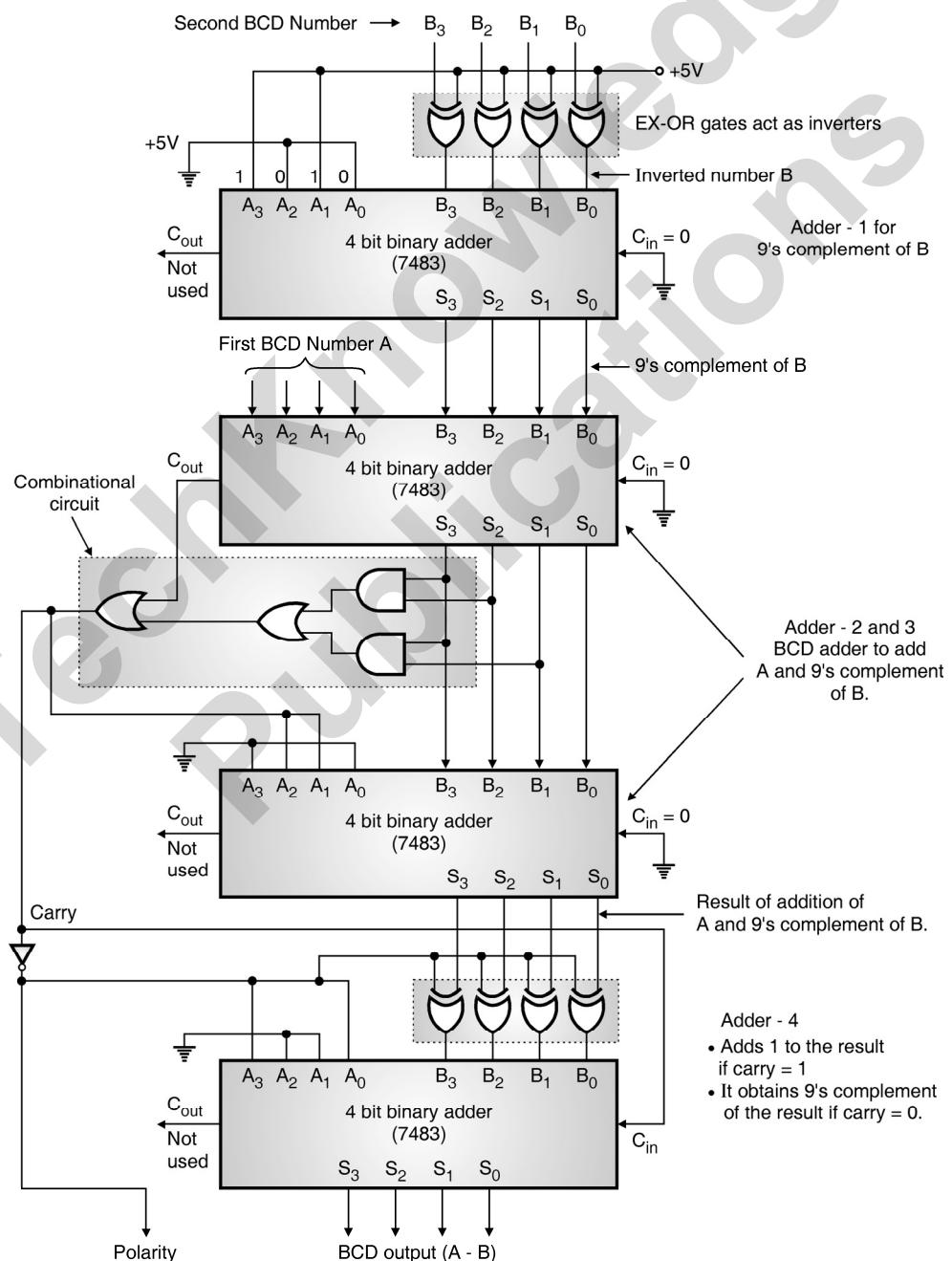
If carry = 1 : Add 1 to the sum output of adder 3. This happens because $C_{in} = 1$ for adder 4.

If carry = 0 : Take 9's complement of sum output of adder 3.

5.7.3 BCD Subtraction using 10's Complement :

- The 10's complement is obtained by adding 1 to the 9's complement. The 10's complement can be used to perform the BCD subtraction as follows :

- Step 1 :** Obtain the 10's complement of subtrahend (number to be subtracted).
- Step 2 :** Add the minuend to the 10's complement of subtrahend.
- Step 3 :** Discard carry. If carry is 1 then the answer is positive and in its true form.
- Step 4 :** If carry is not produced then the answer is negative. So take 10's complement to get the answer.



(C-401) Fig. 5.7.1 : 4-bit BCD subtractor using 9's complement method



Ex. 5.7.2 : Perform the subtraction $(9)_{10} - (4)_{10}$ in BCD using the 10's complement.

Soln. : (C-6193)

Step 1 : Obtain the 10's complement of $(4)_{10}$:

$$\begin{array}{r} \text{9's complement of } 4 = 9 - 4 = (5)_{10} \\ + 1 \\ \hline \end{array}$$

$$\text{10's complement of } 4 \longrightarrow (6)_{10}$$

Step 2 : Add $(9)_{10}$ and 10's complement of $(4)_{10}$:

$$\begin{array}{r} (9)_{10} \rightarrow 1 \ 0 \ 0 \ 1 \\ + (6)_{10} \rightarrow 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 1 \ 1 \ 1 \leftarrow \text{Invalid BCD and carry } = 0 \end{array}$$

Step 3 : Add $(6)_{10}$:
+ 0 1 1 0 ← Add $(0110)_2$ for correction
Carry → 1 1

Discard final carry → $\boxed{1} \ 0 \ 1 \ 0 \ 1$ → Answer is positive and in true BCD form

5.7.4 4-bit BCD Subtraction using 10's Complement Method :

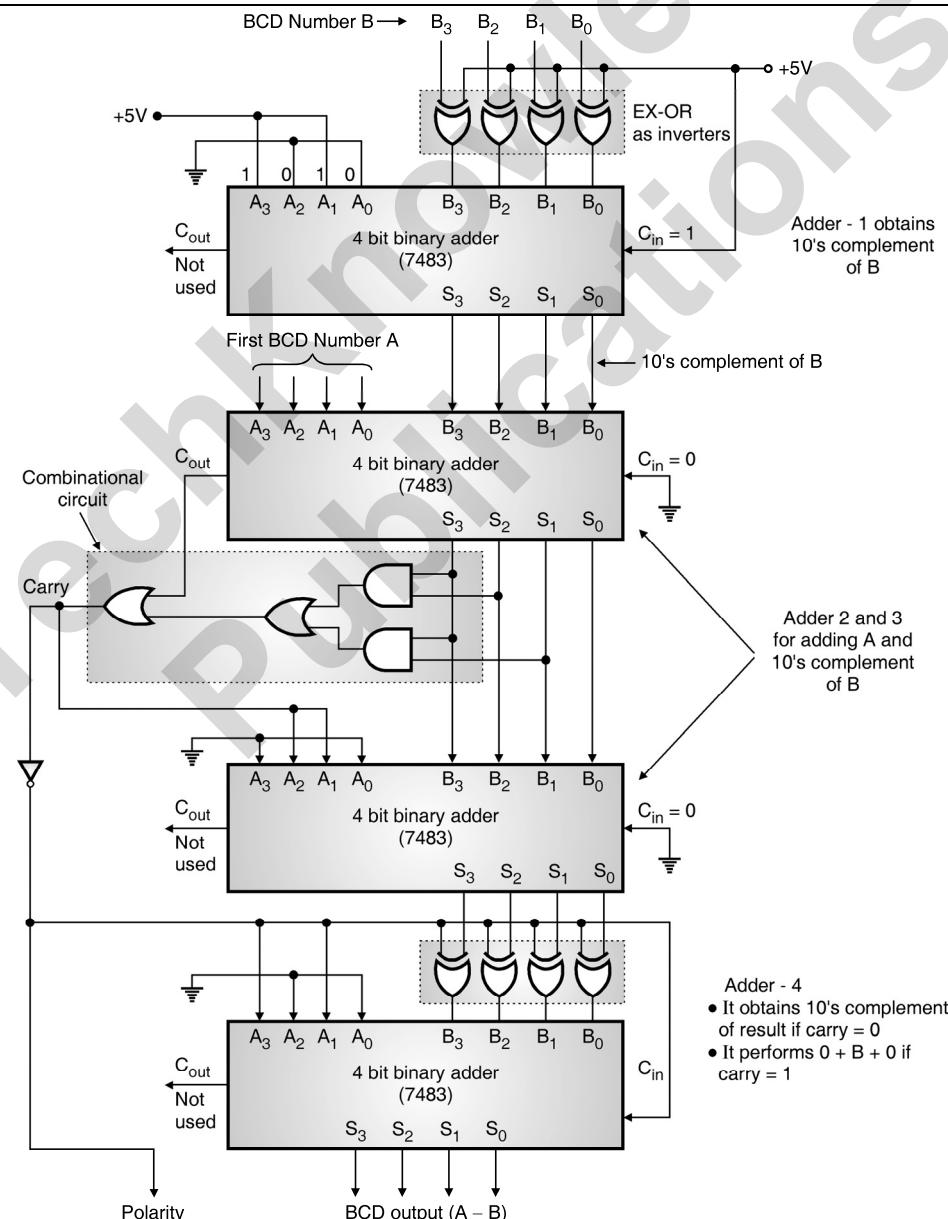
SPPU : Dec. 09

University Questions

Q. 1 Draw and explain 4-bit BCD subtractor using IC 7483. (Dec. 09, 5 Marks)

Circuit diagram :

- The circuit diagram of a 4-bit BCD subtractor using the 10's complement method is shown in Fig. 5.7.2.
- This circuit consists of four, 4-bit binary adders (IC 7483).



(C-402) Fig. 5.7.2 : BCD subtraction using 10's complement method

**Operation :**

- The operations to be performed by this circuit are as follows :
 1. Obtain the 10's complement of number B (i.e the number to be subtracted).
 2. Add number A and 10's complement of B using BCD addition.
 3. If carry is not generated then take the 10's complement of the result.

Adder - 1

- Adder - 1 performs the following operations.
- Adder - 1 produces the 10's complement of number B.
- Number B is inverted using the EX-OR gates and then $C_{in} = 1$ is added to it to obtain 2's complement of B.
- $A_3 A_2 A_1 A_0 = 1010$ i.e. $(10)_{10}$. Adder - 1 adds 1010 and 2's complement of number B. So actually it performs the subtraction $(10 - B)$ to obtain the 10's complement of B.
- Thus we obtain 10's complement of B at the output of Adder - 1.

Adders- 2 and 3

- Adders 2 and 3 together form the normal BCD adder discussed in the earlier sections.
- Adder - 2 adds number A to 10's complement of B.
- Adder - 3 adds six (0110) to the result of this addition, if the correction is necessary.
- Thus at the output of adder - 3 we get the correct BCD equivalent of $(A + 10's \text{ complement of } B)$.
- The output of the combinational circuit is treated as carry. It is passed to adder - 4.

Adder - 4

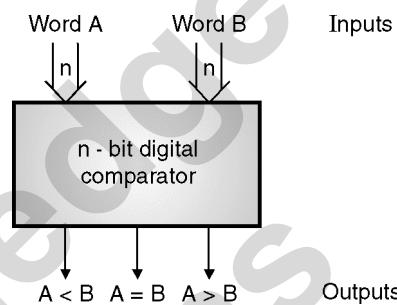
- Depending on the status of carry obtained from the combinational circuit, the adder behaves in the following manner.
- If carry = 0, then due to inverter used $A_3 A_2 A_1 A_0 = 1010$ i.e. $(10)_{10}$ and carry input $C_{in} = 1$. Also the EX-OR gates will act as inverters.
- Hence adder 4 will add $(10)_{10}$ and the 2's complement of adder 3 output. So what it actually does is, it takes the 10's complement $(10 - \text{result})$ of the result.
- But if carry = 1 then adder - 4 will pass the adder - 3 output unchanged.

5.8 Magnitude Comparators :**Definition :**

- Digital(or magnitude) comparator is a combinational circuit, which compares the two n-bit binary words A and B applied at its input and produces three different outputs : $A < B$, $A = B$ and $A > B$.

Block diagram :

- The block diagram of an n bit digital comparator is shown in Fig. 5.8.1.
- The comparator has three outputs namely $A > B$, $A = B$ and $A < B$.
- Depending on the result of comparison, one of these outputs will go high.



(C-403) Fig. 5.8.1 : Block diagram of an n-bit comparator

5.8.1 A 2-Bit Comparator :

SPPU : Dec. 12

University Questions

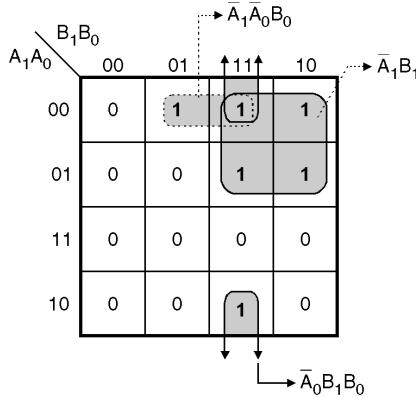
- Q. 1** Design 2-bit magnitude comparator using logic gates. Assume that A and B are 2-bit inputs. The outputs of comparator should be $A > B$, $A = B$, $A < B$. **(Dec. 12, 8 Marks)**

Truth table :

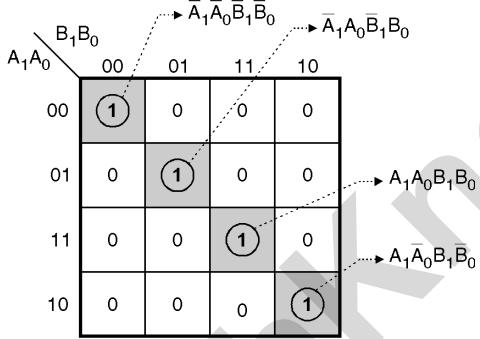
- For a 2-bit comparator, each input word A and B is 2 bit long.
- The truth table of a 2-bit comparator is shown in Table 5.8.1.

(C-406(a)) Table 5.8.1 : Truth table for a 2-bit comparator

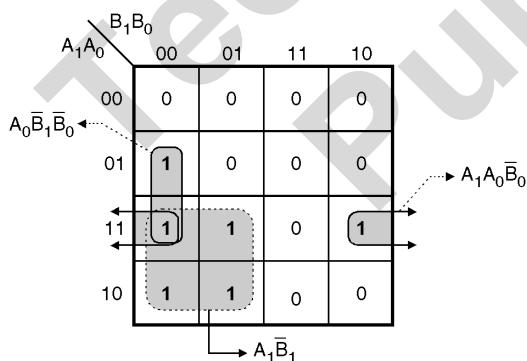
A ₁	A ₀	B ₁	B ₀	Outputs		
				A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

K-maps :**For A < B :**

(C-406) Fig. 5.8.2(a) : K-map for output A < B

For A = B :

(C-406) Fig. 5.8.2(b) : K-map for output A = B

For A > B :

(C-407) Fig. 5.8.2(c) : K-map for output A > B

- The K-maps for the three outputs and corresponding simplified expressions are shown in Figs. 5.8.2(a), (b) and (c).
- From these K-maps we get the simplified expressions for the three outputs of comparator are as follows :

For A < B

- Simplified expression :

$$A < B = \bar{A}_1 \bar{A}_0 B_0 + \bar{A}_1 B_1 + \bar{A}_0 B_1 B_0$$

For A > B

- Simplified expression :

$$A > B = A_0 \bar{B}_1 \bar{B}_0 + A_1 A_0 \bar{B}_0 + A_1 \bar{B}_1$$

Simplification for output A = B :

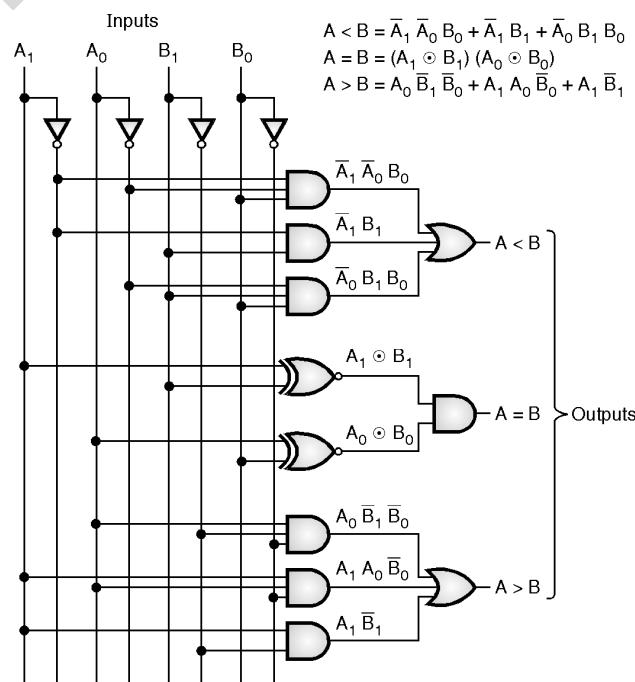
- Refer the K-map for output $A = B$ shown in Fig. 5.8.2(b).
- The expression for $A = B$ is given by,

$$\begin{aligned} (A = B) &= \bar{A}_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + \bar{A}_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \bar{A}_0 B_1 \bar{B}_0 \\ &= \bar{A}_0 \bar{B}_0 (\bar{A}_1 \bar{B}_1 + A_1 B_1) + A_0 B_0 (\bar{A}_1 \bar{B}_1 + A_1 B_1) \\ &= (\bar{A}_1 \bar{B}_1 + A_1 B_1) (\bar{A}_0 \bar{B}_0 + A_0 B_0) \end{aligned}$$

$$\therefore (A = B) = (A_1 \odot B_1) (A_0 \odot B_0) \text{ where } \odot = \text{EX-NOR}$$

Logic diagram for a two bit comparator :

- The logic diagram for the 2-bit digital comparator is shown in Fig. 5.8.3.
- This diagram is drawn by referring to the simplified expressions of the outputs.

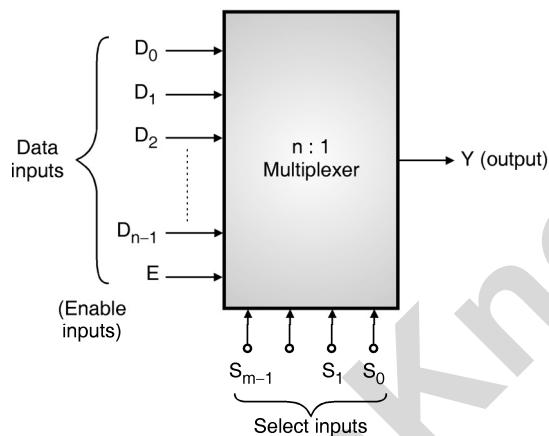


(C-408) Fig. 5.8.3 : Logic diagram for 2-bit comparator

5.9 Multiplexer (Data Selector) :

Definition and block diagram :

- A multiplexer is a combinational circuit with n data inputs one output and m select inputs, which selects one of the n data inputs and routes (connects) it to the output.
- The selection of one of the n inputs is done with the help of the select inputs.
- Multiplexer is a special type of combinational circuit. The block diagram of an n-to-1 multiplexer is shown in Fig. 5.9.1(a).

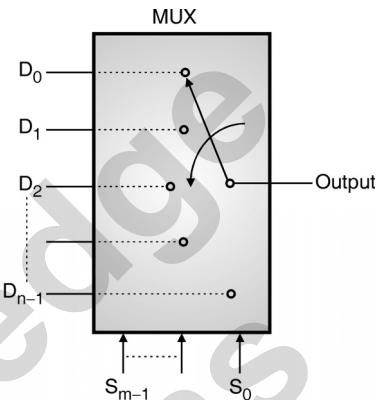


(C-413) Fig. 5.9.1(a) : Block diagram of an n : 1 multiplexer

- As shown, there are n-data inputs (D_0, D_1, \dots, D_{n-1}), one output Y and m select inputs, S_0, S_1, \dots, S_{m-1} .
- The relation between the number of data inputs (n) and number of select inputs (m) is as follows :
$$n = 2^m$$
- A multiplexer is a digital circuit which selects one of the n data inputs and routes (connects) it to the output.
- The selection of one of the n inputs is done with the help of the select inputs.
- To select n inputs we need m select lines such that $2^m = n$.
- Depending on the digital code applied at the select inputs, one out of n data sources is selected and transmitted to the single output Y.
- E is called as a strobe or enable input which is useful for cascading.
- It is generally an active low terminal, that means it will perform the required operation when it is low.

Equivalent Circuit :

- The equivalent circuit of an n:1 multiplexer is as shown in Fig. 5.9.1(b).
- As shown in Fig. 5.9.1(b) the multiplexer acts like a digitally controlled single pole, multiple way switch.



(C-413) Fig. 5.9.1(b) : Equivalent circuit of a multiplexer

- The output gets connected to only one of the n data inputs at given instant of time via the single pole multiple throw rotary switch.
- The position of the rotary arm will be dependent on the status of the m select inputs. It will connect the output to the selected input.

5.9.1 Necessity of Multiplexers :

- In most of the electronic systems, the digital data is available from more than one sources. It is necessary to route this data over a single line.
- Under such circumstances we require a circuit which selects one of the many sources at a time.
- This circuit is nothing else but a multiplexer, which has many inputs, one output and some select inputs.
- Multiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

5.9.2 Advantages of Multiplexers :

1. It reduces the number of wires, required to be used.
2. A multiplexer reduces the circuit complexity and cost.
3. We can implement many combinational circuits using MUX.
4. It simplifies the logic design.
5. It does not need the K maps for simplification.



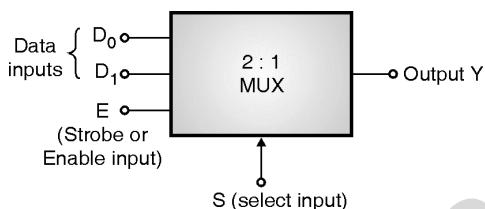
5.10 Types of Multiplexers :

- The types of multiplexers are :
 - 2 : 1 multiplexer.
 - 4 : 1 multiplexer.
 - 8 : 1 multiplexer.
 - 16 : 1 multiplexer.
 - 32 : 1 multiplexer.

5.10.1 2 : 1 Multiplexer :

Block diagram :

- The block schematic of a 2 : 1 multiplexer is shown in Fig. 5.10.1(a). It has two data inputs D_0 and D_1 , one select input S , an enable input and one output.
- The truth table of this MUX is shown in Fig. 5.10.1(b).



(a) Block diagram

Enable	Select Input S	Output Y
0	X	0
1	0	D_0
1	1	D_1

X = Don't care

(b) Truth table

(C-414) (a) Fig. 5.10.1 : 2 : 1 multiplexer

Truth table :

- Write a more elaborate truth table for 2 : 1 MUX as shown in Table 5.10.1.

(C-414(a)) Table 5.10.1 : Truth table of a 2 : 1 MUX

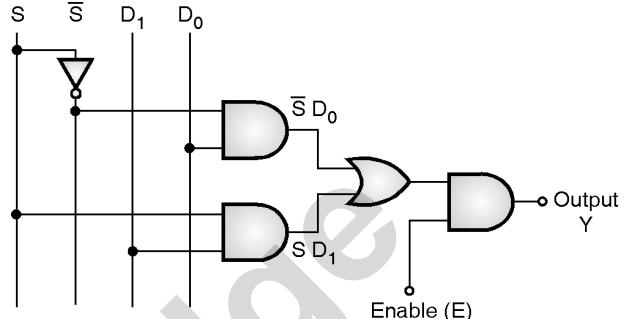
Enable E	Select S	D_1	D_0	Output Y
0	X	X	X	0
1	0	X	0	0
1	0	X	1	1
1	1	0	X	0
1	1	1	X	1

- From the truth table it is clear that $Y = 1$ for only two conditions shown by the shaded boxes.
- We can write down the Boolean expression by taking into consideration these two conditions as follows :

$$\therefore Y = E \bar{S} D_0 + E S D_1 = E (\bar{S} D_0 + S D_1)$$

Realization of a 2 : 1 MUX using gates :

- The realization using gates is shown in Fig. 5.10.2.



(C-415) Fig. 5.10.2 : Realization of 2 : 1 MUX using gates

5.10.2 A 4 : 1 Multiplexer :

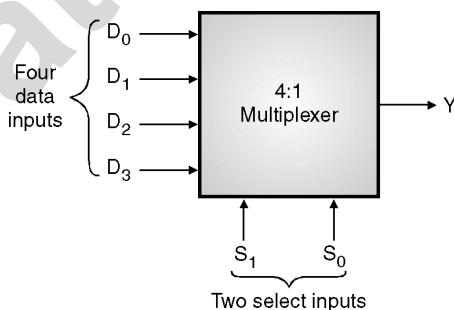
SPPU : May 07

University Questions

- Q. 1 Draw basic circuit diagram of 4 : 1 MUX and its truth table.
 (May 07, 6 Marks)

Block diagram and truth table :

- Fig. 5.10.3 shows the block diagram of a 4 : 1 multiplexer and Table 5.10.2 gives its truth table.



(C-416) Fig. 5.10.3 : 4 : 1 multiplexer

(C-416)(a) Table 5.10.2 : Truth table

Select Inputs		Output Y
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- Note that $n = 4$ hence number of select lines i.e. $m = 2$ so that $2^m = n$. The strobe terminal (G) has not been included to reduce the complexity.
- The truth table tells us that if $S_1 S_0 = 00$, the data bit D_0 is selected and routed to output.

$$\therefore Y = D_0 \quad \dots \text{when } S_1 S_0 = 00$$

- Similarly if $S_1 S_0 = 01$, then D_1 is selected and routed to the output.

$$\therefore Y = D_1 \quad \dots \text{when } S_1 S_0 = 01$$

$Y = D_2$ for $S_1 S_0 = 10$ and

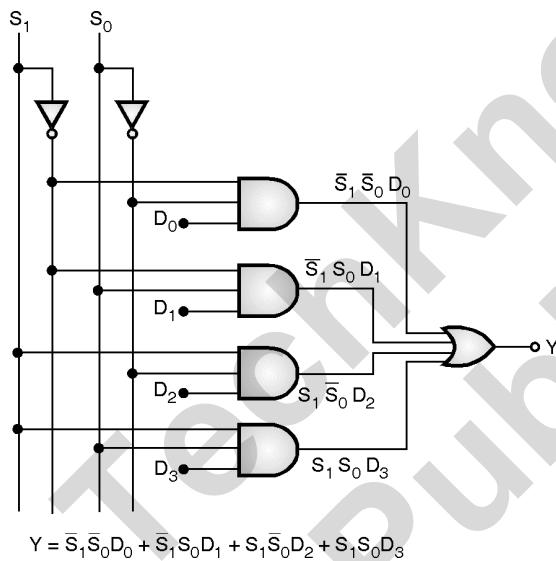
$Y = D_3$ for $S_1 S_0 = 11$

- The output will be high when the selected input (D_0, D_1 etc.) is 1. Hence the logical expression for output in the SOP form is as follows :

$$Y = \bar{S}_1 \bar{S}_0 D_0 + \bar{S}_1 S_0 D_1 + S_1 \bar{S}_0 D_2 + S_1 S_0 D_3$$

Realization of a 4 : 1 MUX using gates :

- This expression can be realized using basic gates as shown in Fig. 5.10.4.

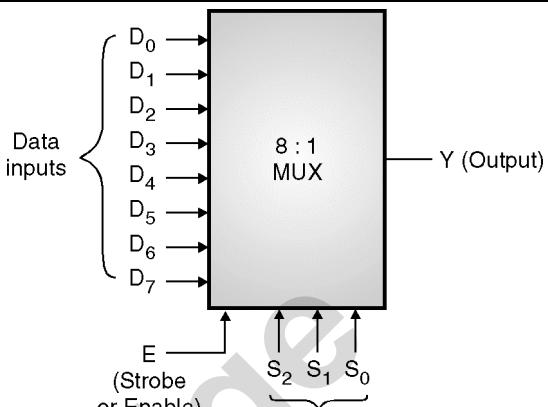


(C-417) Fig. 5.10.4 : Realization of 4 : 1 multiplexer using basic gates

5.10.3 8 : 1 Multiplexer :

Block diagram and truth table :

- The block diagram of an 8 : 1 MUX is shown in Fig. 5.10.5(a) and its truth table is shown in Fig. 5.10.5(b).
- It has eight data inputs, one enable input, three select inputs and one output.



(a) Block diagram

Enable E	Select input			Output Y
	S_2	S_1	S_0	
0	x	x	x	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

(b) Truth table

(C-419(a)) Fig. 5.10.5 : 8 : 1 multiplexer

Operating principle :

- When the strobe or enable input is 0, the output of the multiplexer will be 0 irrespective of any input.
- With $E = 1$, we can select any one of the eight data inputs and connect it to the output.
- For example if $S_2 S_1 S_0 = 0 1 1$ then the data input D_3 is selected and output Y will follow the selected input D_3 .

5.10.4 Applications of a Multiplexer :

SPPU : Dec. 06

University Questions

Q. 1 Explain any one application of multiplexer.

(Dec. 06, 8 Marks)

- Some of the important applications of a multiplexer are as follows :
 - It is used as a data selector to select one out of many data inputs.
 - It is used for simplification of logic design.

3. In the data acquisition system.
4. In designing the combinational circuits.
5. In the D/A converters.
6. To minimize the number of connections in a logic circuit.

5.11 Study of Different Multiplexer ICs :

- The multiplexer ICs available in market are given in Table 5.11.1.

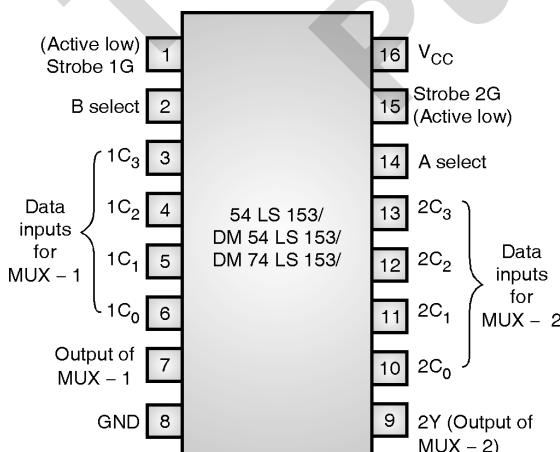
Table 5.11.1

IC Number	Description	Output
74157	Quad 2 : 1 Mux	Same as input (No inversion)
74158	Quad 2 : 1 Mux	Inverted output
74153	Dual 4 : 1 Mux	Same as input (No inversion)
74352	Dual 4 : 1 Mux	Inverted output
74151A	8 : 1 Mux	Complementary outputs
74152	8 : 1 Mux	Inverted output
74150	16 : 1 Mux	Inverted output

5.11.1 54LS 153/DM 54LS 153/DM 74LS 153

(Dual 4 : 1 Multiplexer) :

- The block diagram of these multiplexers is shown in Fig. 5.11.1(a).



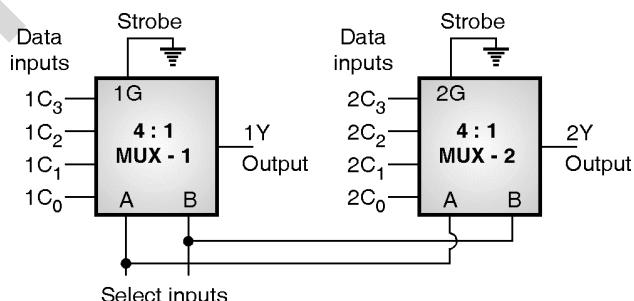
(C-3578) Fig. 5.11.1(a) : Pin configuration of Dual 4 : 1 multiplexer IC 54LS 153/DM 54LS 153/ DM 74LS 153

- These chips consist of two 4 : 1 multiplexers. Each multiplexer has a separate strobe (enable) input.
- There are two select inputs A and B which are common to both the 4 : 1 multiplexers inside the ICs.
- The strobe lines 1G and 2G are the active low enable lines for the two 4 : 1 multiplexer inside the IC. The truth table for each 4 : 1 MUX is shown in Table 5.11.2.

(C-6185) Table 5.11.2 : Truth table of 4:1 MUX

Select inputs		Data inputs				Strobe	Output
B	A	C ₀	C ₁	C ₂	C ₃	G	Y
X	X	X	X	X	X	1	0
C ₀ selected	0	0	0	X	X	X	0
	0	0	1	X	X	X	1
C ₁ selected	0	1	X	0	X	X	0
	0	1	X	1	X	X	1
C ₂ selected	1	0	X	X	0	X	0
	1	0	X	X	1	X	0
C ₃ selected	1	1	X	X	X	0	0
	1	1	X	X	X	1	0

- The functional diagram of IC 74153 is as shown in Fig. 5.11.1(b).



(C-3579) Fig. 5.11.1(b) : Functional diagram of IC-74153

- The function table is shown in Table 5.11.3.

(C-8133) Table 5.11.3 : Function table of a IC-74153

Inputs		Strobe \bar{G}	Output Y
Select inputs	B		
X	X	1	0
0	0	0	C ₀
0	1	0	C ₁
1	0	0	C ₂
1	1	0	C ₃

5.12 Multiplexer Tree/Cascading of Multiplexer :

- The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs.
- This is called as a multiplexer tree. This concept will be clear after solving the following examples.

Ex. 5.12.1 : Implement a 16 : 1 multiplexer using 4 : 1 multiplexers.

Dec. 05, 6 Marks, Dec. 07, 7 Marks,

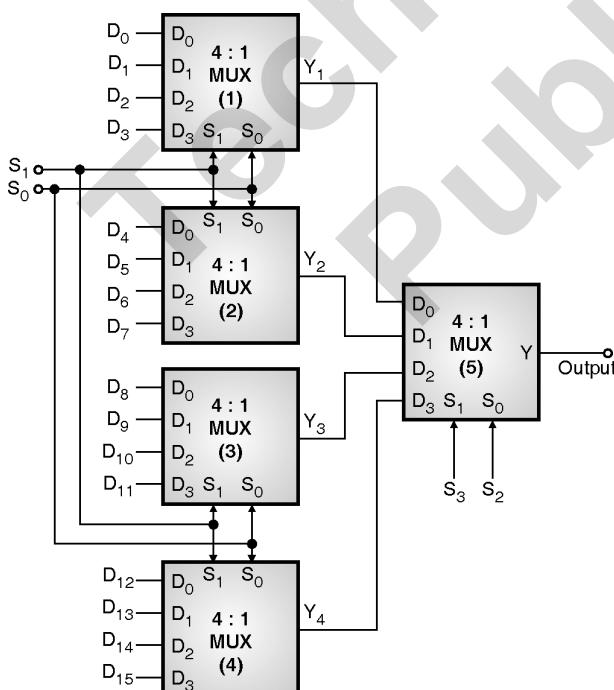
Dec. 08, May 11, 8 Marks

Soln. :

- Refer Fig. P. 5.12.1 for implementation.

Description :

- The select inputs S_1 and S_0 of the multiplexers 1, 2, 3 and 4 are connected together.
- The select inputs S_3 and S_2 are applied to the select inputs S_1 and S_0 of MUX-5.
- The outputs Y_1, Y_2, Y_3, Y_4 are applied to the data inputs D_0, D_1, D_2 and D_3 of MUX-5 as shown in Fig. P. 5.12.1.



(C-425) Fig. P. 5.12.1 : 16 : 1 multiplexer using 4 : 1 multiplexers

- The operation can be summarized using Table P. 5.12.1.

(C-6187) Table P. 5.12.1 : Summary of operation

Select inputs	Mux. Outputs				Final Output Y
	Y_1	Y_2	Y_3	Y_4	
0 0 0 0	D_0	D_4	D_8	D_{12}	D_0
0 0 0 1	D_1	D_5	D_9	D_{13}	D_1
0 0 1 0	D_2	D_6	D_{10}	D_{14}	D_2
0 0 1 1	D_3	D_7	D_{11}	D_{15}	D_3
0 1 0 0	D_0	D_4	D_8	D_{12}	D_4
0 1 0 1	D_1	D_5	D_9	D_{13}	D_5
0 1 1 0	D_2	D_6	D_{10}	D_{14}	D_6
0 1 1 1	D_3	D_7	D_{11}	D_{15}	D_7
1 0 0 0	D_0	D_4	D_8	D_{12}	D_8
1 0 0 1	D_1	D_5	D_9	D_{13}	D_9
1 0 1 0	D_2	D_6	D_{10}	D_{14}	D_{10}
1 0 1 1	D_3	D_7	D_{11}	D_{15}	D_{11}
1 1 0 0	D_0	D_4	D_8	D_{12}	D_{12}
1 1 0 1	D_1	D_5	D_9	D_{13}	D_{13}
1 1 1 0	D_2	D_6	D_{10}	D_{14}	D_{14}
1 1 1 1	D_3	D_7	D_{11}	D_{15}	D_{15}

$S_3S_2 = 00 \therefore \text{MUX-5 selects } Y_1$
 $S_3S_2 = 01 \therefore \text{MUX-5 selects } Y_2$
 $S_3S_2 = 10 \therefore \text{MUX-5 selects } Y_3$
 $S_3S_2 = 11 \therefore \text{MUX-5 selects } Y_4$

Ex. 5.12.2 : Design 12 : 1 mux using 4 : 1 multiplexers (with enable inputs). Explain the truth table of your circuit in short.

Dec. 09, 8 Marks, Dec. 17, 6 Marks

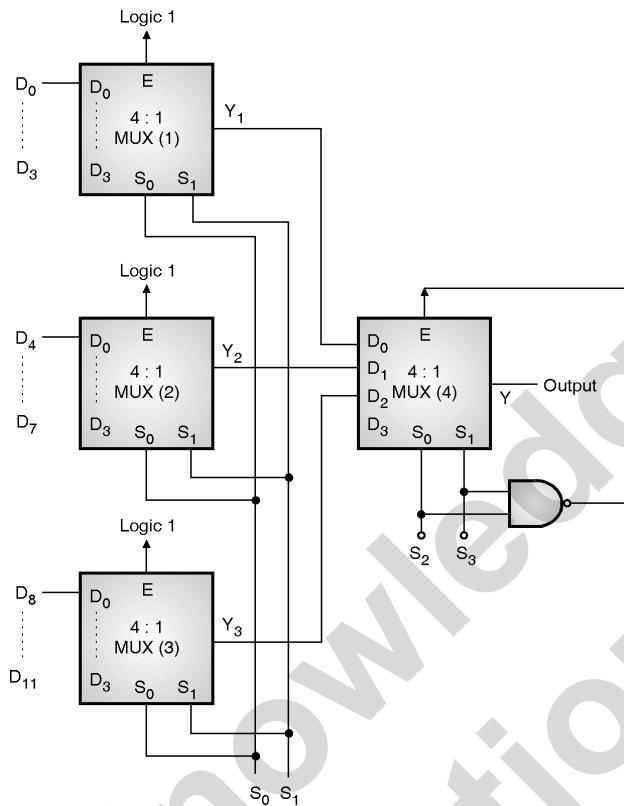
Soln. :

- The 12:1 MUX using 4 : 1 MUXes is as shown in Fig. P. 5.12.2 and its truth table is as shown in Table P. 5.12.2.

(C-8134) Table P. 5.12.2 : Truth table

Select inputs	MUX outputs			Final output Y	Comment
	Y_1	Y_2	Y_3		
0 0 0 0	D_0	D_4	D_8	D_0	$S_3S_2 = 00 \therefore \text{MUX-4 select } Y_1$
0 0 0 1	D_1	D_5	D_9	D_1	
0 0 1 0	D_2	D_6	D_{10}	D_2	
0 0 1 1	D_3	D_7	D_{11}	D_3	
0 1 0 0	D_0	D_4	D_8	D_4	$S_3S_2 = 01 \therefore \text{MUX-4 select } Y_2$
0 1 0 1	D_1	D_5	D_9	D_5	
0 1 1 0	D_2	D_6	D_{10}	D_6	
0 1 1 1	D_3	D_7	D_{11}	D_7	
1 0 0 0	D_0	D_4	D_8	D_8	$S_3S_2 = 10 \therefore \text{MUX-4 select } Y_3$
1 0 0 1	D_1	D_5	D_9	D_9	
1 0 1 0	D_2	D_6	D_{10}	D_{10}	
1 0 1 1	D_3	D_7	D_{11}	D_{11}	
1 1 0 0	X	X	X	0	$S_3S_2 = 11 \therefore \text{MUX-4 disabled.}$
1 1 0 1	X	X	X	0	
1 1 1 0	X	X	X	0	
1 1 1 1	X	X	X	0	

- The enable input E of the first three 4:1 multiplexers are connected to logic 1 permanently in order to enable them.
- The select lines S_0 and S_1 respectively of MUX 1, 2 and 3 are connected together as shown in Fig. P. 5.12.2.



(C-1708) Fig. P. 5.12.2 : 12:1 multiplexer using 4:1 multiplexers

Ex. 5.12.3 : Design 14 : 1 mux using 4 : 1 mux (with enable inputs). Explain the truth table of your circuit in short.

May 10, 8 Marks

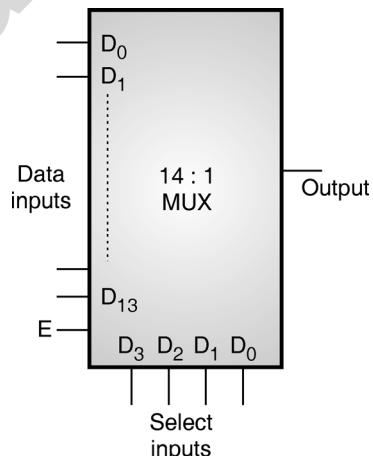
Soln. :

- The truth table of 14 : 1 MUX is as shown below :

(C-8135) Table P. 5.12.3 : Truth table of a 14 : 1 MUX

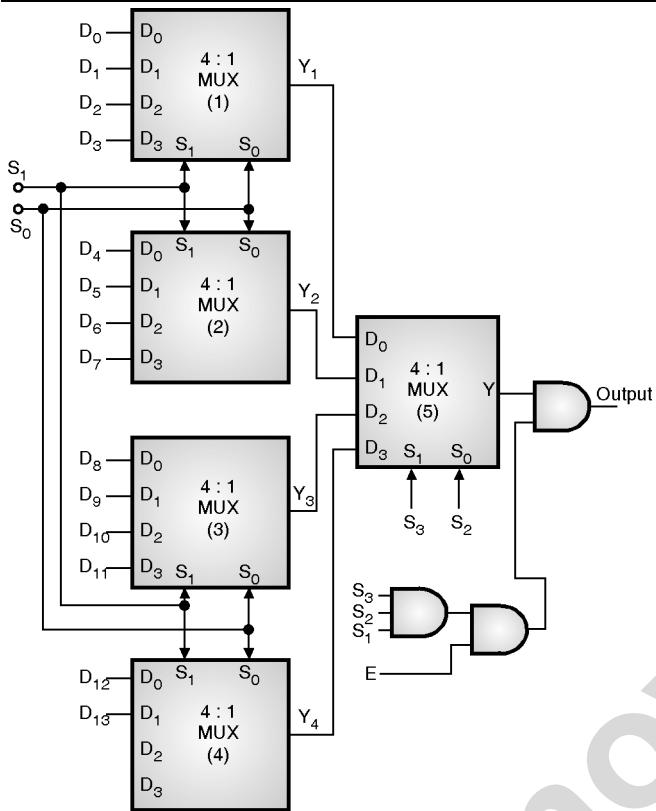
Enable input E	Select inputs				Output Y
	S_3	S_2	S_1	S_0	
0	X	X	X	X	0
1	0	0	0	0	D_0
1	0	0	0	1	D_1
1	0	0	1	0	D_2
1	0	0	1	1	D_3
1	0	1	0	0	D_4
1	0	1	0	1	D_5
1	0	1	1	0	D_6
1	0	1	1	1	D_7
1	1	0	0	0	D_8
1	1	0	0	1	D_9
1	1	0	1	0	D_{10}
1	1	0	1	1	D_{11}
1	1	1	0	0	D_{12}
1	1	1	0	1	D_{13}
X	1	1	1	0	0
X	1	1	1	1	0

Output is 0 for the invalid combinations of select inputs



(C-1711) Fig. P. 5.12.3(a)

- The realization of 14 : 1 MUX using 4 : 1 MUX is shown in Fig. P. 5.12.3(b).



(C-1712) Fig. P. 5.12.3(b) : 14 : 1 MUX using 4 : 1 multiplexers

5.13 Use of Multiplexers in Combinational Logic Design :

- Multiplexers ICs for 2 : 1 , 4 : 1 , 8 : 1 and 16 : 1 multiplexers are available.
- We can use them to implement the given Boolean expressions representing the combinational circuits.
- Thus it is possible to design and implement many combinational circuits by using multiplexer and a few logic gates.

Advantages :

- Advantages of using multiplexers for logic design are as follows :
 1. Logic design is simplified.
 2. It is not necessary to simplify the logic expression.
 3. It minimizes the number of ICs required to be used.

5.13.1 Implementation of a Logical Expression in the Standard SOP Form :

Use of MUX for combinational circuit design :

1. A truth table or logic expression in standard SOP or POS form is given to us.
2. We have to follow the design procedure given below to use MUX for implementing the given logical expression.

Design procedure :

Step 1 : Identify the decimal number corresponding to each minterm in the given expression as illustrated below :

$$\text{If } Y = \underbrace{\overline{ABC}}_0 + \underbrace{\overline{ABC}}_5 + \underbrace{ABC}_7$$

(C-6264)

Step 2 : The input lines of the multiplexer, corresponding to these numbers (0, 5 and 7) are connected to logic 1 level.

Step 3 : All the other input lines of multiplexer are connected to logic 0 level.

Step 4 : The inputs (A, B, C) are to be connected to the select inputs.

- The following example will make this concept clear.

Ex. 5.13.1 : Realize the logic function of the truth table given in Table P. 5.13.1 using a multiplexer.

(C-7617) Table P. 5.13.1

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

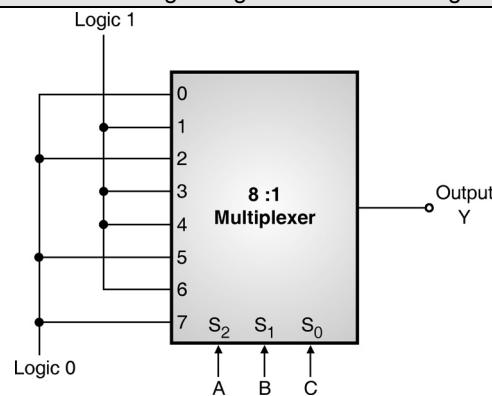
Soln. :

Step 1 : Express the output Y in the standard SOP form
 $Y = \sum m(1, 3, 4, 6)$

Step 2 : Connect the data inputs 1, 3, 4 and 6 to logic 1 and the remaining to logic 0.

Step 3 : Connect the inputs A, B, C to the select lines S₂, S₁ and S₀ respectively.

Step 4 : Draw the logic diagram as shown in Fig. P. 5.13.1.



(C-429) Fig. P. 5.13.1 : Logic diagram for Ex. 5.13.1

Ex. 5.13.2 : Implement the following expression using a multiplexer.

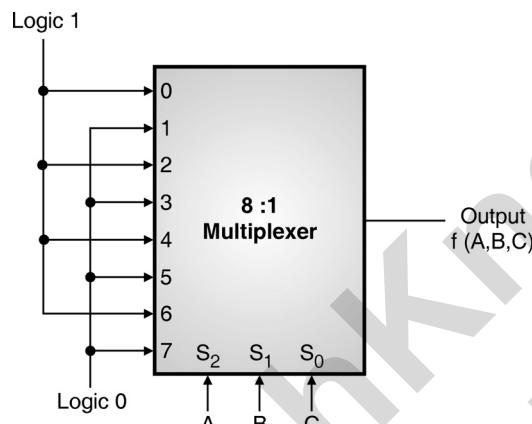
$$f(A, B, C) = \Sigma m(0, 2, 4, 6)$$

Soln. :

- Since there are three variables, the multiplexer having three select inputs should be used. Hence an 8 : 1 multiplexer as shown in Fig. P. 5.13.2 should be used.

Step 1 : Identify the decimal number corresponding to each minterm. The decimal numbers corresponding the minterms are 0, 2, 4 and 6.

Step 2 : Connect the data input lines 0, 2, 4 and 6 to logic 1 and the remaining lines (1, 3, 5, 7) to logic 0 as shown in Fig. P. 5.13.2.



(C-430) Fig. P. 5.13.2 : Implementation of a logic expression using a multiplexer

Step 3 : Connect the variables A, B and C to the select inputs.

Note : We can use IC 74151 which is an 8 : 1 multiplexer to implement Boolean equations using 8 : 1 MUX.

5.13.2 Use of 4 : 1 MUX to Realize a 4 Variable Function :

- It is very easy to use the 4 : 1 MUX to implement a 3 variable function.
- The procedure to be followed for this has been illustrated through the following examples.

Ex. 5.13.3 : Implement the logic function $f(A, B, C) = \Sigma m(1, 3, 4, 6)$ using a 4 : 1 multiplexer.

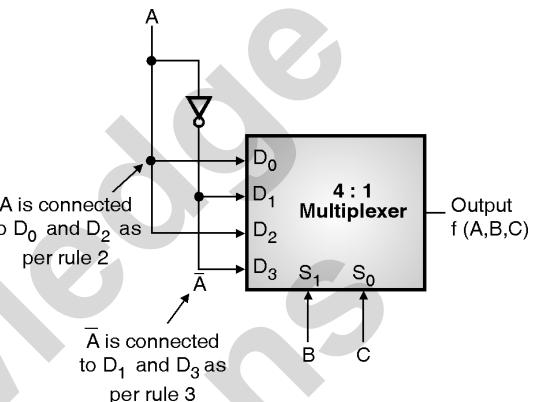
Soln. :

- The logic function to be implemented is

$$f(A, B, C) = \Sigma m(1, 3, 4, 6)$$

Step 1 : Apply two variables B and C to the select inputs :

- As shown in Fig. P. 5.13.3(a), the two input variables B and C are applied to the select lines S_1 and S_0 respectively.



(C-431) Fig. P. 5.13.3(a) : Logic diagram

Step 2 : Write the design table :

- The design table is shown in Fig. P. 5.13.3(b).

	D_0	D_1	D_2	D_3	← Data inputs
\bar{A}	0	1	2	3	Row 1
A	4	5	6	7	Row 2

Unused variable

The terms encircled correspond to the minterms producing a 1 output

(C-432) Fig. P. 5.13.3(b) : Design table

- The data inputs D_0 to D_3 have been written at the top of the table and the two possible values A and \bar{A} of the unused variable A have been written.
- In the eight boxes we enter the decimal numbers corresponding to the minterms (0 to 7) serially.
- Encircle those minterms corresponding to which the output is 1 (minterms 1, 3, 4, 6).

Step 3 : Check each column in the design table :

- The columns of design table are inspected using the following rules :

Rule 1 : If both the minterms in a column are not circled, then apply logic 0 to the corresponding data input. Note that there is no such column in our implementation table.

Rule 2 : If only the minterm in the second row is encircled (see columns 1 and 3) then "A" should be applied to that data input. Hence we should apply A to the D_0 and D_2 inputs.

Rule 3 : If only the minterm in the first row is encircled, (see columns 2 and 4), then A should be connected to that data input. Hence we should apply A to the D₁ and D₃ inputs.

Rule 4 : If both the minterms in a column are encircled, then apply a logic 1 to the corresponding data input.

- Note that there is no such column in our implementation table.

Step 4 : Draw the logic diagram :

- The logic diagram is as shown in Fig. P. 5.13.3(a).

Note : We can use IC 74153 which is a dual 4 : 1 multiplexer in order to implement the Boolean expressions using 4 : 1 MUX.

5.13.3 Use of 8 : 1 MUX to Realize a 4 Variable Function :

- It is very easy to use the 8 : 1 MUX to implement a 4 variable function.
- The procedure to be followed for this has been illustrated through the following examples.

Ex. 5.13.4 : Implement the following Boolean function using 8 : 1 multiplexer.

$$f(A, B, C, D) = \sum m(2, 4, 5, 7, 10, 14)$$

Soln. :

Step 1 : Apply the variables B, C and D to the select inputs :

As shown in Fig. P. 5.13.4(b), the three variables B, C and D are connected to the select lines S₂, S₁ and S₀ respectively.

Step 2 : Write the design table :

- The design table is as shown in Fig. P. 5.13.4(a).

D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{A}	0	1	(2)	3	(4)	(5)	6
A	8	9	(10)	11	12	13	(14)

Row 1 Row 2

Input to MUX : 0 0 1 0 \bar{A} \bar{A} A \bar{A}

The terms encircled correspond to the minterms producing a 1 output

(C-434) Fig. P. 5.13.4(a) : Design table

- In the sixteen boxes we have entered the decimal numbers corresponding to the minterms 0 to 15 serially.
- Encircle those minterms which correspond to an output = 1 (minterms 2, 4, 5, 7, 10, 14).

Step 3 : Inspect each column in the design table :

Rule 1 : If both the minterms in a column are not circled, then apply logic 0 to the corresponding data input. Note that there is no such column in our implementation table.

Rule 2 : If only the minterm in the second row is encircled then "A" should be applied to that data input.

Hence we should apply A to the D₆ input.

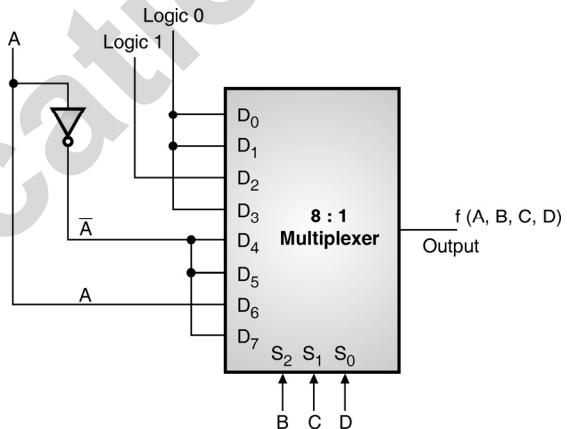
Rule 3 : If only the minterm in the first row is encircled, then \bar{A} should be connected to that data input

.Hence we should apply \bar{A} to the D₄, D₅ and D₇ inputs.

Rule 4 : If both the minterms in a column are encircled, then apply a logic 1 to the corresponding data 1 input. Note that there is no such column in our implementation table.

Step 4 : Draw the logic diagram :

- The logic diagram is as shown in Fig. P. 5.13.4(b).



(C-435) Fig. P. 5.13.4(b) : Logic diagram

Ex. 5.13.5 : Implement the following function using 4 : 1 multiplexers with active low strobe input :

$$f(A, B, C, D) = \sum m(2, 3, 5, 7, 8, 9, 12, 13, 14, 15)$$

Soln. :

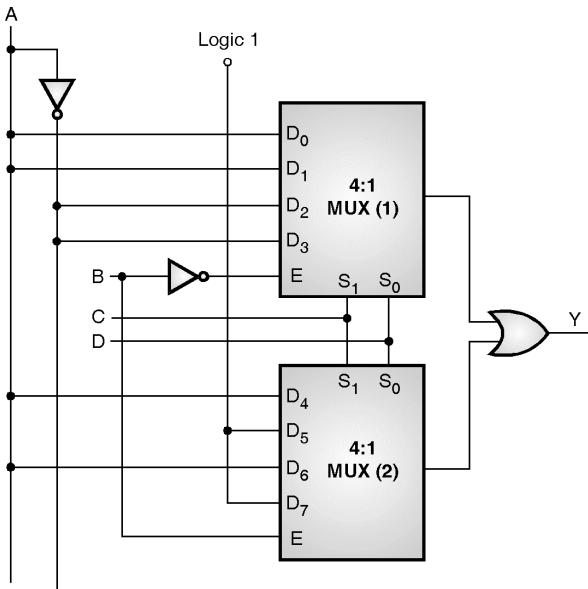
Step 1 : Write the design table :

D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{A}	0	1	(2)	(3)	4	(5)	6
A	(8)	(9)	10	11	(12)	(13)	(14)

Input to MUX : A A \bar{A} \bar{A} A 1 A 1

MUX - 1 MUX - 2

(C-1329(a))

Step 2 : Implementation of logic circuit :


(C-1330) Fig. P. 5.13.5

5.13.4 Implementation of a Logical Expression in the Non-standard SOP Form :

- Till now we have seen the implementation of expressions in the standard SOP form.
- But if the given expression is not in the standard form, then we have to first bring it to the standard form and then proceed.
- Example 5.13.6 illustrates this concept.

Ex. 5.13.6 : Implement the following Boolean expression using 8 : 1 multiplexer.

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

Soln. :

Step 1 : Convert the given expression to standard SOP form :

$$\begin{aligned} f(A, B, C, D) &= \bar{A} \bar{B} \bar{D} (C + \bar{C}) + A B C (D + \bar{D}) \\ &\quad + \bar{B} C D (A + \bar{A}) + \bar{A} C D (B + \bar{B}) \\ &= \bar{A} \bar{B} C \bar{D} + \bar{A} \bar{B} \bar{C} \bar{D} + A B C D \\ &\quad + A B C \bar{D} + A \bar{B} C D + \bar{A} \bar{B} C D \\ &\quad + \bar{A} B C D + \bar{A} \bar{B} \bar{C} D \end{aligned}$$

This is the expression in the standard SOP form.

$$\therefore f(A, B, C, D) = \sum m(2, 0, 15, 14, 11, 3, 7, 3)$$

$$\text{As } X + X = X$$

$$\therefore \bar{A} \bar{B} C D + \bar{A} B \bar{C} D = \bar{A} \bar{B} C D$$

$$\therefore f(A, B, C, D) = \sum m(0, 2, 3, 7, 11, 14, 15)$$

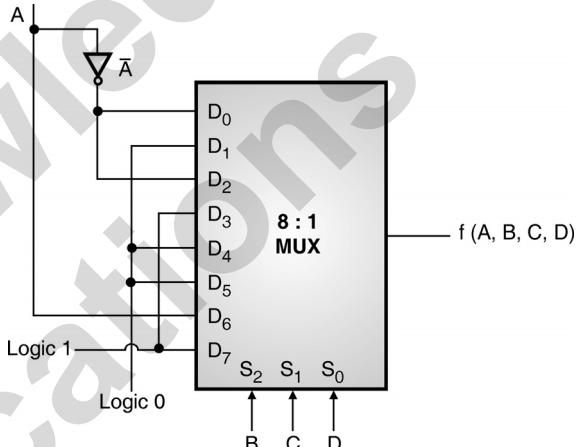
Step 2 : Write the design table :

(C-437(a)) Table P. 5.13.6 : Design table

Inputs	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{A}	(0)	1	(2)	(3)	4	5	6	(7)
A	8	9	10	(11)	12	13	(14)	(15)
Inputs to MUX	\bar{A}	0	\bar{A}	1	0	0	A	1

Step 3 : Implementation using 8 : 1 multiplexer :

- The implementation is as shown in Fig. P. 5.13.6.



(C-438) Fig. P. 5.13.6 : Implementation

5.13.5 Implementing a Standard POS Expression using Multiplexer :

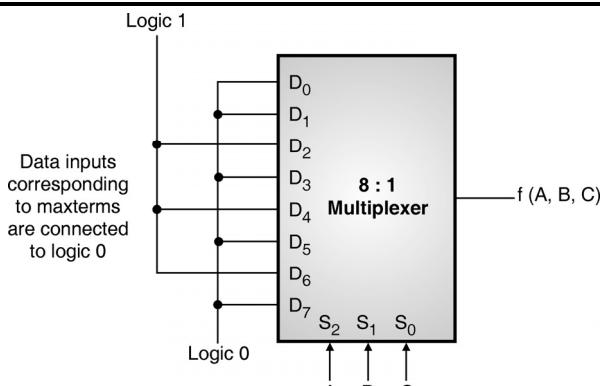
- Let us consider the Boolean expression in the standard POS form and its implementation using a multiplexer.

Ex. 5.13.7 : Implement the following logic function using the 8 : 1 multiplexer.

$$f(A, B, C) = \prod m(0, 1, 3, 5, 7)$$

Soln. :

- In the given expression, the maxterms have been specified.
- Hence we should connect the corresponding data inputs (D_0, D_1, D_3, D_5 and D_7) to logic 0.
- And connect the remaining data inputs to logic 1.
- Connect the input variables A, B, C to the select inputs S_2, S_1 and S_0 respectively.
- The logic diagram is as shown in Fig. P. 5.13.7.



(C-439) Fig. P. 5.13.7 : Implementation of standard POS expression

Ex. 5.13.8 : Implement the following logic function using 4 : 1 multiplexer.

$$f(A, B, C) = \prod M(0, 1, 3, 5, 7)$$

Soln. :

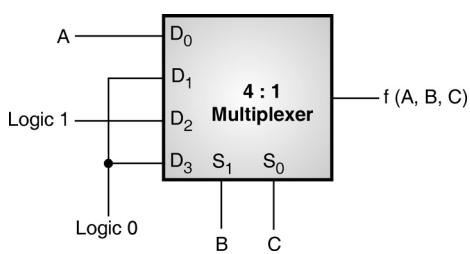
- Connect B and C to the select inputs S_1 and S_0 respectively.
- For the connections of data inputs (D_0 to D_3) and A write down the design table as shown in Fig. P. 5.13.8(a).

	D_0	D_1	D_2	D_3
\bar{A}	0	1	(2)	3
A	(4)	5	(6)	7
Input to MUX:	A	0	1	0

Encircle the maxterms which are not included in the given Boolean expression.

(C-440) Fig. P. 5.13.8(a) : Design table

- Note that as the given expression is in the POS form, we should encircle those maxterms which are not included in the given Boolean expression.
- The rules for deciding the input to the data lines (D_0 to D_3) are however the same as those stated earlier for the SOP form.
- The logic diagram is as shown in Fig. P. 5.13.8(b).



(C-441) Fig. P. 5.13.8(b) : Implementation using 4 : 1 multiplexer

Ex. 5.13.9 : Implement a full adder circuit using two 4 : 1 multiplexers.

Dec. 04, 3 Marks, Dec. 14, Dec. 19, 6 Marks

Soln. :

Step 1 : Write the truth table of full adder :

(C-7621) Table P. 5.13.9 : Truth table of a full adder

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 2 : Write the design tables for sum and carry outputs :

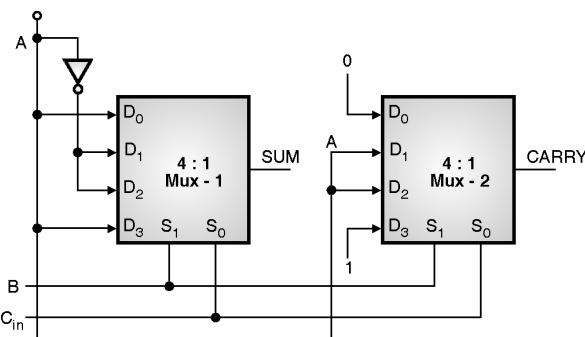
Sum			
D_0	D_1	D_2	D_3
\bar{A}	0	(1)	(2)
A	(4)	5	6
Input to Mux	A	\bar{A}	\bar{A}

Carry			
D_0	D_1	D_2	D_3
\bar{A}	0	1	(3)
A	4	(5)	(6)
Input to Mux	0	A	A

(C-443) Fig. P. 5.13.9(a) : Design tables for sum and carry outputs

Step 3 : Draw the logic diagram :

- The full adder using 4 : 1 multiplexer is shown in Fig. P. 5.13.9(b).



(C-444) Fig. P. 5.13.9(b) : Implementation of full adder using two 4 : 1 multiplexers

5.13.6 Implementation of Boolean SOP Expression with Don't Care Conditions :

- Sometimes the Boolean expressions are given with don't care conditions.

- We know that the don't care conditions can be treated as logic 1's or 0's.
- Ex. 5.13.10 illustrates the use of multiplexer to implement such equations.

Ex. 5.13.10 : Implement the following Boolean expression using an 8 : 1 multiplexer.

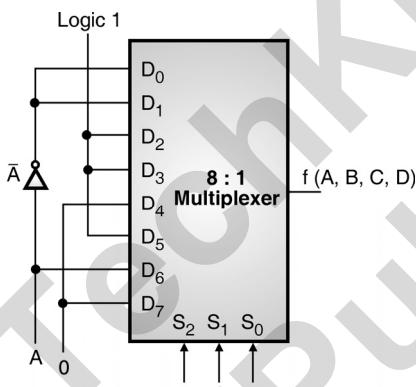
$$f(A, B, C, D) = \sum m(1, 3, 5, 10, 11, 13, 14) + d(0, 2).$$

Soln. :

- The don't care conditions are assumed to be logic 1's. The design table is shown in Fig. P. 5.13.10(a) and the corresponding logic diagram is shown in Fig. P. 5.13.10(b).

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{A}	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to Mux	\bar{A}	\bar{A}	1	1	0	1	A	0

(a) Design table



(b) Implementation

(C-445) Fig. P. 5.13.10

Ex. 5.13.11 : Implement the function $f(A, B, C) = \sum m(1, 3, 7)$ using the same.

May 12, 8 Marks, May 19, 6 Marks

Soln. :

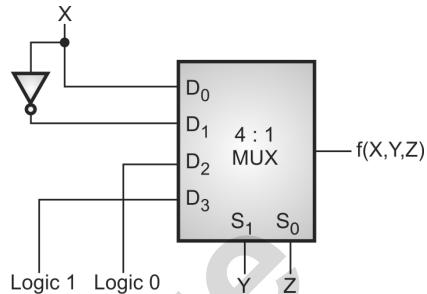
Step 1 : Write the design table :

- The design table is as shown in Fig. P. 5.13.11.

	D ₀	D ₁	D ₂	D ₃
\bar{A}	0	1	2	3
A	4	5	6	7
Input to MUX	X	\bar{X}	0	1

(C-7935) Fig. P. 5.13.11

Step 2 : Implementation using 4:1 MUX :



(C-7936) Fig. P. 5.13.11(a) : Implementation using 4:1 MUX

Ex. 5.13.12 : Implement the following expression using 8 : 1 multiplexer : $f(A, B, C, D) = \sum m(2, 4, 6, 7, 9, 10, 11, 12, 15)$

Dec. 12, 8 Marks

Soln. :

$$f(A, B, C, D) = \sum m(2, 4, 6, 7, 9, 10, 11, 12, 15)$$

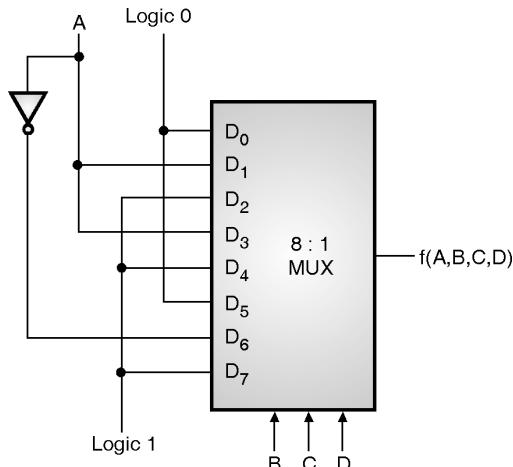
Step 1 : Write the design table :

(C-3599) Table P. 5.13.12

Inputs	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{A}	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	0	A	1	A	1	0	\bar{A}	1

Step 2 : Implementation using 8 : 1 MUX :

- The implementation is as shown in Fig. P. 5.13.12.



(C-3600) Fig. P. 5.13.12 : Implementation

Ex. 5.13.13 : Implement the following Boolean function using single 8 : 1 multiplexer :

$$f(A, B, C, D) = \sum m(1, 4, 6, 9, 13)$$

May 17, 6 Marks

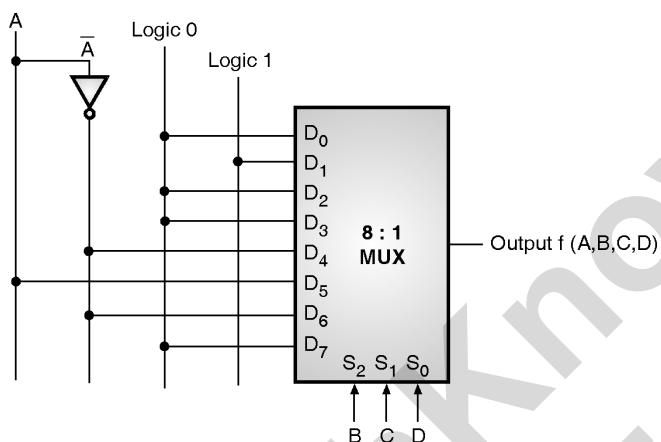
Soln. :

Step 1 : Write the design table :

(C-5905) Table P. 5.13.13 : Design table

Inputs	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
Ā	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
Input to MUX	0	1	0	0	Ā	A	Ā	0

Step 2 : Implementation using 8 : 1 multiplexer :



(C-5906) Fig. P. 5.13.13 : Implementation using 8 : 1 MUX

Ex. 5.13.14 : Implement given function using 8 : 1 MUX and logic gates :

$$F(A, B, C, D) = \sum m(0, 1, 3, 4, 8, 9, 15).$$

Dec. 18, 6 Marks

Soln. :

$$\text{Given : } F(A, B, C, D) = \sum m(0, 1, 3, 4, 8, 9, 15)$$

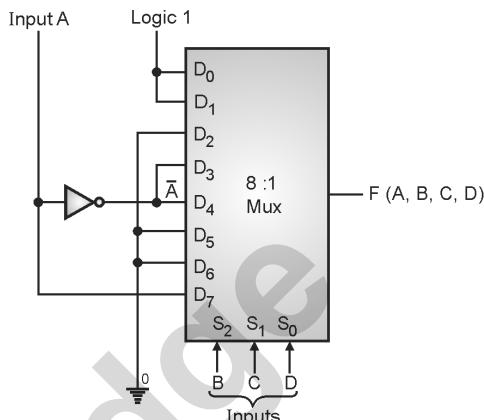
To do : Implementation using 8 : 1 MUX.

Step 1 : Write the design table :

(C-7438) Table P. 5.13.14

Input	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
Ā	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
I/p to MUX	1	1	0	Ā	Ā	0	0	A

Step 2 : Implementation :



(C-7439) Fig. P. 5.13.14

5.14 Demultiplexers :

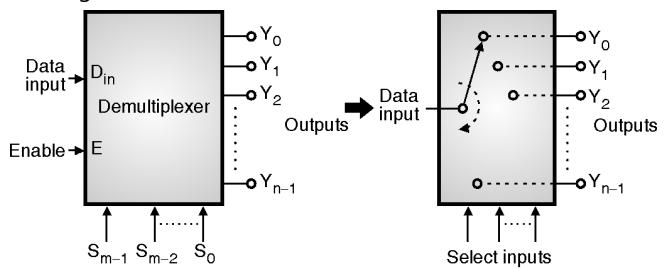
5.14.1 Demultiplexer Principle :

Definition :

- A de-multiplexer is a combinational circuit with one data input, n outputs and m select inputs, which selects one of the n data outputs and routes (connects) it to the input.
- The selection of one of the n outputs is done with the help of the select inputs.

Block diagram :

- The block diagram of a 1 to n demultiplexer is shown in Fig. 5.14.1(a) and its equivalent circuit is as shown in Fig. 5.14.1(b).



(a) 1 : n demultiplexer

(b) Equivalent circuit

(C-447) Fig. 5.14.1

- It has one data input, n outputs, m select inputs and one enable input as shown.
- A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- At a time only one output line is selected by the select lines and the input is connected to the selected output line.

- The enable input will enable the demultiplexer. If the enable (E) input is not active, then the demultiplexer does not work.
- The relation between the n output lines and m select lines is as follows :

$$n = 2^m$$

Equivalent Circuit :

- A demultiplexer is equivalent to a single pole multiple way switch as shown in Fig. 5.14.1(b).
- As shown in Fig. 5.14.1(b) a demultiplexer is equivalent to a digitally controlled single pole, multiple way switch.
- The data input gets connected to only one of the n outputs at given instant of time via the single pole multiple throw rotary switch as shown.
- The position of the rotary arm will be dependent on the status of the m select inputs. It will connect the input to the selected output.

Need of Demultiplexers :

- In most of the electronic systems, the digital data from more than one sources is added together and this common signal is transmitted over a common communication channel.
- It is necessary to separate out this data at the receiving end.
- Under such circumstances we require a circuit which separates the multiple signals from the common one.
- This circuit is nothing else but a demultiplexer, which has one input, many outputs, and some select inputs.
- Demultiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

5.15 Types of Demultiplexers :

- Similar to the multiplexers, the demultiplexers are classified as follows :

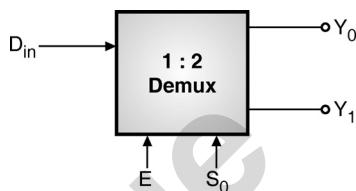
 - 1 : 2 demultiplexer.
 - 1 : 4 demultiplexer.
 - 1 : 8 demultiplexer.
 - 1 : 16 demultiplexer.

5.15.1 1 : 2 Demultiplexer :

Block diagram :

- The block diagram of 1 : 2 demultiplexer is shown in Fig. 5.15.1. It has one data input D_{in} , one select input S_0 , one enable (E) input and two outputs Y_0 and Y_1 .

- D_{in} is connected to Y_0 if $S_0 = 0$ and $E = 1$. Similarly D_{in} is connected to Y_1 if $S_0 = 1$ and $E = 1$.
- If $E = 0$, then both the outputs will be 0 irrespective of the inputs, because the DEMUX is disabled.



(C-448) Fig. 5.15.1 : A 1:2 demultiplexer

Truth Table :

- The truth table of the 1 : 2 demultiplexer is as follows :

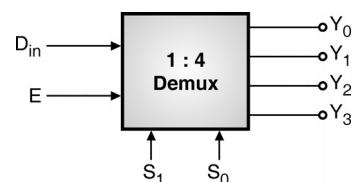
(C-7620) Table 5.15.1 : Truth table of demux 1 : 2

Enable	Select	Outputs	
		Y_1	Y_0
0	X	0	0
1	0	0	D_{in}
1	1	D_{in}	0

5.15.2 1 : 4 Demultiplexer :

Block diagram and truth table :

- The 1 : 4 demultiplexer is shown in Fig. 5.15.2 and its truth table is given in Table 5.15.2.



(C-451) Fig. 5.15.2 : Block diagram of 1 : 4 demultiplexer

(C-6189) Table 5.15.2 : Truth table for 1 : 4 demultiplexer

E	D_{in}	Inputs		Outputs			
		S_1	S_0	Y_0	Y_1	Y_2	Y_3
1	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0
1	0	0	1	0	0	0	0
1	1	0	1	0	1	0	0
1	0	1	0	0	0	0	0
1	1	1	0	0	0	1	0
1	0	1	1	0	0	0	0
1	1	1	1	0	0	0	1

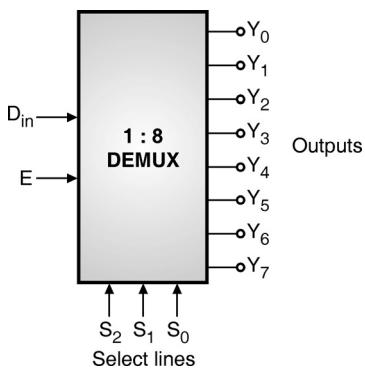
- D_{in} is connected to Y_0 when $S_1S_0 = 00$, it is connected to Y_1 when $S_1S_0 = 01$ and so on. The other outputs will remain zero.

- The enable input needs to be high in order to enable the demux. If $E = 0$ then all the outputs will be low irrespective of everything.

5.15.3 1 : 8 Demultiplexer :

Block diagram and truth table :

- The block diagram of 1 : 8 demux is shown in Fig. 5.15.3(a).



(C-454) Fig. 5.15.3(a) : 1 : 8 demux

- It has one data input, eight outputs, three select inputs and an enable input E.
- The truth table is shown in Fig. 5.15.3(b).

Enable	Select			Outputs								
	E	S ₂	S ₁	S ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	X	X	X	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	D _{in}
1	0	0	1	0	0	0	0	0	0	0	0	D _{in}
1	0	1	0	0	0	0	0	0	0	D _{in}	0	0
1	0	1	1	0	0	0	0	0	D _{in}	0	0	0
1	1	0	0	0	0	0	D _{in}	0	0	0	0	0
1	1	0	1	0	0	D _{in}	0	0	0	0	0	0
1	1	1	0	0	D _{in}	0	0	0	0	0	0	0
1	1	1	1	D _{in}	0	0	0	0	0	0	0	0

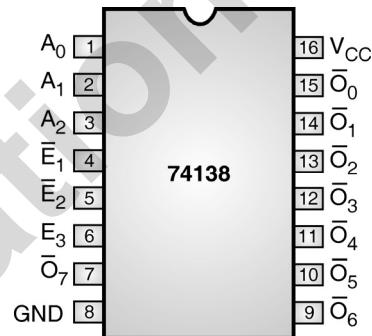
(C-8103) Fig. 5.15.3(b) : Truth table for 1 : 8 demux

- Depending on the combination of the select inputs S₂ S₁ S₀, the data input D_{in} is connected to one of the eight outputs.
- For example if S₂ S₁ S₀ = 1 1 0 then D_{in} is connected to output Y₆.

5.15.4 IC 74138 as 1 : 8 DE-MUX :

- The pin configuration of the 3 : 8 decoder IC 74138 is shown in Fig. 5.15.4.

- A₀, A₁, A₂ are the three address lines or the three input lines. We have to apply the 3-bit binary data to these inputs.
- But when this IC is to be used as a 1 : 8 DEMUX we have to use these lines as the three select lines.
- \bar{O}_0 to \bar{O}_7 are the 8-output active low lines.
- There are three enable inputs out of which \bar{E}_1 and \bar{E}_2 are the active low enable inputs whereas E₃ is an active high enable input.
- We have to make $\bar{E}_1 = \bar{E}_2 = 0$ and E₃ = 1 in order to enable the IC.



(C-489) Fig. 5.15.4(a) : Pin configuration of IC 74138

Pin names	Description
A ₀ – A ₂	Address inputs (Select lines)
\bar{E}_1 – \bar{E}_2	Enable inputs (Active Low)
E ₃	Enable Input (Active HIGH)
\bar{O}_0 – \bar{O}_7	Outputs (Active Low)

Fig. 5.15.4(b) : Pin names and description

Features of IC 74138 :

- Schottky process for achieving a high speed.
- It can work as a decoder or as a demultiplexer.
- Multiple enable inputs ensure easy expansion.
- Active low, mutually exclusive outputs.
- The truth table for IC 74138 is shown in Table 5.15.3.



(C-6190) Table 5.15.3 : Truth table of 74138

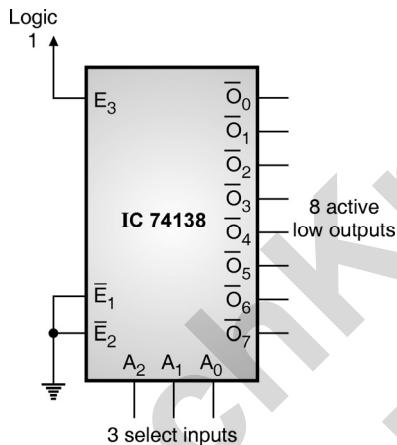
Inputs						Outputs							
E_1	\bar{E}_2	E_3	A_0	A_1	A_2	\bar{O}_0	\bar{O}_1	O_2	\bar{O}_3	\bar{O}_4	\bar{O}_5	\bar{O}_6	\bar{O}_7
H	x	x	x	x	x	H	H	H	H	H	H	H	H
x	H	x	x	x	x	H	H	H	H	H	H	H	H
x	x	L	x	x	x	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	L	H	H	H	H	H
L	L	H	L	L	H	H	H	H	L	H	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

Decoder is disabled
Decoder is enabled

A_0 = LSB, A_2 = MSB, H = HIGH Voltage Level,

L = LOW Voltage Level x = Don't care condition.

- The connection diagram of IC 74138 as 1:8 DEMUX is as shown in Fig. 5.15.5.



(C-3580) Fig. 5.15.5 : IC74138 as 1:8 DEMUX

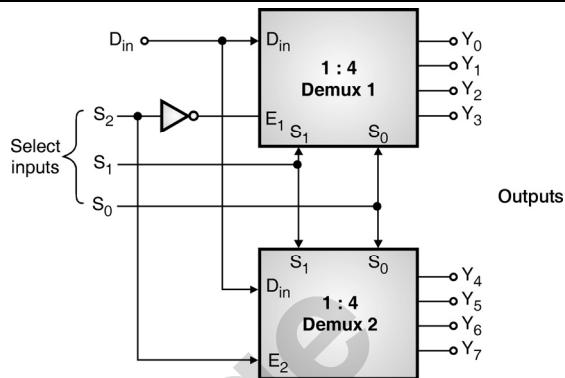
5.16 Demultiplexer Tree :

- Similar to multiplexer we can construct the demultiplexer having more number of lines using demultiplexers having less number lines.
- This is called as demultiplexer tree. It is also called as cascading of demultiplexers.
- This concept will be clear by solving the following examples.

Ex. 5.16.1 : Obtain a 1 : 8 line demultiplexer using two 1 : 4 line demultiplexers.

Soln. :

- The 1 : 8 demultiplexer using two 1 : 4 demultiplexers is shown in Fig. P. 5.16.1. This is similar to cascading of multiplexers.



(C-460) Fig. P. 5.16.1 : 1 : 8 demultiplexer using two 1 : 4 demultiplexers

- The select lines S_1 and S_0 of the two 1 : 4 demultiplexers are connected in parallel with each other and S_2 is used for selecting one of the two 1 : 4 demultiplexers.
- S_2 is connected directly to enable (E) input of Demux – 2 whereas inverted S_2 is connected to the enable input of demux – 1.
- The truth table of this circuit is shown in Table P. 5.16.1.

(C-6192) Table P. 5.16.1 : Truth table of 1 : 8 demultiplexer using two 1 : 4 demultiplexers

Select inputs			Outputs							
S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	D_{in}	0	0	0	0	0	0	0
0	0	1	0	D_{in}	0	0	0	0	0	0
0	1	0	0	0	D_{in}	0	0	0	0	0
0	1	1	0	0	0	D_{in}	0	0	0	0
1	0	0	0	0	0	0	D_{in}	0	0	0
1	0	1	0	0	0	0	0	D_{in}	0	0
1	1	0	0	0	0	0	0	0	D_{in}	0
1	1	1	0	0	0	0	0	0	0	D_{in}

Demux-1 enabled
Demux-2 enabled

5.16.1 Use of DEMUX in Combinational Logic Design :

- Like multiplexers, we can use the demultiplexers for designing the combinational circuits.
- Demultiplexers ICs for 1:2, 1:4, 1:8 and 1:16 demultiplexers are available.
- We can use them to implement the given Boolean expressions representing a combinational circuits.
- Thus it is possible to design and implement many combinational circuits by using demultiplexer and a few logic gates.

**Advantages :**

- Advantages of using demultiplexers for logic design are as follows :
 1. Logic design is simplified.
 2. It is not necessary to simplify the logic expression.
 3. It minimizes the number of ICs required to be used.

Use of DEMUX for combinational circuit design :

1. A truth table or logic expression is given to us.
2. We have to follow the design procedure given below to use DEMUX for implementing the given logical expression.

Design procedure :

Step 1 : Identify the decimal number corresponding to each minterm in the given expression as illustrated below.

$$\text{If } Y = \underbrace{\overline{ABC}}_0 + \underbrace{\overline{ABC}}_5 + \underbrace{ABC}_7 \quad (\text{C-6264})$$

Step 2 : The input data line of the demultiplexer is connected to a logic 1. The outputs corresponding to these numbers (0, 5 and 7) are connected as inputs to a suitable multi input OR gate.

Step 3 : The inputs (A, B, C) of the combinational circuit being designed are connected to the select inputs.

- The following example will demonstrate this concept.

Ex. 5.16.2 : Implement a full adder using demultiplexer.

Dec. 09, Dec. 10, 4 Marks, May 10, 3 Marks

Soln. :

Step 1 : Write the truth table of the full adder :

(C-7621) **Table P. 5.16.2 : Truth table of a full adder**

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 2 : Write expressions for sum and carry :

$$\text{Sum} = \Sigma m(1, 2, 4, 7)$$

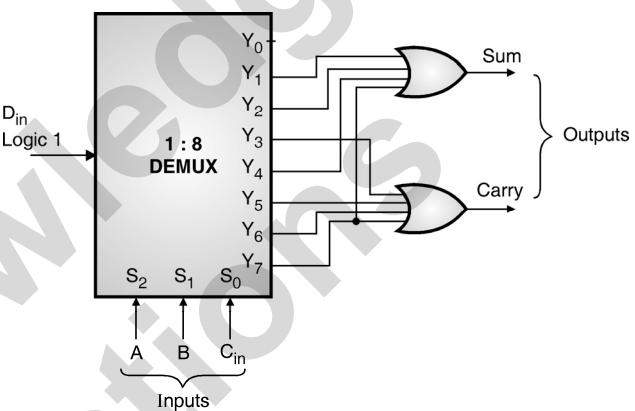
∴ Outputs Y_1, Y_2, Y_4 and Y_7 should be Ored to get sum output.

$$\text{Carry} = \Sigma m(3, 5, 6, 7)$$

∴ Outputs Y_3, Y_5, Y_6 and Y_7 should be Ored to get the carry output.

Step 3 : Implementation using a 1 : 8 demultiplexer :

- Fig. P. 5.16.2 shows the implementation.



(C-465) **Fig. P. 5.16.2 : Full adder using a demultiplexer**

Note : We can use IC 74138 which is an 1 : 8 demultiplexer to implement Boolean equations using 1:8 demultiplexer..

Ex. 5.16.3 : Implement the full subtractor using a 1 : 8 demultiplexer.

Dec. 09, Dec. 10, 4 Marks, May 10, 3 Marks

Soln. :

Step 1 : Write the truth table of the full subtractor :

- The truth table of a full subtractor is as follows :

(C-7461) **Table P. 5.16.3 : Truth table of a full subtractor**

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Step 2 : Write expressions for sum and carry :

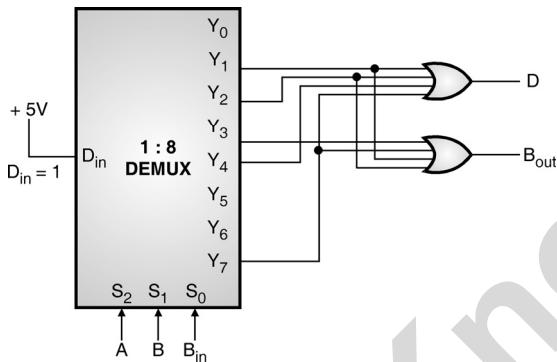
- From the truth table we can express the difference and borrow out outputs in the standard SOP forms as,

$$D = f(A, B, B_{in}) = \sum m(1, 2, 4, 7)$$

$$\text{and } B_{out} = f(A, B, B_{in}) = \sum m(1, 2, 3, 7)$$

Step 3 : Implementation using a 1 : 8 demultiplexer :

- Connect D_{in} to logic 1 permanently and connect A, B and B_{in} to the select inputs S_2, S_1, S_0 respectively as shown in Fig. P. 5.16.3.



(C-464) Fig. P. 5.16.3 : Full subtractor using 1 : 8 demux

- As D_{in} is connected to 1, we get the required minterms at the Demux outputs.
- We have to OR the required minterms to obtain the D and B_{out} outputs as shown in Fig. P. 5.16.3.

Note : We can use IC 74138 as 1 : 8 DEMUX to implement the Boolean expression with 1 : 8 DEMUX.

Ex. 5.16.4 : Implement the following functions using demultiplexer :

$$f_1(A, B, C) = \sum m(0, 3, 7)$$

$$f_2(A, B, C) = \sum m(1, 2, 5)$$

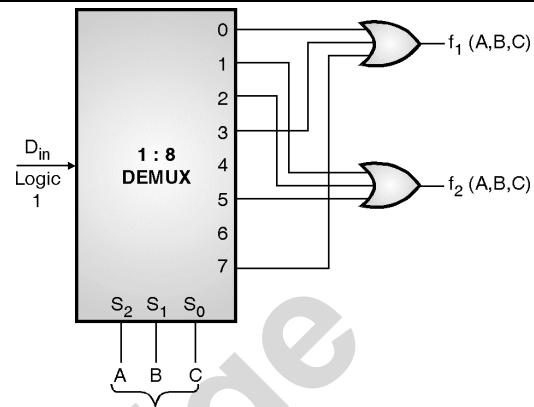
Soln. :

Step 1 : Select the DEMUX :

- We have to use a 1 : 8 demux because there are 8 inputs.

Step 2 : Implement the circuit :

- Fig. P. 5.16.4 shows the implementation.



(C-1323) Fig. P. 5.16.4

Ex. 5.16.5 : Implement two bit comparator using 1 : 16 demultiplexer (active low output). Draw the truth table of two bit comparator and explain the design in steps.

Dec. 11, 8 Marks

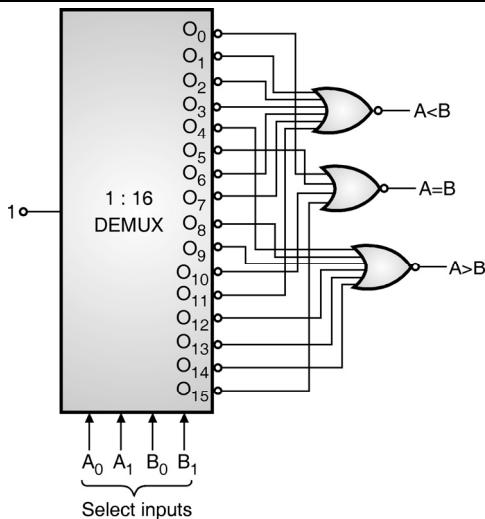
Soln. :

- For a 2-bit comparator, each input word is 2 bit long.
- The truth table of a 2-bit comparator is shown in Table P. 5.16.5.

(C-406(a)) Table P. 5.16.5 : Truth table for a 2-bit comparator

Inputs				Outputs		
A ₁	A ₀	B ₁	B ₀	A < B	A = B	A > B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

- Implementation using 1 : 16 demultiplexer is as shown in Fig. P. 5.16.5.



(C-1969) Fig. P. 5.16.5 : 2 bit comparator using 1 : 16 DEMUX

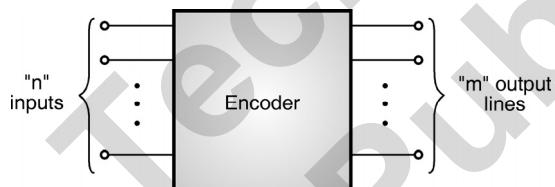
5.17 Encoders :

Definition :

- Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder.
- An encoder has "n" number of input lines and "m" number of output lines.
- An encoder produces an m bit binary code corresponding to the n bit digital number, applied at its input.

Block diagram :

- Block diagram of an encoder is shown in Fig. 5.17.1.



(C-466) Fig. 5.17.1 : Block diagram of an encoder

- The encoder accepts an n input digital word and converts it into an m bit another digital word.
- For example a BCD number applied at the input can be converted into a binary number at the output.
- The internal combinational circuit of the encoder is designed accordingly.

5.17.1 Types of Encoders :

- The types of encoders which we are going to discuss are as follows :
 1. Priority encoders.
 2. Decimal to BCD encoder.
 3. Octal to binary encoder.

- 4. Hexadecimal to binary encoder.

5.18 Priority Encoder :

SPPU : Dec. 08

University Questions

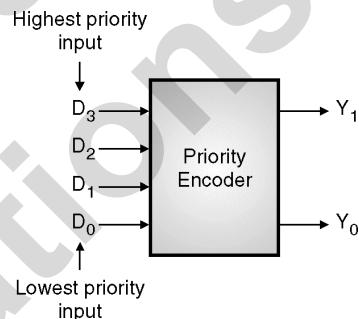
Q. 1 What is priority encoder ? (Dec. 08, 2 Marks)

Definition :

- This is a special type of encoder with priorities assigned to all its input lines. If two or more input lines are "1" at the same time, then the input line with highest priority will be considered.

Block diagram and truth table :

- The block diagram of a priority encoder is shown in Fig. 5.18.1(a) and its truth table is shown in Fig. 5.18.1(b).



(a) Block diagram of a priority encoder

Highest	Inputs			Lowest	Outputs	
D ₃	D ₂	D ₁	D ₀		Y ₁	Y ₀
0	0	0	0	0	X	X
0	0	0	1	1	0	0
0	0	1	X	X	0	1
0	1	X	X	X	1	0
1	X	X	X	X	1	1

(b) Truth table of a priority encoder

(C-467) Fig. 5.18.1

- Priorities are given to the input lines. If two or more input lines are "1" at the same time, then the input line with highest priority will be considered.
- There are four inputs, D₀ through D₃ and two outputs Y₁ and Y₀. Out of the four inputs D₃ has the highest priority and D₀ has the lowest priority.
- That means if D₃ = 1 then Y₁ Y₀ = 11 irrespective of the other inputs. Similarly if D₃ = 0 and D₂ = 1 then Y₁ Y₀ = 10 irrespective of the other inputs.
- Carefully go through the truth table shown in Fig. 5.18.1(b) to get the feel of priority encoder operation.

5.18.1 Priority Encoders in the IC Form :

- The priority encoders are available in the integrated circuit form. The available encoders are :

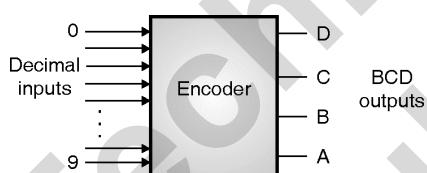
74147	10 : 4 Priority encoder (Decimal to BCD)
74148	Octal to binary priority encoder
- 74147 is basically a decimal to BCD priority encoder.

5.18.2 Decimal to BCD Encoder :

- The block diagram of decimal to BCD encoder is shown in Fig. 5.18.2.
- The truth table for a decimal to BCD encoder is as given in Table 5.18.1.

(C-470(a)) **Table 5.18.1 : Truth table for decimal to BCD encoder**

Input	Output			
	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



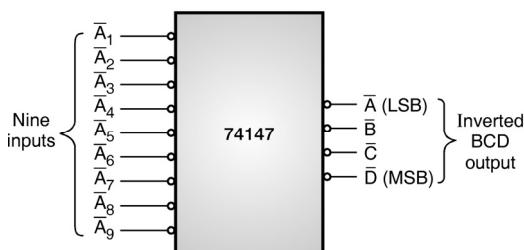
(C-470) **Fig. 5.18.2 : Block diagram of decimal to BCD encoder**

5.18.3 Decimal to BCD Encoder MSI IC 74147 :

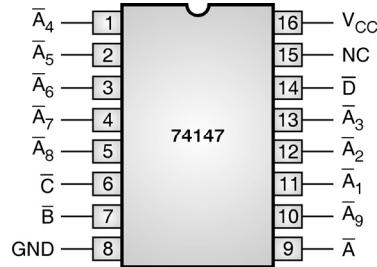
- IC 74147 is basically a 10:4 encoder or Decimal to BCD encoder.

Pin configuration and logic symbol :

- Fig. 5.18.3(b) shows the pin configuration and Fig. 5.18.3(a) shows the logic symbol of IC 74147.



(a) **Logic symbol of IC 74147**



(b) **Pin configuration of 74147**

(C-472) **Fig. 5.18.3**

- A_1 to A_9 are the active low inputs and A , B , C , D are the active low outputs. Therefore bubbles have been added in Fig. 5.18.3(a).

Truth table :

- Fig. 5.18.3(c) shows the truth table of decimal to BCD encoder IC 74147.
- A_1 to A_9 are the inputs with A_1 having the lowest priority and A_9 having the highest priority.
- From truth table we conclude that all nine inputs are ACTIVE LOW representing decimal digit from 1 to 9. In response to input, chip produces inverted BCD code corresponding to highest numbered ACTIVE INPUT.

Inputs									Outputs (Inverted BCD)				Normal BCD				
\bar{A}_1	\bar{A}_2	\bar{A}_3	\bar{A}_4	\bar{A}_5	\bar{A}_6	\bar{A}_7	\bar{A}_8	\bar{A}_9	D	C	B	A	\bar{D}	\bar{C}	\bar{B}	\bar{A}	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
X	X	X	X	X	X	X	X	X	0	0	1	1	0	1	0	0	1
X	X	X	X	X	X	X	X	0	1	0	1	1	1	1	0	0	0
X	X	X	X	X	X	X	0	1	1	1	0	0	0	0	1	1	1
X	X	X	X	X	X	0	1	1	1	1	0	0	0	1	0	1	1
X	X	X	X	X	0	1	1	1	1	1	0	0	1	0	1	1	0
X	X	X	X	0	1	1	1	1	1	1	0	1	0	0	1	0	1
X	X	X	0	1	1	1	1	1	1	1	0	1	1	0	1	0	0
X	X	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1
X	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1

(C-8309) **Fig. 5.18.3(c) : Truth table for 74147**

- When all inputs are held high, output $\bar{D} \bar{C} \bar{B} \bar{A} = 1111$ i.e. $DCBA = (0000)_2 = (0)_{10}$. Thus a decimal 0 is represented.
- The truth table also shows the normal BCD output which is actually the inversion of the output of IC.
- As A_9 is the highest priority input, if $\bar{A}_9 = 0$ then the remaining input lines are treated as don't care and the inverted BCD output is produced as $\bar{D} \bar{C} \bar{B} \bar{A} = 0110$. The same logic is applicable to the other inputs.

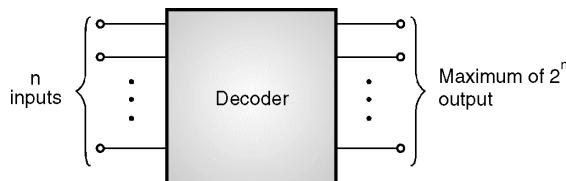
5.19 Decoder :

Definition :

- A decoder is a combinational circuit with "n" inputs and to a maximum 2^n outputs that are related to each other with a certain rule.

Block diagram :

- Fig. 5.19.1 shows the block diagram of a decoder. It has "n" inputs and to a maximum 2^n outputs.



(C-479) Fig. 5.19.1 : Block diagram of a decoder

- Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

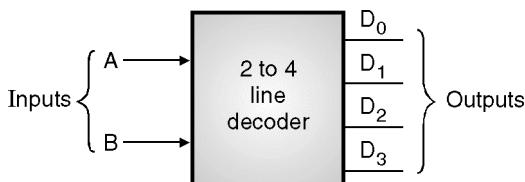
Typical applications :

1. Code converters.
2. BCD to seven segment decoders.
3. Nixie tube decoders.
4. Relay actuators.

5.19.1 2 to 4 Line Decoder :

Block diagram :

- The block diagram of a 2 to 4 line decoder is shown in Fig. 5.19.2. A and B are the two inputs whereas D_0 through D_3 are the four outputs.



(C-480) Fig. 5.19.2 : Block diagram of a 2 line to 4 line decoder

Truth table :

- Table 5.19.1 shows the truth table which explains the operation of the decoder. It shows that each output is "1" for only a specific combination of inputs.

(C-480(a)) Table 5.19.1 : Truth table of a 2 line to 4 line decoder

Inputs		Outputs			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

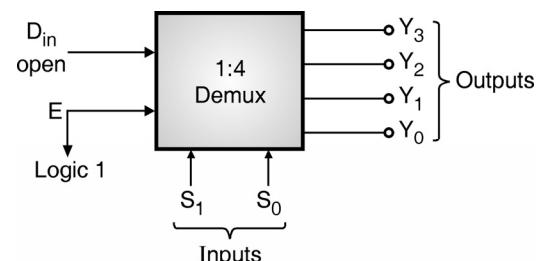
- The Boolean expressions for the four outputs are,

$$D_0 = \bar{A} \bar{B} \quad D_1 = \bar{A} B$$

$$D_2 = A \bar{B} \quad \text{and} \quad D_3 = A B$$

5.19.2 Demultiplexer as Decoder :

- We can use a demultiplexer as a decoder.
- Let us see how to operate a 1 : 4 demux as 2 : 4 decoder.
- Consider Fig. 5.19.3 which shows a 1 : 4 demultiplexer.
- D_{in} is the data input, S_1 S_0 are the select lines and Y_3 through Y_0 are the outputs.
- In order to operate it as 2 : 4 decoder, we have to use S_1 S_0 as inputs, keep D_{in} open and use Y_3 to Y_0 as outputs as shown in Fig. 5.19.3.

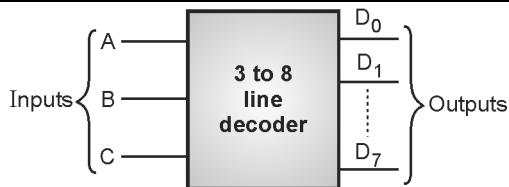


(C-483) Fig. 5.19.3 : 1 : 4 demux as 2 : 4 decoder

5.19.3 3 to 8 Line Decoder :

Block diagram :

- The block diagram of a 3 to 8 line decoder is shown in Fig. 5.19.4.
- A, B and C are the three inputs whereas D_0 through D_7 are the eight outputs.



(C-7773) Fig. 5.19.4 : Block diagram of a 3 line to 8 line decoder

Truth table :

- The truth table of 3 to 8 line decoder is shown in Table 5.19.2.

(C-6477) Table 5.19.2 : Truth table for 3 to 8 line decoder

Inputs			Outputs							
A	B	C	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D 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

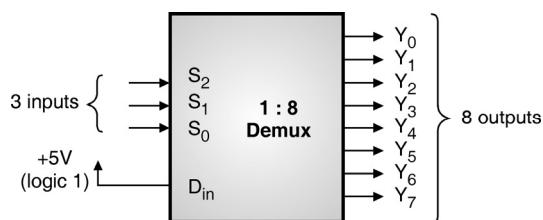
$$\begin{aligned}D_0 &= \bar{A}\bar{B}\bar{C} \\D_1 &= \bar{A}\bar{B}C \\D_2 &= \bar{A}B\bar{C} \\D_3 &= \bar{A}BC \\D_4 &= AB\bar{C} \\D_5 &= A\bar{B}\bar{C} \\D_6 &= A\bar{B}C \\D_7 &= ABC\end{aligned}$$

Boolean expressions :

- The Boolean expressions for the eight outputs in terms of the three inputs are as given in Table 5.19.2.

5.19.4 1 : 8 DEMUX as 3:8 Decoder :

- In order to operate 1 : 8 Demux as a 3:8 line decoder, the connections are to be made as shown in Fig. 5.19.5.
- The data input D_{in} is connected to logic 1 permanently. The three select inputs S₀, S₁ and S₂ will act as three input lines of the decoder and Y₀ to Y₇ are the 8-output lines.



(C-485) Fig. 5.19.5 : Use of 1:8 demux as 3:8 decoder

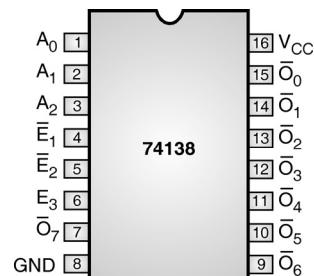
- The truth table of this circuit is as given in Table 5.19.3 which shows that the circuit works as a 3 : 8 decoder.

(C-485(a)) Table 5.19.3 : Truth table

D _{in}	S ₂	S ₁	S ₀	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

5.19.5 IC 74138 / IC 74238 as 3 : 8 Decoder :

- The pin configuration of the 3 : 8 decoder IC 74138 is shown in Fig. 5.19.6.
- We have already discussed this IC as a demultiplexer. Let us now see how to use it as a decoder.
- A₀, A₁, A₂ are the three address lines or the three input lines. We have to apply the 3-bit binary data to these inputs. So these lines act as the 3-data input lines of the 3:8 line decoder.
- ̄O₀ to ̄O₇ are the 8-output active low lines. These lines act as 8-data output lines of the 3 : 8 line decoder.
- There are three enable inputs out of which ̄E₁ and ̄E₂ are the active low enable inputs whereas E₃ is an active high enable input.
- We have to make ̄E₁ = ̄E₂ = 0 and E₃ = 1 in order to enable the IC.



(a) Pin configuration of IC 74138

Pin names	Description
A ₀ – A ₂	Address inputs (Select lines)
̄E ₁ – ̄E ₂	Enable inputs (Active Low)



Pin names	Description
E_3	Enable Input (Active HIGH)
$\bar{O}_0 - \bar{O}_7$	Outputs (Active Low)

(b) Pin names and description
(C-489) Fig. 5.19.6

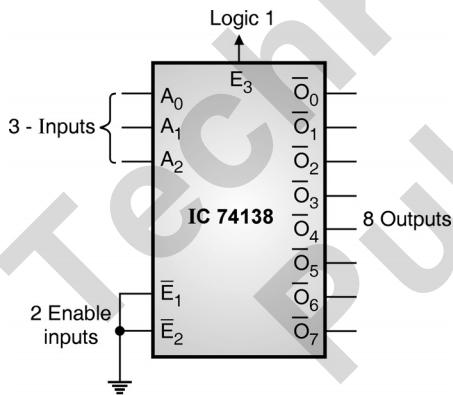
- The truth table for IC 74138 is shown in Table 5.19.4.

(C-6190) Table 5.19.4 : Truth table of 74138

Inputs						Outputs							
\bar{E}_1	\bar{E}_2	E_3	A_0	A_1	A_2	\bar{O}_0	\bar{O}_1	O_2	\bar{O}_3	\bar{O}_4	O_5	\bar{O}_6	\bar{O}_7
H	x	x	x	x	x	H	H	H	H	H	H	H	H
x	H	x	x	x	x	H	H	H	H	H	H	H	H
x	x	L	x	x	x	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	L	H	H	H	H
L	L	H	H	L	H	H	H	H	L	H	H	H	H
L	L	H	L	H	H	H	H	H	H	L	H	H	H
L	L	H	L	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

A_0 = LSB, A_2 = MSB, H = HIGH Voltage Level,
L = LOW Voltage Level, x = Don't care condition

- Fig. 5.19.7 shows the connection diagram for IC 74138 used as a 3 : 8 decoder.



(C-3581) Fig. 5.19.7 : IC 74138 connected as 3:8 decoder

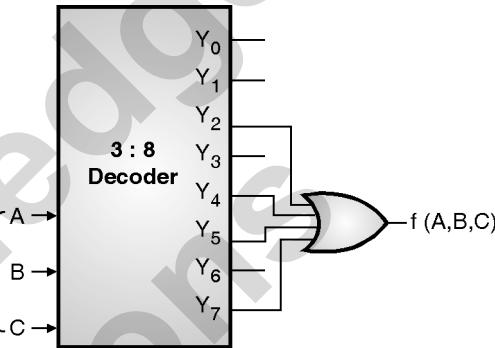
5.19.6 Combinational Logic Design Using Decoders :

- In order to implement a logic function using a decoder, the outputs corresponding to all the minterms will be **NANDED or bubbled ORed** if the decoder outputs are active low type (for example decoder IC 74138 has active low outputs).
- Following examples demonstrate the use of decoder for logic circuit design.

Ex. 5.19.1 : Implement the following Boolean function using a 3 : 8 decoder and external gates.
 $f(A, B, C) = \Sigma(2, 4, 5, 7)$

Soln. :

- The decoder produces minterms. The outputs Y_2, Y_4, Y_5 and Y_7 are ORed to produce the required output.
- The logic implementation of the given logic function is shown in Fig. P. 5.19.1.



(C-486) Fig. P. 5.19.1 : Implementation of Boolean equation using decoder and gate

Ex. 5.19.2 : Implement the following multiple output function using a suitable decoder.

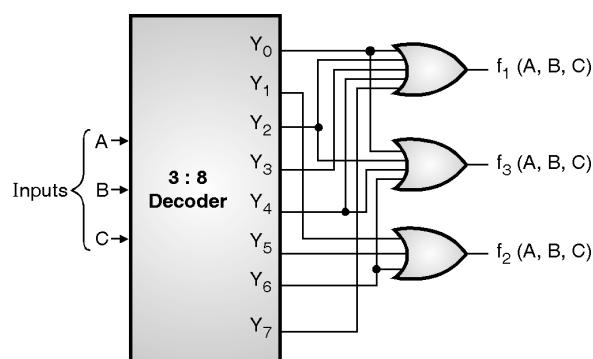
$$f_1(A, B, C) = \Sigma m(0, 4, 7) + d(2, 3),$$

$$f_2(A, B, C) = \Sigma m(1, 5, 6)$$

$$f_3(A, B, C) = \Sigma m(0, 2, 4, 6).$$

Soln. :

- Let us use 3 line to 8 line decoder. f_1 consists of don't care conditions. So we will consider them to be logic 1.
- The implementation of the given three functions using a 3 : 8 decoder and a few OR gates is shown in Fig. P. 5.19.2.



(C-487) Fig. P. 5.19.2 : Implementation of the given Boolean functions using a 3 : 8 decoder and OR gates



Ex. 5.19.3 : Design a full adder using 3 : 8 decoder IC 74138.

Dec. 13, Dec. 16, 6 Marks

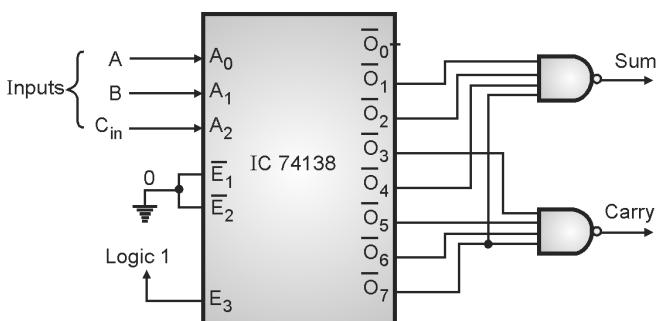
Soln. :

- The truth table of a full adder is shown in Table P. 5.19.3.

(C-7621) **Table P. 5.19.3 : Truth table of full adder**

Inputs			Outputs	
A	B	C _{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- The A, B and C_{in} inputs are applied to the A₀, A₁ and A₂ inputs of the decoder.
- The outputs of 74138 are active low. Hence we should apply \bar{O}_1 , \bar{O}_2 , \bar{O}_4 and \bar{O}_7 to a NAND gate as shown in Fig. P. 5.19.3 to obtain the sum output.
- Similarly outputs \bar{O}_3 , \bar{O}_5 , \bar{O}_6 and \bar{O}_7 to another NAND gate to obtain the carry output.
- Implementation of full adder is shown in Fig. P. 5.19.3.
- All the enable terminals are connected to their respective active levels to enable the IC.



(C-491) **Fig. P. 5.19.3 : Full implemented using IC 74138**

Ex. 5.19.4 : Implement a 3-bit binary to gray code converter using decoder IC 74138.

May 11, 8 Marks

Soln. :

Step 1 : Write the truth table relating the binary and gray codes :

- The truth table is as follows :

(C-8138) **Table P. 5.19.4 : Truth table relating the binary and gray codes**

Decimal	Binary inputs			Gray outputs		
	B ₂	B ₁	B ₀	G ₂	G ₁	G ₀
0	0	0	0	0	0	0
1	0	0	1	1	0	0
2	0	1	0	0	0	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	1	0

Step 2 : Obtain the expressions for G₂, G₁ and G₀:

- Refer to the shaded portions of Table P. 5.19.4. Normally the expressions for G₂, G₁ and G₀ would have been written in the SOP form as,

$$G_2 = O_4 + O_5 + O_6 + O_7$$

$$G_1 = O_2 + O_3 + O_4 + O_5$$

$$G_0 = O_1 + O_2 + O_5 + O_6$$

- But the outputs of IC 74138 are active low. Hence we will have to convert these equations in terms of inverted "O" outputs as follows :

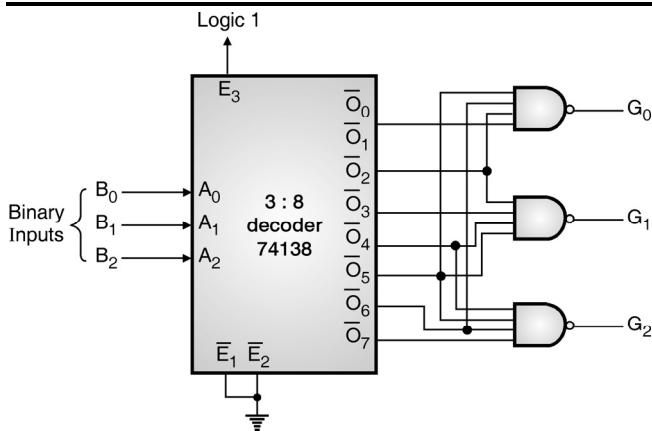
$$G_2 = O_4 + O_5 + O_6 + O_7 = \overline{\overline{O}_4 + \overline{O}_5 + \overline{O}_6 + \overline{O}_7}$$

$$\text{But } \overline{A + B} = \overline{A} \cdot \overline{B}$$

$$\therefore G_2 = \overline{\overline{O}_4 \cdot \overline{O}_5 \cdot \overline{O}_6 \cdot \overline{O}_7} \quad \dots(1)$$

$$\text{Similarly } G_1 = \overline{\overline{O}_2 \cdot \overline{O}_3 \cdot \overline{O}_4 \cdot \overline{O}_5} \quad \dots(2)$$

$$\text{And } G_0 = \overline{\overline{O}_1 \cdot \overline{O}_2 \cdot \overline{O}_5 \cdot \overline{O}_6} \quad \dots(3)$$



(C-492) Fig. P. 5.19.4 : Binary to gray code converter using decoder 74138

- Equations (1), (2) and (3) are implemented as shown in Fig. P. 5.19.4. Note that each equation represents a 4-input NAND gate.

Note : We can use IC 74138 as 3 : 8 decoder in order to implement the Boolean equations using 3 : 8 decoder.

Ex. 5.19.5 : Implement full subtractor using decoder IC 74138 and 2-input NAND gate ICs 7400.

Dec. 15, May 18, 6 Marks

Soln. :

Truth table :

- The truth table for full subtractor is as shown in Table P. 5.19.5.

(C-7461) Table P. 5.19.5 : Truth table for full subtractor

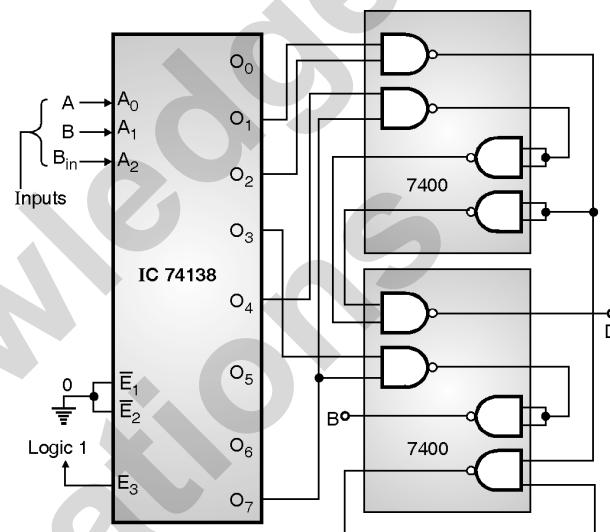
Inputs			Outputs	
A	B	B _{in}	D	B _{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- The A, B, B_{in} inputs are applied to the A₀, A₁ and A₂ inputs of the decoder.
- The outputs of IC 74138 are active low. Hence they should be applied to NAND gates as shown in

Fig. P. 5.19.5 to obtain the difference and borrow outputs.

Logic diagram :

- Implementation of full subtractor is as shown in Fig. P. 5.19.5.
- All the enable terminals are connected to their respective active levels to enable the decoder IC.



(C-1317) Fig. P. 5.19.5

- Implementation is to be done using 2 input NAND gates. Hence the output equations for "Borrow" and "Difference" are modified as,

$$\begin{aligned}
 \text{Difference } D &= \overline{\overline{O_1} \overline{O_2} \overline{O_4} \overline{O_7}} \\
 &= \overline{\overline{(O_1 O_2)} + \overline{(O_4 O_7)}} \\
 &= \overline{\overline{O_1} \overline{O_2}} + \overline{\overline{O_4} \overline{O_7}} \\
 \text{Borrow } B &= \overline{\overline{O_1} \overline{O_2}} \overline{\overline{O_3} \overline{O_7}} \\
 &= \overline{\overline{O_1} \overline{O_2}} + \overline{\overline{O_3} \overline{O_7}} \\
 &= \overline{\overline{O_1} \overline{O_2}} \overline{\overline{O_3} \overline{O_7}}
 \end{aligned}$$

- Ex. 5.19.6 :** Implement Gray to Binary code converter using suitable decoder.

Soln. : Solve it yourself.

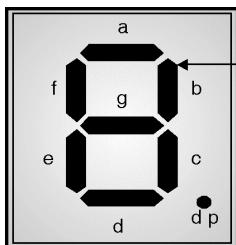
5.19.7 Advantage of Decoder Realization :

- The decoder may not be as efficient as the MUX in realizing a single Boolean function because a NAND gate is required to be used on the output side.
- But the decoder can be preferred over a MUX in realizing multiple Boolean functions simultaneously.

5.20 Case Study : Combinational Logic Design of BCD to 7 Segment Display Controller :

5.20.1 Seven Segment LED Display :

- All of us know what a seven segment display is.
- Fig. 5.20.1 shows the construction of a seven segment display.



Note : Each segment (a to g and dp) is an LED in the shape of the segment

(B-2342) Fig. 5.20.1 : Standard form of seven segment display

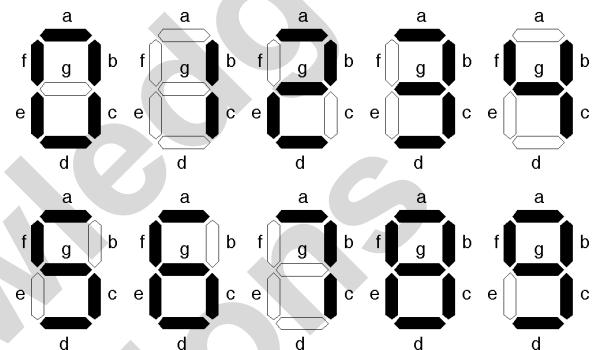
- We can display any number from 0 to 9 by turning on various combinations of the segments as shown in Table 5.20.1.

(B-2343(a)) Table 5.20.1

Segments							Number	Display
a	b	c	d	e	f	g		
ON	ON	ON	ON	ON	ON	ON	—	(0)
—	ON	ON	—	—	—	—	(1)	1
ON	ON	—	ON	ON	—	ON	(2)	2
ON	ON	ON	ON	—	—	ON	(3)	3
—	ON	ON	—	—	ON	ON	(4)	4
ON	—	ON	ON	—	ON	ON	(5)	5
ON	—	ON	ON	ON	ON	ON	(6)	6
ON	ON	ON	—	—	—	—	(7)	7
ON	ON	ON	ON	ON	ON	ON	(8)	8
ON	ON	ON	ON	—	ON	ON	(9)	9

- Actually each segment (a to f) is nothing but an LED in a segment form.

- Their connection leads are brought out, and by applying a forward voltage to the segments to be turned on we can display any number between 0 and 9.
- For example if we want to display the number 7 then the segments a, b and c should be turned on and all other segments should be off.
- Similarly the other numbers can be displayed as shown in Fig. 5.20.2.



(B-2343) Fig. 5.20.2 : Various digits displayed with seven segment display

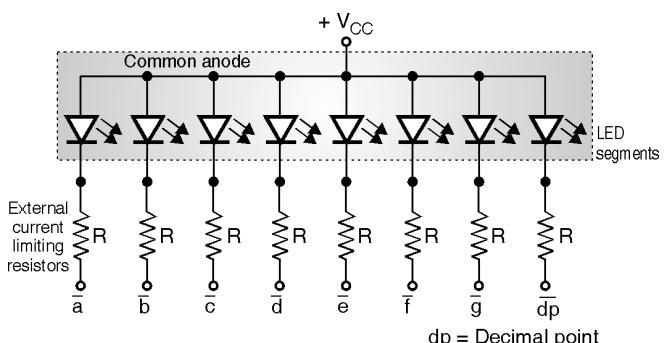
5.20.2 Types of Seven Segment Displays :

There are two types of seven segment LED displays :

1. Common anode 2. Common cathode

5.20.3 Common Anode Display :

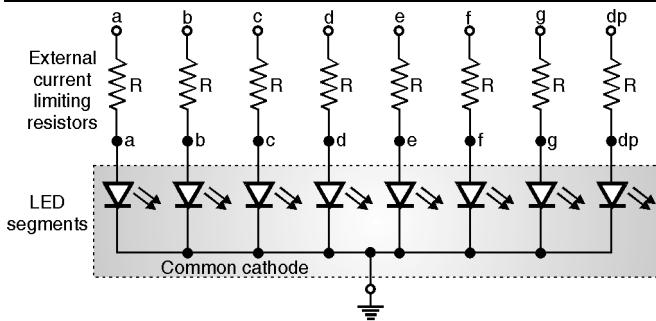
- A common anode seven segment display is as shown in Fig. 5.20.3.
- Here as the name indicates the anode terminals of all LED segments are connected together.



(C-495) Fig. 5.20.3 : Common anode LED display

5.20.4 Common Cathode Display :

- A common cathode seven segment LED display is as shown in Fig. 5.20.4.

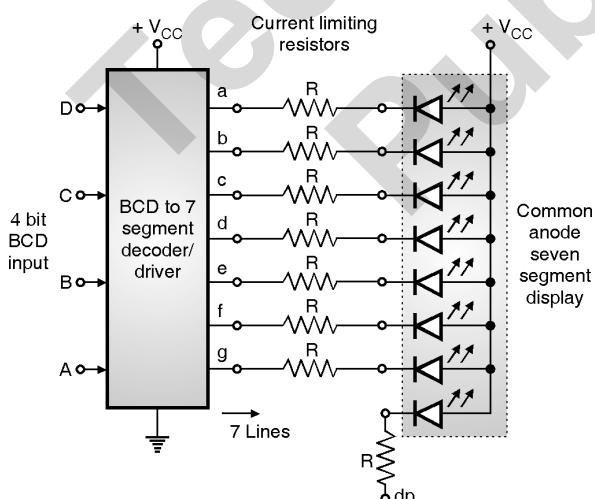


(C-496) Fig. 5.20.4 : Common cathode LED display

- As the name indicates, the cathode terminal is made common and all the anode terminals are brought out separately.
- A current limiting resistor is externally connected in series with each segment.
- The common cathode point is connected to ground and the anodes of the segments to be illuminated are connected to positive supply voltage V_{CC} .

5.20.5 Use of a Decoder for Driving the Seven Segment Display :

- The use of a BCD to seven segment decoder / driver for driving the seven segment display is shown in Fig. 5.20.5.



(C-497) Fig. 5.20.5 : Driving the segment common anode display

- Many times the seven segment LED displays are connected at the output of digital ICs such as counters.

- Now as the counter output is in the BCD (binary coded decimal) form, which has only four lines, it cannot drive the seven segment display directly.
- Therefore we have to use a BCD to seven segment decoder / driver IC (integrated circuit) between the counter output and seven segment display.
- The common anode type display is being used here.
- The decoder accepts a four bit BCD count from a counter, converts it to a seven bit code suitable for the seven segment display (\bar{a} to \bar{g}) and drives the display.
- To turn on a segment the corresponding decoder output goes low, and sinks current (for common anode display).
- A current limiting resistor is connected in series with each segment.

5.20.6 BCD to Seven Segment Display

Driver (Common Anode Display) :

- Let us assume that the type of display is common anode.
- Hence the output of the converter should be "0" if a display segment is to be turned on.

Truth table :

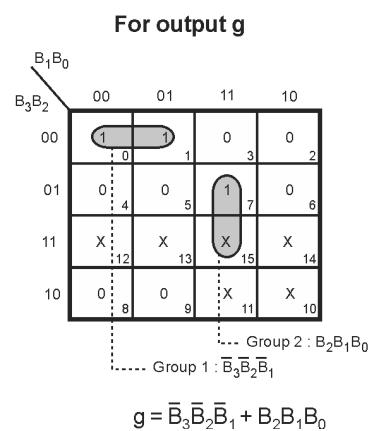
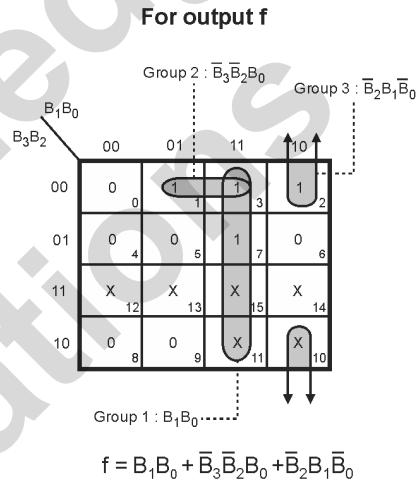
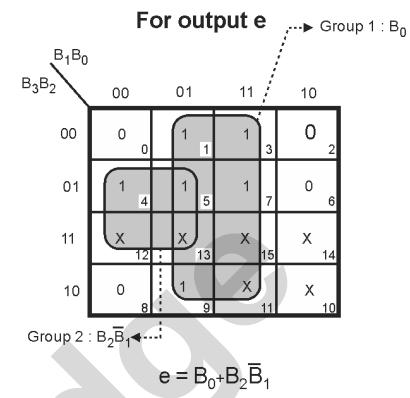
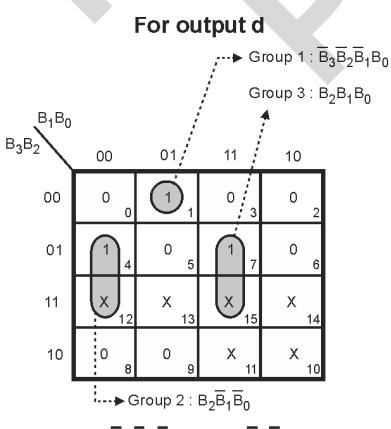
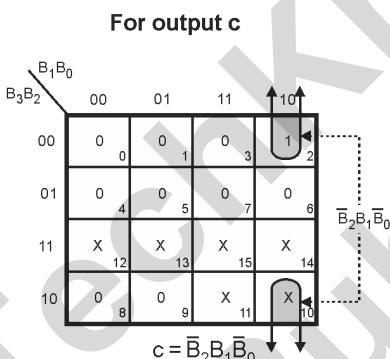
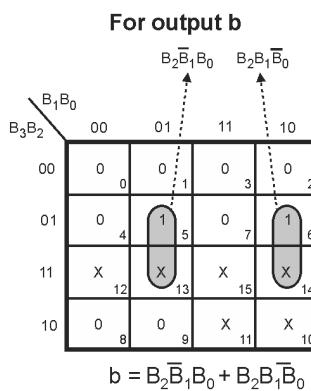
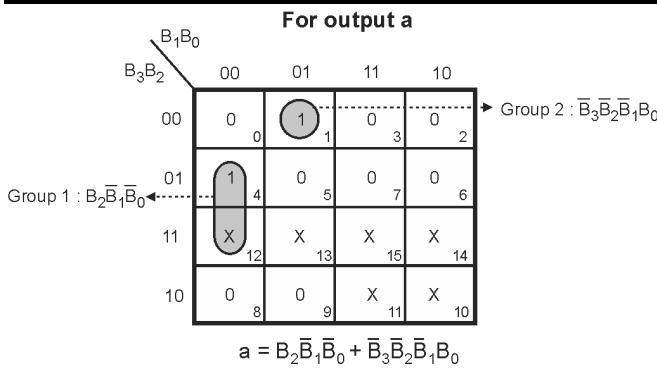
- The truth table is as shown in Table 5.20.2.

(C-499(a)) Table 5.20.2 : Truth table for BCD to 7 segment decoder

Decimal	Inputs				Outputs						
	B ₃	B ₂	B ₁	B ₀	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	0	1	0	0	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

K-maps and simplifications :

- The K maps for the seven outputs are as shown in Fig. 5.20.6.

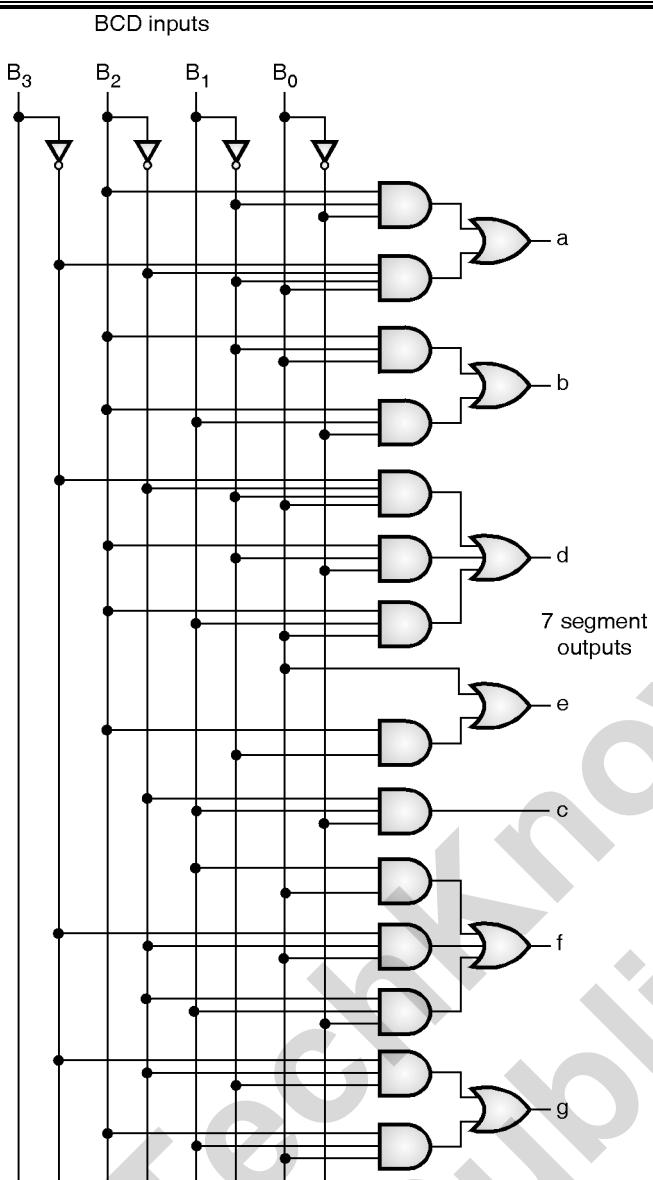


(C-499(b)) Fig. 5.20.6 : K maps for the outputs of BCD to seven segment decoder

Realization :

- The logic diagram for BCD to seven segment decoder is shown in Fig. 5.20.7.

(C-499) Fig. 5.20.6(Contd...)



(C-501) Fig. 5.20.7 : BCD to seven segment decoder

Review Questions

- Q. 1 Explain the working of a half adder ? Draw its logic diagram.
- Q. 2 Draw the logic diagram of full adder and its truth table.
- Q. 3 Give the circuit diagram of a full adder using NAND gate and explain. Give its truth table.
- Q. 4 What is similar between a half adder and a half subtractor ?
- Q. 5 Build a full adder from half adder circuits.
- Q. 6 With suitable block diagram explain the operation of n-bit serial adder.
- Q. 7 Write a note on a 4 bit parallel binary adder.
- Q. 8 Draw pin diagram of IC 7483 and explain its operation as 4-bit binary adder.
- Q. 9 What is half subtractor ? Draw the logic diagram and truth table of half subtractor.
- Q. 10 What is meant by a full subtractor ? Draw a full subtractor circuit.
- Q. 11 Draw the circuit diagram of single digit BCD adder using IC 7483.
- Q. 12 Implement a full subtractor circuit using only NAND gates.
- Q. 13 In detail explain the working of 4-bit binary adder/subtractor using IC 7483.
- Q. 14 For one digit BCD adder :
1. Draw circuit diagram.
 2. Explain its working.
 3. Can $(11)_{10}$ and $(9)_{10}$ be added by this circuit ? Justify.
- Q. 15 Draw half subtractor circuit. Use NAND gates only. Explain its working.
- Q. 16 Draw 4 bit adder/subtractor circuit using IC 7483 and IC 7486. Explain its working.
- Q. 17 Draw the circuit of 4 bit binary parallel adder. Explain its working.
- Q. 18 What is meant by a multiplexer ? Explain with block diagram the principle of multiplexing.
- Q. 19 Explain with diagram and truth table the operation of 4 : 1 Mux.
- Q. 20 Explain briefly with pin-diagram the following ICs IC-74153,
- Q. 21 What is a 'Multiplexer tree' ?



- Q. 22 Give applications of multiplexers.
- Q. 23 Explain with diagram the working of 1 to 8 demultiplexer.
- Q. 24 Explain with diagram the working of 1 to 16 demultiplexer.
- Q. 25 Explain with pin-diagram the following ICs : IC74151, IC 74138.

- Q. 26 Explain how demultiplexer can be used as a decoder.
- Q. 27 What is the necessity of multiplexer ?
- Q. 28 With a neat block diagram explain the function of an encoder.
- Q. 29 What is meant by a priority encoder ? Give example.
- Q. 30 What do you mean by a 'Decoder' ? Give its applications.

□□□

TechKnowledge
Publications

Unit 3

Chapter 6

Flip Flops

Syllabus

Introduction to sequential circuits : Difference between combinational circuits and sequential circuits; Memory element-latch & Flip-Flop.

Flip-Flops : Logic diagram, Truth table & excitation table of SR, JK, D, T flip flops; Conversion from one FF to another, Study of flip flops with regard to asynchronous and synchronous, Preset & clear, Master slave configuration; Study of 7474, 7476 flip flop ICs.

Case study : Use of sequential logic design in a simple traffic light controller.

Chapter Contents

6.1	Introduction	6.11	Master Slave (MS) JK Flip Flop
6.2	Triggering Methods	6.12	Preset and Clear Inputs
6.3	Gated Latches (Level Triggered SR Flip Flop)	6.13	Various Representations of Flip Flops
6.4	The Gated S-R Latch (Level Triggered S-R Flip Flop)	6.14	Excitation Table of Flip-Flop
6.5	The Gated D Latch (Clocked D Flip Flop)	6.15	Conversion of Flip Flops
6.6	Gated JK Latch (Level Triggered JK Flip Flop)	6.16	Applications of Flip Flops
6.7	Edge Triggered Flip Flops	6.17	Study of Flip-Flop ICs
6.8	Edge Triggered D Flip Flop	6.18	Analysis of Clocked Sequential Circuits
6.9	Edge Triggered J-K Flip Flop	6.19	Design of Clocked Synchronous State Machine using State Diagram
6.10	Toggle Flip Flop (T Flip Flop)	6.20	Case Study : Use of Sequential Logic Design in a Simple Traffic Light Controller

6.1 Introduction :

Combinational circuits :

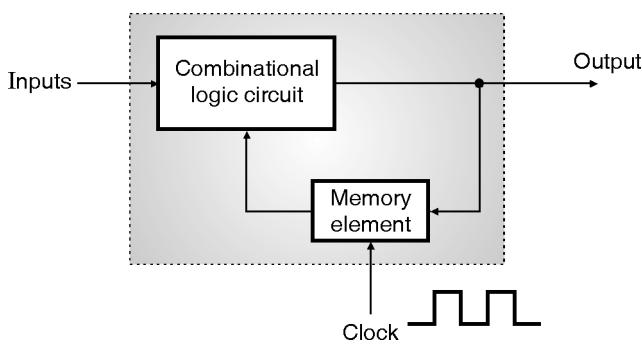
Definition :

- A combinational circuit is a logic circuit the output of which depends only on the combination of the inputs. The output does not depend on the past value of inputs or outputs.
- Hence combinational circuits do not require any memory (to store the past values of inputs or outputs).
- Till now we have discussed only the combinational circuits.
- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals. It does not depend on the past status of inputs.
- The combinational circuits do not use any memory. Therefore the previous states of input does not have any effect on the present state of the circuit.
- Also the sequence in which the inputs are being applied has no effect on the output of a combinational circuit.
- We do not have to use any timing and synchronization signal such as clock signal in a combinational circuit.

Sequential circuits :

Definition :

- In the sequential circuit, the **timing** parameter also needs to be taken into consideration.
- The output of a sequential circuit depends on the present time inputs, the previous output (past) and the sequence in which the inputs are applied.
- In order to provide the previous input or output a memory element is required to be used. Thus a sequential circuit needs to use a memory element as shown in Fig. 6.1.1.



(C-560) Fig. 6.1.1 : Block diagram of a sequential circuit

- Fig. 6.1.1 shows the block diagram of a sequential circuit which includes the memory element in the feedback path.

Present state of sequential circuit :

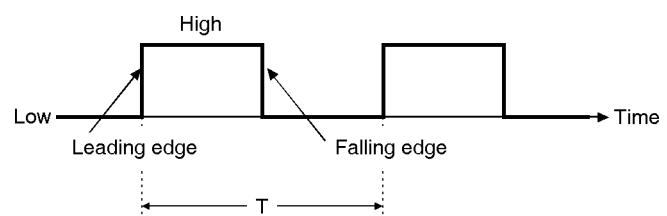
- The data stored by the memory element at any given instant of time is called as the present state of the sequential circuit.

Next state :

- The combinational circuit shown in Fig. 6.1.1 operates on the external inputs and the present state to produce new outputs.
- Some of these new outputs are stored in the memory element and called as the next state of the sequential circuit.
- The most important part of the sequential circuit seems to be the memory element. The memory element of Fig. 6.1.1 is known as Flip Flop (FF). It is the basic memory element.

6.1.1 Clock Signal :

- The clock signal shown in Fig. 6.1.2 is a timing signal. Every sequential signal will have this timing signal applied as an input signal as an input signal.
- Clock is a rectangular signal as shown in Fig. 6.1.2, with a duty cycle equal to 50%. That means its on time is equal to its off time.
- The clock signal repeats itself after every T seconds. Hence the clock frequency is $f = 1/T$.



(C-561) Fig. 6.1.2 : Clock signal

6.1.2 Comparison of Combinational and Sequential Circuits :

SPPU : Dec. 09, Dec. 13, Dec. 18, Dec. 19

University Questions

- Q. 1** Explain the difference between combinational and sequential circuits. (Dec. 09, 4 Marks)
- Q. 2** Explain the difference between combinational and sequential circuit. Design S-R flip-flop using J-K flip-flop. (Dec. 13, 6 Marks)

- Q. 3** Give comparison of combinational circuit with sequential circuit. Draw and explain one-bit memory cell using NAND gates. **(Dec. 18, 6 Marks)**
- Q. 4** Compare combinational circuits with sequential circuits. Convert JK Flip-Flop into SR flip-flop. **(Dec. 19, 6 Marks)**

Sr. No.	Parameter	Combinational circuits	Sequential circuits
1.	Output depends on	Inputs present at that instant of time.	Present inputs and past inputs/outputs.
2.	Memory	Not necessary	Necessary
3.	Clock input	Not necessary	Necessary
4.	Examples	Adders, subtractors, code converters	Flip flops, shift registers, counters

6.1.3 1-Bit Memory Cell (Basic Bistable Element) : **SPPU : Dec. 18**

University Questions

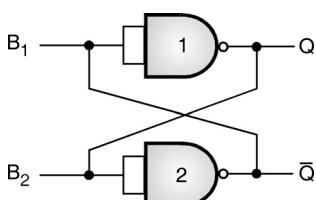
- Q. 1** Give comparison of combinational circuit with sequential circuit. Draw and explain one-bit memory cell using NAND gates. **(Dec. 18, 6 Marks)**

Definition :

- Flip-flop is also known as the basic digital memory circuit.
- It has two stable states namely logic 1 state and logic 0 state. We can design it either using NOR gates or NAND gates.

Circuit diagram :

- A flip-flop can be designed by using the fundamental circuit shown in Fig. 6.1.3.
- NAND gates 1 and 2 are basically acting as inverters. Hence this circuit is called as a cross coupled inverter.
- Output of gate 1 is connected to the input of gate-2 and output of gate 2 is connected to input of gate-1 as shown in Fig. 6.1.3.



(C-562) Fig. 6.1.3 : A cross coupled inverter as memory element

Operation :

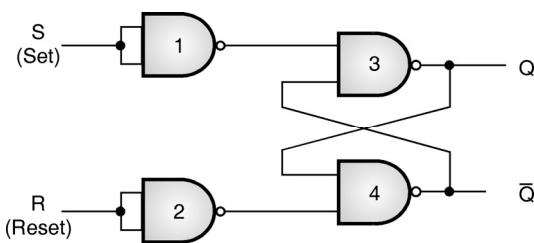
- Assume that output of gate-1 i.e. $Q = 1$. Hence $B_2 = 1$.
- As $B_2 = 1$, output of gate-2 i.e. $\bar{Q} = 0$. This makes $B_1 = 0$. Hence Q continues to be equal to 1.
- Similarly we can demonstrate that if we start with $Q = 0$, then we end up obtaining $Q = 0$ and $\bar{Q} = 1$.

Conclusions :

- From the above discussion we can draw the following conclusions :
- The outputs of the circuit (Q and \bar{Q}) will always be complementary. That means if $Q = 0$ then $\bar{Q} = 1$ and vice versa. They will never be equal $Q = \bar{Q} = 0$ or 1 is an invalid state.
- This circuit has two stable states.
- One of them corresponds to $Q = 1$, $\bar{Q} = 0$ and it is called as 1 state or **set state**. Whereas the other state corresponds to $Q = 0$, $\bar{Q} = 1$ and it is called as 0 state or **reset state**.
- If the circuit is in the reset state ($Q = 0$, $\bar{Q} = 1$), then it will continue to be in the reset state and if it is in the set state ($Q = 1$, $\bar{Q} = 0$) then it will continue to remain in the set state.
- This property of the circuit shows that it can store 1 bit of digital information. Therefore it is called as a **1-bit memory cell**.

6.1.4 Latch :

- The cross coupled inverter of Fig. 6.1.3 is capable of locking or latching the information. Hence this circuit is also called as a latch.
- The disadvantage of the cross coupled inverter circuit is that we cannot enter the desired digital data into it.
- This disadvantage can be overcome by modifying the circuit as shown in Fig. 6.1.4.
- This modification will allow us to enter the desired digital data into the circuit.



(C-563) Fig. 6.1.4 : Modified memory cell

**Operation :****Case I : S = R = 0 (No change) :**

- The outputs of gates 1 and 2 will become 1.
- Let $Q = 0$ and $\bar{Q} = 1$ initially. Hence both the inputs to gate 3 are 1 and the inputs to gate-4 are (01).
- So gate-3 output i.e. $Q = 0$ and gate-4 output i.e. $\bar{Q} = 1$.
- Thus with $S = R = 0$, there is no change in the state of outputs.

Case II : S = 1, R = 0 (Set) :

- Since $S = 1$ and $R = 0$, one of the inputs to gate-3 will be 0. This will force Q output to 1.
- Hence both the inputs to gate-4 will be 1. This forces \bar{Q} to 0.
- Hence for $S = 1, R = 0$, the outputs are $Q = 1$ and $\bar{Q} = 0$. This is set state.

Case III : S = 0, R = 1 (Reset) :

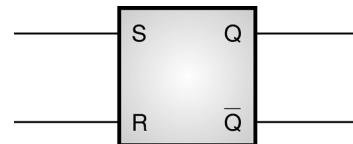
- If $S = 0, R = 1$ then one of the inputs to gate-4 will be 0. This will force the \bar{Q} output to 1.
- Hence both the inputs to gate 3 will be 1. This forces Q to 0.
- Thus for $S = 0, R = 1$, the outputs are $Q = 0, \bar{Q} = 1$. This is the reset state or clear state.

Case IV : S = R = 1 (Race : Prohibited) :

- If $S = R = 1$ then outputs of gates 1 and 2 will be zero.
- Hence one of the inputs to gates 3 and 4 will be 0.
- So both the outputs Q and \bar{Q} will try to become 1. It is not allowed as Q and \bar{Q} should be complementary. Hence $S = R = 1$ condition is prohibited.

6.1.5 Symbol and Truth Table of S-R Latch :

- The symbol and truth table of S-R latch are as shown in Figs. 6.1.5(a) and (b) respectively.
- In the truth table Q_n and \bar{Q}_n represent the present states of outputs i.e. these are the outputs before applying a new set of inputs.
- Q_{n+1} and \bar{Q}_{n+1} represent the next states of outputs i.e. the outputs after applying a new set of inputs.



Q_n and \bar{Q}_n : Present states Q_{n+1} and \bar{Q}_{n+1} : Next states

(C-569) (a) Symbol

Inputs		Outputs				Comment
S	R	Q_n	\bar{Q}_n	Q_{n+1}	\bar{Q}_{n+1}	
0	0	0	1	0	1	No change (NC)
0	0	1	0	1	0	
0	1	0	1	0	1	
0	1	1	0	0	1	
1	0	0	1	1	0	Set
1	0	1	0	1	0	
1	1	0	1	X	X	Prohibited state
1	1	1	0	X	X	

(C-8074) (b) Truth table of S-R latch

Fig. 6.1.5

Summary of operation of S-R latch :

The summary of operation of an S-R latch is as follows :

- For $S = R = 0$ the latch output does not change.
- $S = 0, R = 1$ is called as the "Reset" condition as $Q = 0$ and $\bar{Q} = 1$.
- $S = 1, R = 0$ is called as the "Set" condition as $Q = 1$ and $\bar{Q} = 0$.
- $S = R = 1$ is the prohibited state. The output is unpredictable. This condition should therefore be avoided.

Race condition :

- The condition $S = R = 1$ is called as "Race" condition.
- When any one input to a NOR gate is 1, its output becomes 0. Thus both the outputs will try to become 0. This is called as the RACE condition.

6.1.6 Characteristic Equation :

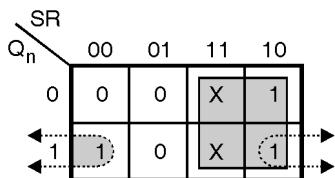
- One way of explaining the behaviour of a latch or flip flop is to use its truth table.
- The truth table is also called as **excitation table**.
- Another way of doing it is to use special type of equations called **characteristic equations**.



- The characteristic equation of a flip-flop is the equation which relates the next state of the flip flop or latch Q_{n+1} or \bar{Q}_{n+1} to the current state and inputs (Q_n , S and R).
- Characteristic equation is actually obtained from the truth table of the flip flop or latch, using the K-map.

Characteristic equation of SR latch :

- Refer to the truth table of SR latch and write the K-map for the next state of output i.e. Q_{n+1} as shown in Fig. 6.1.6.

(C-570) Fig. 6.1.6 : K-map for next state Q_{n+1} of SR latch

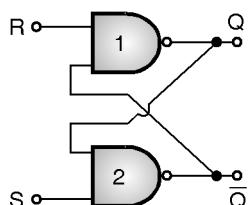
- After simplification, the characteristic equation of SR latch is given by,

$$Q_{n+1} = S + \bar{R} Q_n \quad \dots(6.1.1)$$

6.1.7 NAND Latch [S-R Latch using NAND Gates] :

Logic diagram :

- We can construct a S-R latch with NAND gates as shown in Fig. 6.1.7(a).
- Note that the outputs of two NAND gates are cross connected, in an identical manner as that in the NOR latch.
- Fig. 6.1.7(a) shows the NAND latch and Fig. 6.1.7(b) shows its truth table.



(a) NAND latch

S	R	Q_{n+1}	\bar{Q}_{n+1}
0	0	RACE	RACE
0	1	0	1
1	0	1	0
1	1	(NC) Q_n	(NC) \bar{Q}_n

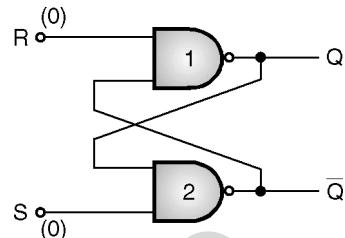
(b) Truth table

(C-571) Fig. 6.1.7

Operation :

- The operation of S-R NAND latch is summarised in Fig. 6.1.8.

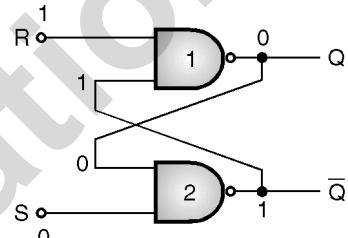
Case I : S = 0, R = 0 : Race



(C-572) Fig. 6.1.8(a) : S = 0, R = 0

- When any one input of a NAND gate becomes 0, its output is forced to 1.
- Here S = R = 0. $\therefore Q$ and \bar{Q} both will be forced to be equal to 1.
- This is an undeterminate state and hence should be avoided.
- This is also called as Race condition.

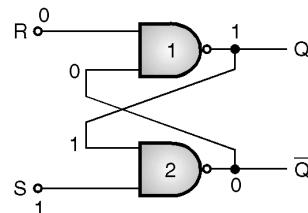
Case II : S = 0, R = 1 : Reset



(C-573) Fig. 6.1.8(b)

- Since S = 0, it forces \bar{Q} to be 1.
- Hence both inputs to NAND-1 are 1.
- Hence Q = 0.
- Thus with S = 0 and R = 1 the outputs are Q = 0 and $\bar{Q} = 1$.
- This is the reset condition.

Case III : S = 1, R = 0 : Set



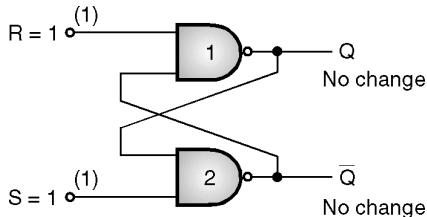
(C-573(a)) Fig. 6.1.8(c)

- Since R = 0, Q is forced to 1.
- Hence both inputs to NAND-2 are 1.
- Hence $\bar{Q} = 0$.



- Thus with $S = 1$ and $R = 0$ the outputs are $Q = 1$ and $\bar{Q} = 0$.
- This is the set condition.

Case IV : $S = 1, R = 1$: No change



(C-574) Fig. 6.1.8(d)

$$Q_{n+1} = \overline{R \cdot Q_n} \text{ and } \bar{Q}_{n+1} = \overline{S \cdot Q_n}$$

- Using De-Morgan's theorem,

$$Q_{n+1} = \overline{R} + Q_n \text{ and } \bar{Q}_{n+1} = \overline{S} + \bar{Q}_n.$$

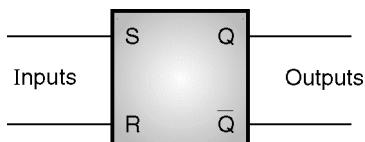
- Substitute $\overline{R} = 0$ and $\overline{S} = 0$ to get,

$$Q_{n+1} = 0 + Q_n = Q_n \text{ and } \bar{Q}_{n+1} = 0 + \bar{Q}_n = \bar{Q}_n.$$

- Thus there is no change in the outputs if $S = R = 1$.
- The symbol for S-R latch is shown in Fig. 6.1.9 alongwith the summary of operation.

Circuit symbol and summary of operation of S-R latch :

- For $S = 0, R = 0$, both the outputs will be forced to become 1. This is RACE condition and should be avoided.
- For $S = 0, R = 1$, the outputs are $Q = 0, \bar{Q} = 1$ and it is called as the Reset condition.
- For $S = 1, R = 0$, the outputs are $Q = 1, \bar{Q} = 0$ and it is called as the set condition.
- For $S = R = 1$, there is no change in the output state.



(C-575) Fig. 6.1.9 : Symbol of S-R latch

6.2 Triggering Methods :

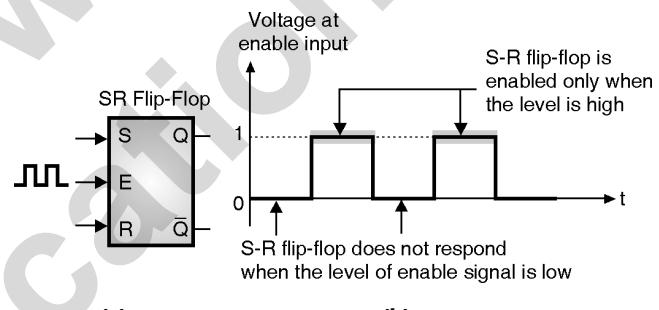
- In the latches and flip-flops, we use the additional signal called clock signal.
- Depending on which portion of the clock signal the latch or flip-flop responds to, we can classify them into two types :

1. Level triggered circuits.
2. Edge triggered circuits.

6.2.1 Concept of Level Triggering :

Definition :

- The latch or flip-flop circuits which respond to change in their inputs, only if their enable input (E) held at an active level which may be either HIGH or LOW level are called as **level triggered** latches or flip-flops.
- Thus these circuits do not respond at the rising or falling edges of clock. They only respond to the steady HIGH or LOW levels of the clock signal.
- Fig. 6.2.1(a) shows the symbol of a level triggered SR flip flop and Fig. 6.2.1(b) shows the clock signal applied at its input.



(C-576) Fig. 6.2.1 : Concept of level triggering

6.2.2 Types of Level Triggered Flip-flops :

- There are two types of level triggered flip-flops :
 1. Positive level triggered.
 2. Negative level triggered.

Positive level triggered :

- If the outputs of a flip-flop respond to the input changes, only when its clock inputs at HIGH (1) level, then it is called as the positive level triggered flip-flop.
- The block diagram shown in Fig. 6.2.1(a) is a positive level triggered S-R FF.

Negative level triggered FF :

If the outputs of a flip-flop respond to the input changes, only when its clock input is at LOW (0) level, then it is called as the negative level triggered flip-flop.

Note : The level triggering is not used practically, due to some of its disadvantages.



6.2.3 Concept of Edge Triggering :

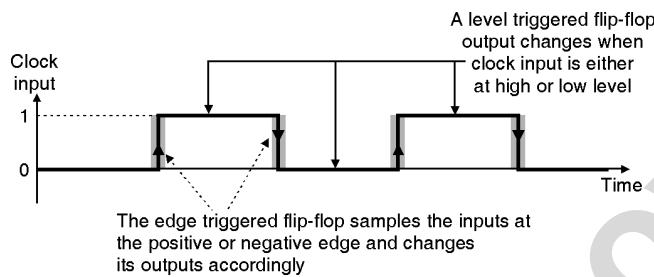
SPPU : May 11

University Questions

Q. 1 What is edge triggering ? **(May 11, 2 Marks)**

Definition :

- The flipflops which change their outputs only corresponding to the positive (rising) or negative (falling) edge of the clock input are called as **edge triggered flipflops**.
- These flip-flops are therefore said to be edge sensitive or edge triggered rather and not level triggered.



(C-577) Fig. 6.2.2

- The rectangular signal applied to the clock input of a flip-flop is shown in Fig. 6.2.2.
- If the same signal is applied as the clock signal to an edge triggered flip flop, then its outputs will change only at either rising (positive) edge or at the falling (negative) edge of the clock.
- The edge triggered flip-flops do not respond to the steady state high or low level in the clock signal at all.

6.2.4 Types of Edge Triggered Flip Flops :

- There are two types of edge triggered flip flops :
 1. Positive edge triggered flip flops.
 2. Negative edge triggered flip flops.
- Positive edge triggered flip flops will allow its outputs to change in response to its inputs only at the instants corresponding to the rising edges of clock (or positive spikes).
- Its outputs will not respond to change in inputs at any other instant of time.
- Negative edge triggered flip flops will respond only to the negative going edges (or spikes) of the clock.

6.3 Gated Latches (Level Triggered SR Flip Flop) :

- We have discussed the RS latches using the NAND and NOR gates.
- Now 2 more NAND gates are added to the basic SR latch and one more input called enable (E) is added, in order to obtain a gated SR latch or a level triggered SR flip-flop.
- A level voltage (0 or 1) or a clock signal can be applied to the enable (E) input.
- These flipflops will respond to the inputs if and only if we apply an **active level** at the enable input. This active level can be either 0 or 1 depending on the type of flip-flop.
- Such flipflops are called as level triggered flipflops or gated latches or clocked flipflops.

6.3.1 Types of Level Triggered (Clocked) Flip Flops :

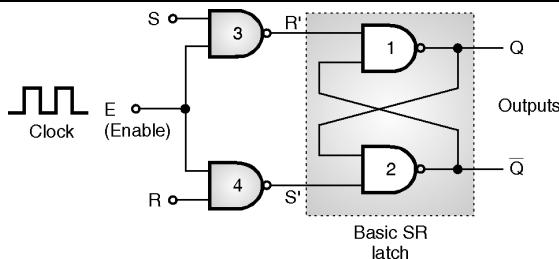
- There are two types of level triggered latches :
 1. Positive level triggered.
 2. Negative level triggered.

6.4 The Gated S-R Latch (Level Triggered S-R Flip Flop) :

6.4.1 Positive Level Triggered SR Flip-flop :

Logic diagram :

- The gated S-R latch is shown in Fig. 6.4.1. It is also called as clocked SR flip flop.
- It is basically the S-R latch using NAND gates with an additional "enable" (E) input. It is also called as **level triggered S-R FF**.
- The outputs of basic S-R latch used to change instantly in response to any change made at the input. But this does not happen with the gated S-R latch.
- For this circuit, the change in output will take place if and only if the enable input (E) is made active.
- This circuit being positive level triggered, will respond to changes in input only if the enable input is held at logic 1 level.
- In short this circuit will operate as an S-R latch if E = 1 (Enable input is active) but there is no change in the outputs if E = 0 (Enable input is inactive).



(C-578) Fig. 6.4.1 : Gated S - R latch

Operation :**Case I : S = X, R = X, E = 0 (No change)**

- Since enable $E = 0$, the outputs of NAND gates 3 and 4 will be forced to be 1 irrespective of the values of S and R .
- That means $R' = S' = 1$. These are the inputs of the basic S-R latch enclosed in the dotted box in Fig. 6.4.1.
- Hence the outputs of NAND latch i.e. Q and \bar{Q} will not change. Thus if $E = 0$, then there is no change in the output of the gated S-R latch.

Case II : S = R = 0 E = 1 : No change

- If $S = R = 0$ then outputs of NAND gates 3 and 4 are forced to become 1.
- Hence R' and S' both will be equal to 1. Since S' and R' are the inputs of the basic S-R latch using NAND gates, there will be no change in the state of outputs.
- Thus for $S = R = 0$ the output state of this flip-flop remains unchanged.

Case III : S = 0, R = 1, E = 1 (Reset)

- Since $S = 0$, output of NAND-3 i.e. $R' = 1$. And as $R = 1$ and $E = 1$ the output of NAND-4 i.e. $S' = 0$.
- Hence $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$. This is the reset condition.

Case IV : S = 1, R = 0, E = 1 (Set)

- Output of NAND 3 i.e. $R' = 0$ and output of NAND 4 i.e. $S' = 1$.
- Hence output of S-R NAND latch is $Q_{n+1} = 1$ and $\bar{Q}_{n+1} = 0$. This is the set condition.

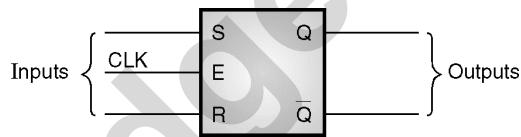
Case V : S = 1, R = 1, E = 1 (RACE)

- As $S = 1$, $R = 1$ and $E = 1$, the outputs of NAND gates 3 and 4 both are 0. i.e. $S' = R' = 0$.

- Hence the "Race" condition will occur in the basic NAND latch. This operation should be avoided as both Q and \bar{Q} will try to become 1 at the same time as discussed earlier.

Symbol and truth table :

- The symbol and truth table of the gates S-R latch are as shown in Fig. 6.4.2(a) and Fig. 6.4.2(b) respectively.



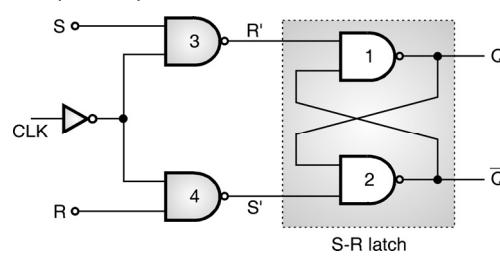
(C-579) Fig. 6.4.2(a) : Symbol for S - R latch

(C-7827) Fig. 6.4.2(b) : Truth table of gated S - R latch

Case	Enable E	Inputs		Outputs		Comments
		S	R	Q_{n+1}	\bar{Q}_{n+1}	
I	0	x	x	Q_n	\bar{Q}_n	No change as E = 0
II	1	0	0	Q_n	\bar{Q}_n	No change (NC)
III	1	0	1	0	1	Reset condition
IV	1	1	0	1	0	Set condition
V	1	1	1	RACE (Indeterminate)		Avoid this condition

6.4.2 Negative Level Triggered SR Flip Flop :**Logic diagram :**

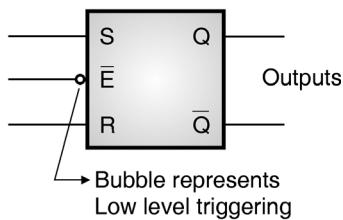
- Fig. 6.4.3(a) shows the circuit diagram of a negative level triggered SR flip-flop.
- It is the same circuit that we discussed in the previous section with only one additional inverter.
- Due to the additional inverter connected to the enable terminal, this circuit becomes sensitive to the low level (0) applied to the enable input. Hence it will enable the outputs if $E = 0$.
- If a square wave is applied to the enable input then the latch output will respond to input changes only when the square input is at its low (0) level.



(C-580) Fig. 6.4.3(a) : Logic diagram of Negative level triggered S - R latch



- Fig. 6.4.3(b) shows the symbol of negative level triggered S-R latch. Note that there is a bubble added to the enable input which is active low input.



(b) Symbol

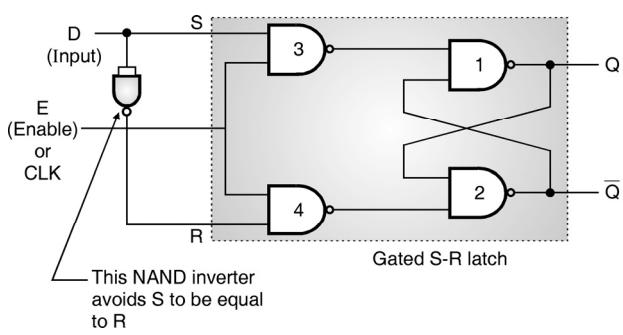
(C-580) Fig. 6.4.3 : Negative level triggered S - R latch

6.5 The Gated D Latch (Clocked D Flip Flop) :

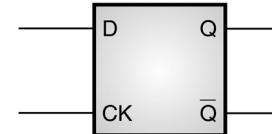
- In some applications, the S and R inputs will always be complementary. i.e. when $S = 0, R = 1$ and when $S = 1, R = 0$. That means $S = \bar{R}$.
- For such applications we can use the gated D latch.

Logic diagram :

- The circuit diagram of the gated D latch is shown in Fig. 6.5.1(a) and its logic symbol is shown in Fig. 6.5.1(b). This is also called as level triggered D flip-flop or clocked D flip-flop.
- Note that D latch is the simple gated S-R latch with a small modification. A NAND inverter is connected between its S and R inputs as shown in Fig. 6.5.1(a).
- This latch has only one input denoted by D.
- Due to the NAND inverter, S and R inputs will always be the complements of each other. Hence the input conditions such as $S = R = 0$ or $S = R = 1$, will never appear.
- This will avoid the problems associated with $SR = 00$ and $SR = 11$ conditions of SR flip-flop.



(a) Gated D latch



(b) Logic symbol of gated D latch

(C-3419) Fig. 6.5.1

Truth table :

- Truth table for the gated D latch is given in Table 6.5.1.

(C-8061) Table 6.5.1 : Truth table for the gated D latch

Inputs		Outputs		Comment
E	D	Q_{n+1}	\bar{Q}_{n+1}	
0	X	Q_n	\bar{Q}_n	No change (NC)
1	0	0	1	Reset condition
1	1	1	0	Set condition

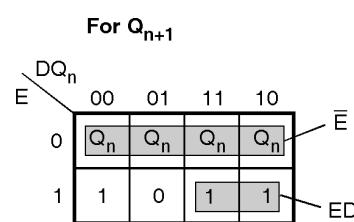
Operation :

- If $E = 0$ then the latch is disabled. Hence there is no change in output.
- If $E = 1$ and $D = 0$ then $S = 0$ and $R = 1$. Hence irrespective of the present state the, next state is $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$. This is the reset condition.
- On the other hand if $E = 1$ and $D = 1$, then $S = 1$ and $R = 0$. This will set the latch and $Q_{n+1} = 1$, $\bar{Q}_{n+1} = 0$ irrespective of the present state.

From the truth table it is evident that Q output is same as D input, in otherwords Q output follows the D input. If $D = 0$ then $Q = 0$ and if $D = 1$ then $Q = 1$. However output follows input after some propagation delay hence the other name of the D flip-flop is delay flip-flop.

Characteristic equation for D latch :

- Refer to the truth table of D latch and write the K-map for Q_{n+1} as shown in Fig. 6.5.2.



(C-584) Fig. 6.5.2 : K-map for D latch

- After simplification we get the characteristic equation of D latch as,

$$Q_{n+1} = E \cdot D + \bar{E}$$

6.6 Gated JK Latch (Level Triggered JK Flip Flop) :

SPPU : May 19

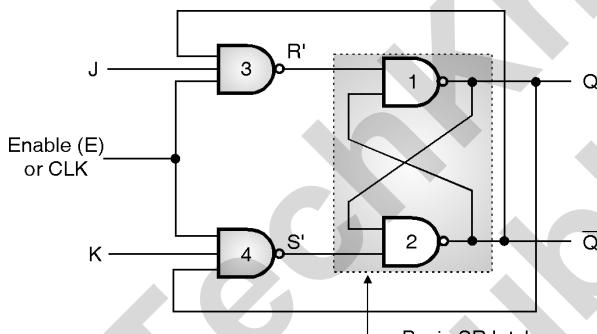
University Questions

Q. 1 Draw JK flip flop using gates and explain race around condition with the help of timing diagram.

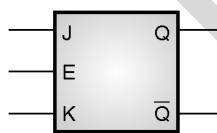
(May 19, 6 Marks)

Logic diagram :

- The JK latch using NAND gates is shown in Fig. 6.6.1(a). It consists of the basic SR latch and an enable input. It is also called as level triggered JK flip-flop.
- Note the outputs Q and \bar{Q} have been fed back and connected to the inputs of NAND gates 4 and 3 respectively.
- The JK latch of Fig. 6.6.1 (a) responds, to the input changes if a positive level is applied at the enable (E) input. Hence it is a positive level triggered latch.



(a) A level triggered JK flip-flop



(b) Symbol of the gated JK latch

(C-3420) Fig. 6.6.1

Operation :

- The operation of NAND JK latch is exactly identical to that of the positive edge triggered JK flip flop discussed in section 6.9.1.
- The only difference between them is that, this circuit is level triggered.
- The operation of NAND JK latch has been summarized in Table 6.6.1.

(C-3420(a)) Table 6.6.1 : Truth table of positive level triggered JK latch

Case No.	Inputs			Outputs		State
	E	J	K	Q_{n+1}	\bar{Q}_{n+1}	
Case I	0	x	x	Q_n	\bar{Q}_n	No change
Case II	1	0	0	Q_n	\bar{Q}_n	No change
Case III	1	0	1	0	1	Reset
Case IV	1	1	0	1	0	Set
Case V	1	1	1	\bar{Q}_n	Q_n	Toggle

6.6.1 Race Around Condition in JK Latch :

SPPU : May 07, Dec. 13, May 15, Dec. 15, May 19

University Questions

Q. 1 What is race-around condition ? How will you eliminate it ? Explain with the help of necessary circuit diagram and timing diagram.

(May 07, 8 Marks)

Q. 2 What is race around condition ? How it can be avoided ? Convert D flip-flop to T flip-flop.

(Dec. 13, 6 Marks)

Q. 3 What is race around condition ? Explain with the help of timing diagram. How is it removed in basic flip-flop circuit ?

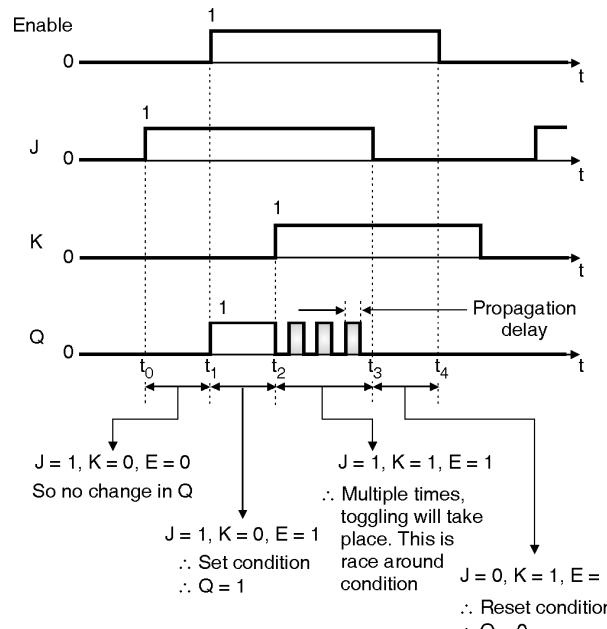
(May 15, Dec. 15, 6 Marks)

Q. 4 Draw JK flip flop using gates and explain race around condition with the help of timing diagram.

(May 19, 6 Marks)

– The “Race Around Condition” that we are going to explain occurs when $J = K = 1$ i.e. when the latch is in the toggle mode.

– Refer Fig. 6.6.2 which shows the waveforms for the various modes, when a rectangular waveform is applied to the “Enable” input.



(C-586) Fig. 6.6.2 : Waveforms for various modes of a JK latch

Interval $t_0 - t_1$:

- During this interval $J = 1, K = 0$ and $E = 0$.
- Hence the latch is disabled and there is no change in Q .

Interval $t_1 - t_2$:

- During this interval $J = 1, K = 0$ and $E = 1$.
- Hence this is a set condition and Q becomes 1.

Interval $t_2 - t_3$: Race around

- At instant t_2 , $J = K = 1$ and $E = 1$ Hence the JK latch is in the toggle mode and Q becomes low (0) and $\bar{Q} = 1$.
- These changed outputs get applied at the inputs of NAND gates 3 and 4 of the JK latch. Thus the new inputs to Gates 3 and 4 are :

NAND - 3 : $J = 1, E = 1, \bar{Q} = 1$,

NAND - 4 : $K = 1, E = 1, Q = 0$.

- Hence R' will become 0 and S' will become 1.
- Therefore after a time period corresponding to the propagation delay, the Q and \bar{Q} outputs will change to, $Q = 1$ and $\bar{Q} = 0$.
- These changed output again get applied to the inputs of NAND-3 and 4 and the outputs will toggle again.
- Thus as long as $J = K = 1$ and $E = 1$, the outputs will keep toggling indefinitely as shown in Fig. 6.6.2. This multiple toggling in the J-K latch is called as Race Around condition. It must be avoided.

Interval $t_3 - t_4$:

- During this interval $J = 0, K = 1$ and $E = 1$.
- Hence it is the reset condition. So Q becomes zero.

How to avoid race around condition ?

- The race around condition in JK latch can be avoided by,
 1. Using the edge triggered JK flip flop.
 2. Using the master slave JK flip flop.

6.6.2 Difference between Latch and Flip-flop :**Latches :**

- Latches and flip flops both are basically the bistable elements.
- As discussed earlier a latch has got an enable input. As long as it is active, the latch output will keep changing according to the changes in its input.

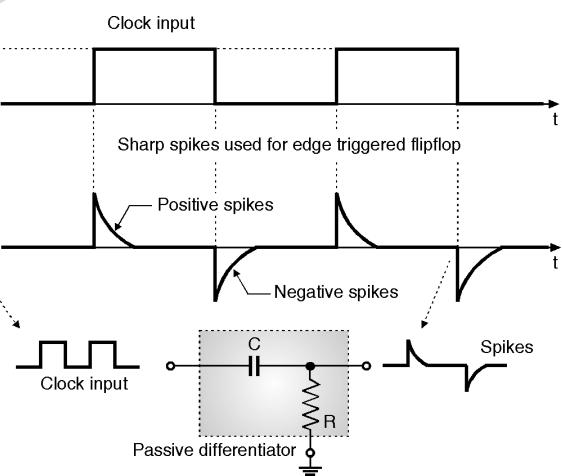
- In other words latch is a level triggered flip flop.

Flip-flop :

- But flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously.
- The flip-flops are therefore said to be edge sensitive or edge triggered rather than being level triggered like latches.

6.7 Edge Triggered Flip Flops :

- We have already discussed the concept of edge triggering.
- For the edge triggered flip flops, it is necessary to apply the clock signal (timing signal) in the form of sharp positive and negative spikes instead of in the form of a rectangular pulse train.
- Such sharp spikes are shown in Fig. 6.7.1. These spikes can be derived from the rectangular clock pulses with the help of a passive differentiator circuit shown in Fig. 6.7.1.
- Thus the passive differentiator acts as a pulse shaping circuit.



(C-587) Fig. 6.7.1 : Use of differentiator to obtain sharp edges

6.7.1 Positive Edge Triggered S-R Flip Flop :

SPPU : Dec. 11

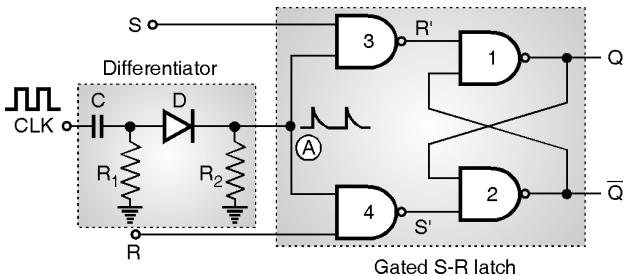
University Questions

Q. 1 Draw and explain SR flip-flop using NAND gates.

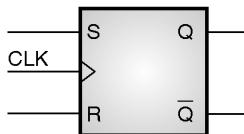
(Dec. 11, 8 Marks)

Circuit diagram :

- Fig. 6.7.2(a) shows the circuit diagram of a positive edge triggered S-R flip flop and Fig. 6.7.2(b) shows the logic symbol for it.



(a) Positive edge triggered S-R flip flop

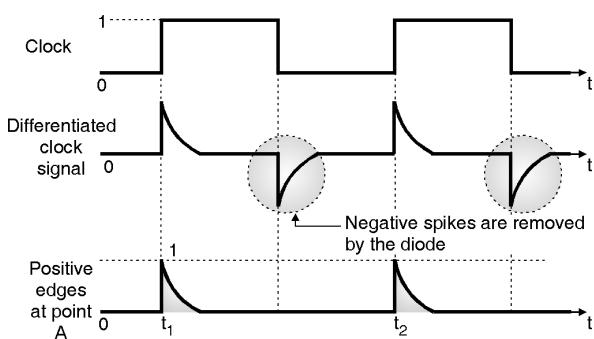


(b) Logic symbol

(C-588) Fig. 6.7.2

Operation :

- Note that the SR flip flop of Fig. 6.7.2(a) consists of a differentiator circuit and a gated SR latch (level triggered SR flip-flop) which we have already discussed.
- $C - R_1$ acts as a differentiator and converts the rectangular clock pulses into positive and negative spikes.
- The diode acts as rectifying diode and allows only the positive spikes to pass through it while, blocking the negative spikes.
- Thus we get only the positive spikes, corresponding to the leading edges of the clock signal, as shown in Fig. 6.7.3 at point A which is enable input of the gated SR latch.



(C-589) Fig. 6.7.3 : Generation of positive triggering pulses

- Due to these sharp positive spikes applied at point "A", the gated S-R latch in the S-R flip flop will be enabled only for a short duration of time when the positive spike is present at A. (at instants $t_1, t_2 \dots$ in Fig. 6.7.3)
- At these instants, the flip-flop behaviour is exactly identical to that of the enabled gated S-R latch (enabled level triggered SR flip-flop).
- The truth table of positive edge triggered S-R flip flop will also be identical to that of the gated S-R latch with only one change.
- The "enable" input will now be replaced by **clock** input. And the outputs will change only if a positive edge is applied to the clock input.
- The positive clock edge is denoted by an arrow (\uparrow) in Table 6.7.1.

Truth Table :

- The truth table of a positive edge triggered SR flip flop is shown in Table 6.7.1.

(C-6260) Table 6.7.1 : Truth table of a positive edge triggered SR flip flop

CLK	Inputs		Outputs		Remark
	S	R	Q_{n+1}	\bar{Q}_{n+1}	
0	x	x	Q_n	\bar{Q}_n	No change (NC)
1	x	x	Q_n	\bar{Q}_n	No change (NC)
↓	x	x	Q_n	\bar{Q}_n	No change (NC)
	0	0	Q_n	\bar{Q}_n	No change (NC)
	0	1	0	1	Reset
↑	1	0	1	0	Set
	1	1	Race	Race	Avoid

↓ = Negative edge of clock, ↑ = Positive edge of clock

- Note that this flip-flop does not respond to the positive or negative levels in the clock signal.
- Similarly it does not respond to the negative edges of the clock.
- This flip flop will respond only to the positive edges of clock signal.
- With positive edge of the clock, the SR flip flop behaves in the following way.

$S = R = 0 \rightarrow$ No change in output

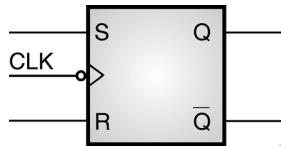
$S = 0, R = 1 \rightarrow Q_{n+1} = 0, \bar{Q}_{n+1} = 1$ Reset condition

$S = 1, R = 0 \rightarrow Q_{n+1} = 1, \bar{Q}_{n+1} = 0$ Set condition
 $S = R = 1 \rightarrow$ Race condition.

6.7.2 Negative Edge Triggered S - R Flip Flop :

Symbol and Truth table :

- The internal circuit (with NAND gates) of the negative edge triggered S-R flip flop is exactly same as that for the positive edge triggered SR flip-flop discussed earlier.
- The differentiator circuit is slightly modified in order to make this flip flop respond to the negative (falling) edges of the clock input.
- Fig. 6.7.4 shows the circuit symbol of the negative edge triggered S-R flip flop and Table 6.7.2 shows its truth table.



(C-590(a)) Fig. 6.7.4 : Circuit symbol of negative edge triggered SR FF

(C-6207) Table 6.7.2 : Truth table of negative edge triggered S-R FF

Inputs			Outputs		State
CLK	S	R	Q_{n+1}	\bar{Q}_{n+1}	
0	x	x	Q_n	\bar{Q}_n	No change (NC)
1	x	x	Q_n	\bar{Q}_n	No change (NC)
↑	x	x	Q_n	\bar{Q}_n	No change (NC)
↓	0	0	Q_n	\bar{Q}_n	No change (NC)
↓	0	1	0	1	Reset
↓	1	0	1	0	Set
↓	1	1	Race	Race	Avoid

↑ = Positive edge of clock, ↓ = Negative edge of clock

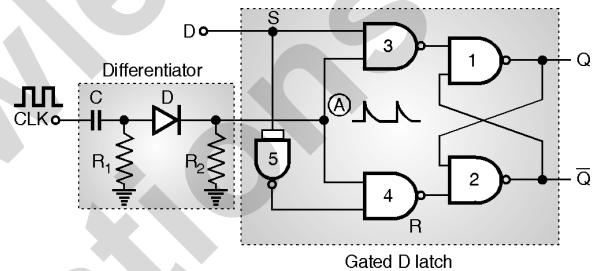
6.8 Edge Triggered D Flip Flop :

- The edge triggered D flip flops can be of two types :
 - Positive edge triggered D flip flop.
 - Negative edge triggered D flip flop.
- These flip flops can be derived from the level triggered D latch which we have discussed in section 6.5.

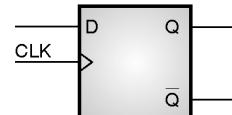
6.8.1 Positive Edge Triggered D Flip Flop :

Logic diagram :

- Fig. 6.8.1(a) shows the positive edge triggered D flip flop. It consists of a gated D latch and a differentiator circuit. The symbol is as shown in Fig. 6.8.1(b).
- The clock pulses are applied to the circuit through a differentiator formed by $R_1 C$ and a rectifier circuit consisting of diode D and R_2 .
- The NAND gates 1 through 5 form a D latch.
- The differentiator converts the clock pulses into positive and negative spikes and the combination of D and R_2 will allow only the positive spikes to pass through to point "A", by blocking the negative spikes.



(a) Positive edge triggered D-flip flop



(b) Symbol

(C-592(a)) Fig. 6.8.1

Operation :

- The operation of edge triggered D FF is exactly same as that of the gated D latch discussed earlier with only one difference.
- The D latch had an enable terminal whereas the D flip flop has a clock input.
- The gated D latch of Fig. 6.8.1 is enabled by the positive spikes obtained at point A.
- Hence the outputs will change based on the inputs at these particular instants only.
- Thus the edge triggered D flip-flop responds only to the positive (leading) edges of the clock pulses.
- At any other instants of time, the D flip flop will not respond to the changes in input.

Truth table :

- Table 6.8.1 shows the truth table of a positive edge triggered D flip flop.



(C-6208) Table 6.8.1 : Truth table of a positive edge triggered D flip flop

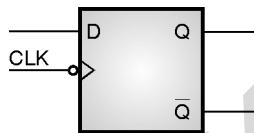
Inputs		Outputs		State
CLK	D	Q_{n+1}	\bar{Q}_{n+1}	
0	x	Q_n	\bar{Q}_n	No change
1	x	Q_n	\bar{Q}_n	No change
↓	x	Q_n	\bar{Q}_n	No change
↑	0	0	1	Q follows D input
↑	1	1	0	

From the truth table it is clear that the Q output of the flip flop follows the D input.

6.8.2 Negative Edge Triggered D Flip Flop :

Symbol :

- The symbol for negative edge triggered D flip flop is shown in Fig. 6.8.2.



(C-593) Fig. 6.8.2 : Symbol of negative edge triggered D flip flop

- This F/F responds only to the negative edges of the clock pulses. This is how it is different from the positive edge triggered flip flop.
- Otherwise, the operation of the negative edge triggered D flip flop is exactly same as that of the positive edge triggered D flip flop.

Truth Table :

- The truth table of negative edge triggered D flip flop is shown in Table 6.8.2.

(C-6209) Table 6.8.2 : Truth table of negative edge triggered D flip flop

Inputs		Outputs		Comment
CLK	D	Q_{n+1}	\bar{Q}_{n+1}	
0	x	Q_n	\bar{Q}_n	No change
1	x	Q_n	\bar{Q}_n	No change
↑	x	Q_n	\bar{Q}_n	No change
↓	0	0	1	Q output follows D input
↓	1	1	0	

6.8.3 Applications of D Flipflop :

- As a delay element.
- In the digital latches.

Transparent latch :

- In level triggered D FF (D latch) the Q output always follows the changes in D input.
- Hence D latch is sometimes called as transparent latch.

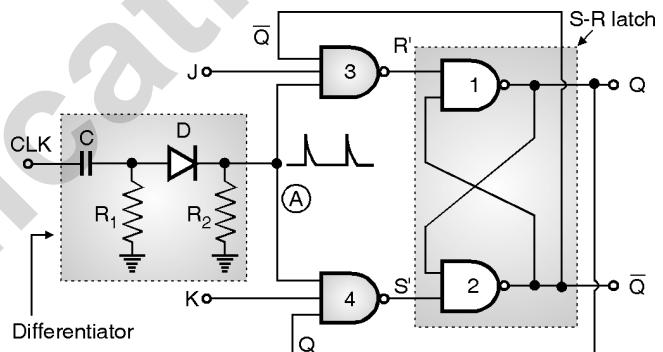
6.9 Edge Triggered J-K Flip Flop :

- Edge triggered J-K flip flops are of two types :
 - Positive edge triggered JK flip flop
 - Negative edge triggered JK flip flop

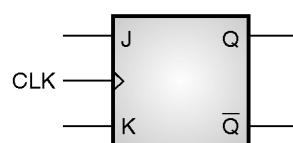
6.9.1 Positive Edge Triggered JK Flip Flop :

Logic diagram :

- The circuit diagram of the positive edge triggered JK flip flop is shown in Fig. 6.9.1(a) and its circuit symbol is shown in Fig. 6.9.1(b).



(C-596) Fig. 6.9.1(a) : Positive edge triggered JK flip flop



(C-596) Fig. 6.9.1(b) : Symbol

Operation :

- The clock signal is a train of rectangular or square waves.
- It is passed through a differentiator ($C - R_1$) and a rectifier (D and R_2), to obtain only the positive spikes at point A as already discussed.
- The negative spikes are blocked by the diode.



- NAND gates 1 and 2 form the basic S-R latch. The other two NAND gates (3 and 4) have three inputs each. The outputs Q and \bar{Q} are fed back to the inputs of gates 4 and 3 respectively.
- Referring to Fig. 6.9.1(a) we can write the mathematical expressions for S' and R' as follows :

$$S' = \overline{K \cdot Q \cdot CLK} \text{ and } R' = J \cdot \bar{Q} \cdot CLK$$

- Let us now understand the operation step by step.

Case I : CLK = 0 or 1 i.e. level

- If CLK = 0 or 1 i.e. level and no pulse this flip flop is disabled, because the output of differentiator is zero for any level input.
- Therefore Q and \bar{Q} output do not change their state.

Case II : If clock edge is falling edge (\downarrow)

- For the falling edge of the clock, the rectifier (D – R_2) will block the negative spike. Hence voltage at point A is logic 0.
- So one input to both the NAND gates 3 and 4 is 0.
- Hence both S' and R' will be forced to 1.
- For an SR latch using NAND gates, the outputs will remain unchanged if S = R = 1.
- Corresponding to the falling edge of clock, the outputs Q and \bar{Q} will remain unchanged.

Case III : CLK = \uparrow , J = 0, K = 0 (No change)

- Since J = 0, R' = 1 and as K = 0, S' = 1.
- As S' = R' = 1 the outputs Q and \bar{Q} will not change their state even though a positive edge of the clock pulse is being applied.

$$\therefore J = 0, K = 0, Q_{n+1} = Q \text{ and } \bar{Q}_{n+1} = \bar{Q}.$$

- No change in output.

Case IV : CLK = \uparrow , J = 0, K = 1 (Reset)

- Recall the expressions for S' and R',

$$S' = \overline{K \cdot Q \cdot CLK} \text{ and } R' = J \cdot \bar{Q} \cdot CLK$$

- If the previous state of Q and \bar{Q} is Q = 1 and $\bar{Q} = 0$ then

$$S' = \overline{1 \cdot 1 \cdot 1} = 0 \text{ and } R' = \overline{0 \cdot 0 \cdot 1} = 1$$

- Therefore according to the operation of S-R latch if S' = 0, R' = 1 then Q = 0, and $\bar{Q} = 1$.

$$\therefore Q = 0 \text{ and } \bar{Q} = 1$$

- Thus with J = 0, K = 1 and positive going clock, the JK flip flop will **reset**.

- If Q = 0 and $\bar{Q} = 1$ before the application of clock pulse, then there will not be any change in their state.

Case V : CLK = \uparrow , J = 1, K = 0 (Set)

- If the previous state of Q and \bar{Q} is Q = 0 and $\bar{Q} = 1$ then,

$$S' = \overline{0 \cdot 0 \cdot 1} = 1 \text{ and } R' = \overline{1 \cdot 1 \cdot 1} = 0$$

- Hence according to the operation of S-R latch, if S' = 1 and R' = 0 then,

$$Q = 1 \text{ and } \bar{Q} = 0 \dots \text{i.e. the flip flop is set.}$$

- If Q and \bar{Q} are already 1,0 then there will not be any change in the output state.

Case VI : CLK = \uparrow , J = 1, K = 1 Toggle mode

- We will discuss this case for two different previous cases.

• Previously let Q = 1 and $\bar{Q} = 0$	• Previously let Q = 0 and $\bar{Q} = 1$
↓	↓
• $S' = \overline{K \cdot Q \cdot CLK}$ and $R' = \overline{J \cdot \bar{Q} \cdot CLK}$	• $S' = \overline{1 \cdot 0 \cdot 1}$ and $R' = \overline{1 \cdot 1 \cdot 1}$
↓	↓
$\therefore S' = \overline{1 \cdot 1 \cdot 1} = 0$ and $R' = \overline{1 \cdot 0 \cdot 1} = 1 \therefore S = 1 \text{ and } R = 0$	$\therefore S = 1 \text{ and } R = 0$
↓	↓
• Hence the SR latch will reset	• Hence the SR latch will set
↓	↓
• So $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$	• So $Q_{n+1} = 1$ and $\bar{Q}_{n+1} = 0$

(C-6399)

- From the operation discussed above, we conclude that when J = K = 1 and a positive clock edge is applied, then Q and \bar{Q} outputs are inverted i.e. $Q_{n+1} = \bar{Q}_n$ and $\bar{Q}_{n+1} = Q_n$.
- This is called as the **TOGGLING** mode. This is a very important mode of operation.



Truth table of JK flip flop :

(C-6837) Table 6.9.1 : Truth table for positive edge triggered JK FF

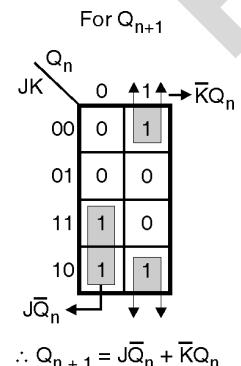
Case No.	Inputs			Outputs		State
	CLK	J	K	Q_{n+1}	\bar{Q}_{n+1}	
Case I	0 or 1	x	x	No change	No change	Flip flop is disabled
Case II	↓	x	x	No change	No change	
Case III	↑	0	0	No change	No change	
Case IV	↑	0	1	0	1	Reset
Case V	↑	1	0	1	0	Set
Case VI	↑	1	1	\bar{Q}_n	Q_n	Toggle

6.9.2 Characteristic Equation of JK Flip Flop :

- Refer to the truth table of JK flip flop (Table 6.9.2) and write the K-map for Q_{n+1} as shown in Fig. 6.9.2.

(C-8062) Table 6.9.2 : Truth table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



(C-597) Fig. 6.9.2 : K-map for Q_{n+1}

∴ The characteristic equation is,

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

6.9.3 How does an Edge Triggered JK FF Avoid Race Around Condition ?

SPPU : Dec. 07, Dec. 08, Dec. 09

University Questions

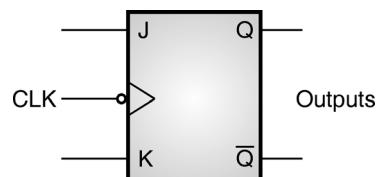
- Q. 1 Discuss methods to avoid race around condition in JK flip-flop. (Dec. 07, 4 Marks)
- Q. 2 How is race around condition avoided ? (Dec. 08, Dec. 09, 3 Marks)

- For the race around to take place, it is necessary to have the enable input high along with $J = K = 1$.
- As the enable input remains high for a long time in a JK latch, the problem of multiple toggling arises which is known as Race condition.
- But in edge triggered JK flip flop, the positive clock pulse is present only for a very short time because it is in the form of a narrow differentiated spike.
- Hence by the time the toggled outputs (Q and \bar{Q}) return back to the inputs of NAND gates 3 and 4, the positive clock spike has died down to zero.
- Hence the multiple toggling cannot take place.
- Thus the edge triggering avoids the race around condition.

6.9.4 Negative Edge Triggered JK Flip-Flop :

Symbol :

- The symbol for negative edge triggered JK flip-flop is shown in Fig. 6.9.3.



(C-598) Fig. 6.9.3 : Symbol of negative edge triggered JK FF

- This FF responds only to the negative edges of the clock pulses.
- The rest of the operation of this FF is exactly same as that of the positive edge triggered JK FF.

Truth table :

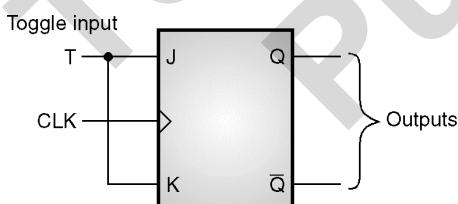
- The truth table of negative edge triggered JK FF is as shown in Table 6.9.3.

(C-6838) Table 6.9.3 : Truth table of negative edge triggered JK FF

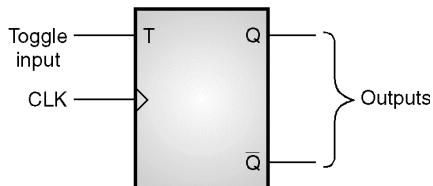
Inputs			Outputs		State
CLK	J	K	Q_{n+1}	\bar{Q}_{n+1}	
0 or 1	x	x	Q_n	\bar{Q}_n	NC (FF is disabled)
↑	x	x	Q_n	\bar{Q}_n	
↓	0	0	Q_n	\bar{Q}_n	No change
↓	0	1	0	1	Reset
↓	1	0	1	0	Set
↓	1	1	\bar{Q}_n	Q_n	Toggle

6.10 Toggle Flip Flop (T Flip Flop) :**6.10.1 Positive Edge Triggered T-FF :****Symbol :**

- Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only one input denoted by "T", as shown in Fig. 6.10.1(a).
- The symbol for positive edge triggered T flip flop is shown in Fig. 6.10.1(b) and Table 6.10.1 shows its truth table.



(a) JK FF is converted into T flip flop



(b) Logic symbol of positive edge triggered T flip flop

(C-599) Fig. 6.10.1

Truth table :

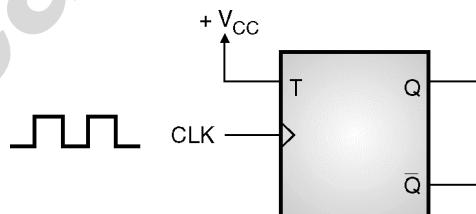
(C-6839) Table 6.10.1 : Truth table of a Toggle FF (positive edge triggered)

Inputs		Outputs		State
CLK	T	Q_{n+1}	\bar{Q}_{n+1}	
↑	0	Q_n	\bar{Q}_n	No change
↓	x	Q_n	\bar{Q}_n	No change
1	x	Q_n	\bar{Q}_n	No change
0	x	Q_n	\bar{Q}_n	No change
↑	1	\bar{Q}_n	Q_n	Toggle

Operation :

- When $T = 0$, $J = K = 0$. Hence the outputs Q and \bar{Q} won't change.
- But if $T = 1$, then $J = K = 1$ and the outputs will toggle corresponding to every leading edge of clock signal.
- This has been illustrated in Ex. 6.10.1.

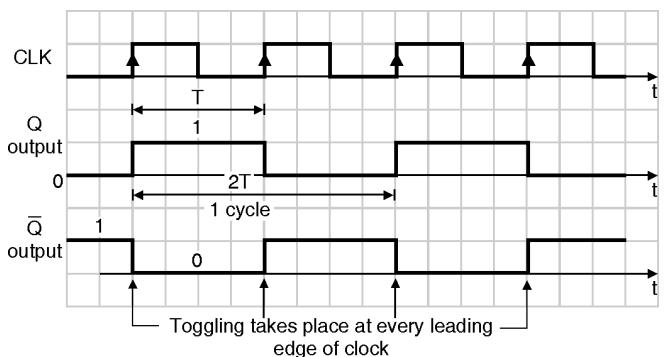
Ex. 6.10.1 : Determine the output Q for the set up shown in Fig. P. 6.10.1(a). What is the relation between the frequency of clock and that of Q output ?



(C-600) Fig. P. 6.10.1(a) : Given set up

Soln. :

- The flip flop of Fig. P. 6.10.1(a) is a toggle flip flop with $T = 1$. It is a positive edge triggered flip flop.
- Hence the outputs will toggle in response to each rising edge of the clock, as shown in Fig. P. 6.10.1(b).



(C-601) Fig. P. 6.10.1(b) : Waveforms for a T F/F



Relation between input and output frequency :

- Referring to Fig. P. 6.10.1(b),

$$1 \text{ cycle period of CLK signal} = T,$$

$$\therefore \text{Clock frequency } f_{\text{CLK}} = \frac{1}{T}$$

$$1 \text{ cycle period of Q output} = 2T,$$

$$\therefore \text{Output frequency } f_o = \frac{1}{2T}$$

$$\text{But } \left(\frac{1}{T}\right) = f_{\text{CLK}}$$

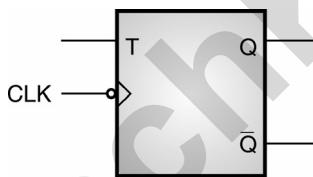
$$\therefore \text{Output frequency } f_o = \frac{f_{\text{CLK}}}{2} \quad \dots \text{Ans.}$$

The T flip flop divides the clock frequency by 2. Hence a T flip flop can be used as a frequency divider.

6.10.2 Negative Edge Triggered T Flip Flop :

Symbol :

- Fig. 6.10.2 shows the logic symbol of a negative edge triggered toggle flip flop and Table 6.10.2 gives its truth table.



(C-602) Fig. 6.10.2 : Logic symbol of a negative edge triggered toggle flip flop

Truth table :

(C-6840) Table 6.10.2 : Truth table

Inputs		Outputs		State
T	CLK	Q_{n+1}	\bar{Q}_{n+1}	
0	x	Q_n	\bar{Q}_n	No change
1	↑	Q_n	\bar{Q}_n	No change
1	0	Q_n	\bar{Q}_n	No change
1	1	Q_n	\bar{Q}_n	No change
1	↓	\bar{Q}_n	Q_n	Toggle

6.10.3 Application of T F/F :

- The T F/F acts as the basic building block of a ripple counter.

6.11 Master Slave (MS) JK Flip Flop :

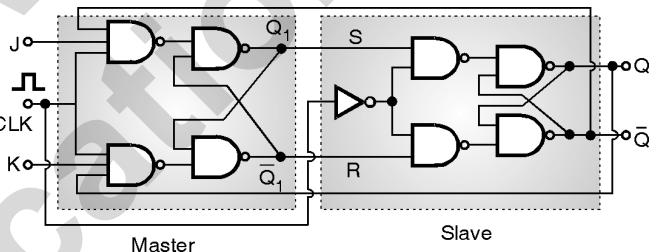
SPPU : Dec. 06, Dec. 10, May 14

University Questions

- Q. 1** What do you mean by Master-Slave JK Flip-flop ? Explain the advantage of this Flip-flop. Draw suitable circuit diagram and timing diagram.
(Dec. 06, 10 Marks)
- Q. 2** What is the advantage of M-S flip-flop ? Explain working of MS J-K flip-flop in detail.
(Dec. 10, 8 Marks)
- Q. 3** Draw and explain the behavior of M-S JK flip flop with waveform.
(May 14, 6 Marks)

Logic diagram :

- Fig. 6.11.1 shows the master slave JK flip flop.



(C-606) Fig. 6.11.1 : Master slave JK FF

- It is a combination of a clocked JK latch (level triggered JK FF) and clocked SR latch (level triggered SRFF).
- The clocked JK latch acts as the master and the clocked SR latch acts as the slave.
- Master is positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level.
- Thus both master and slave circuits are level triggered circuits.
- Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and the master is inactive.

Truth table :

- Table 6.11.1 gives truth table of master slave JK flip flop.

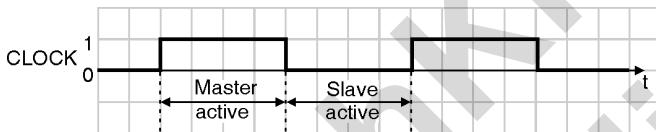


(C-6267) Table 6.11.1 : Truth table of master slave JK FF

Case	Inputs			Outputs		Remark
	CLK	J	K	Q_{n+1}	\bar{Q}_{n+1}	
I	X	0	0	Q_n	\bar{Q}_n	No change
II	$\square(1)$	0	0	Q_n	\bar{Q}_n	No change
III	$\square(1)$	0	1	0	1	Reset
IV	$\square(1)$	1	0	1	0	Set
V	$\square(1)$	1	1	\bar{Q}_n	Q_n	Toggle

Operation :

- We will discuss the operation of the master slave JK FF with reference to its truth table.
- We must always remember one important thing that in the positive half cycle of the clock, the master is active and in the negative half cycle, the slave is active.
- This is shown in Fig. 6.11.2.



(C-607) Fig. 6.11.2

Case I : Clock = X, J = K = 0 (No change)

- For clock = 1, the master is active, slave inactive. As $J = K = 0$. \therefore Outputs of master i.e. Q_1 and \bar{Q}_1 will not change. Hence the S and R inputs to the slave will remain unchanged.
- As soon as clock = 0, the slave becomes active and master is inactive.
- But since the S and R inputs have not changed, the slave outputs will also remain unchanged.
- \therefore The outputs will not change if $J = K = 0$.

Case II : Clock = $\square(1)$, J = K = 0 (No change) (C-6395)

This condition has been already discussed in case I.

Case III : Clock = $\square(1)$, J = 0 and K = 1 (Reset) (C-6395(a))

- Clock = 1 : Master active, slave inactive.

\therefore Outputs of the master become $Q_1 = 0$ and $\bar{Q}_1 = 1$. That means S = 0 and R = 1.

- Clock = 0 : Slave active, master inactive

\therefore Outputs of the slave become Q = 0 and $\bar{Q} = 1$. This is the RESET operation.

- Again if clock = 1 : Master active, slave inactive.

\therefore Even with the changed outputs Q = 0 and $\bar{Q} = 1$ fed back to master, its outputs will be $Q_1 = 0$ and $\bar{Q}_1 = 1$. That means S = 0 and R = 1.

- Hence with clock = 0 and slave becoming active, the outputs of slave will remain Q = 0 and $\bar{Q} = 1$.

- Thus we get a stable output from the Master slave.

Case IV : Clock = $\square(1)$, J = 1 and K = 0 (Set) (C-6395(b))

- Clock = 1 : Master active, slave inactive.

\therefore Outputs of master become $Q_1 = 1$ and $\bar{Q}_1 = 0$ i.e. S = 1, R = 0.

- Clock = 0 : Master inactive, slave active.

\therefore Outputs of slave become Q = 1 and $\bar{Q} = 0$.

- Again if clock = 1 then it can be shown that the outputs of the slave are stabilized to Q = 1 and $\bar{Q} = 0$.

Case V : Clock = $\square(1)$, J = 1 and K = 1 (Toggle without race) (C-6395(c))

- Clock = 1 : Master active, slave inactive.

\therefore Outputs of master will toggle. So S and R also will be inverted.

- Clock = 0 : Master inactive, slave active.

\therefore Outputs of the slave will toggle.

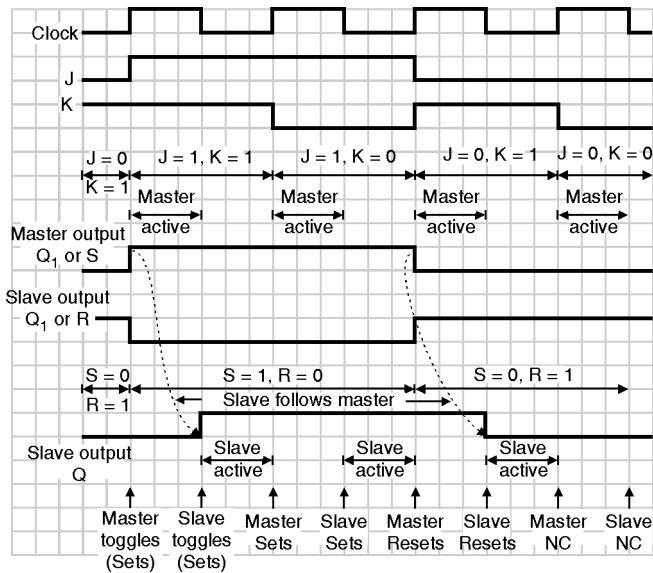
- These changed output are returned back to the master inputs.

- But since clock = 0, the master is still inactive. So it does not respond to these changed outputs.

- This avoids the multiple toggling which leads to the race around condition.

- Thus the master slave flip flop will avoid the race around condition.

- The waveforms for the master slave flipflop are shown in Fig. 6.11.3.



(C-608) Fig. 6.11.3 : Waveforms of master slave JK flip flop

Observations from the waveforms :

- We can make the following important observations from the waveforms of the master slave JK FF :

 1. The slave always follows the master, after a delay of half clock cycle period.
 2. The multiple toggling or the race around condition is successfully avoided.

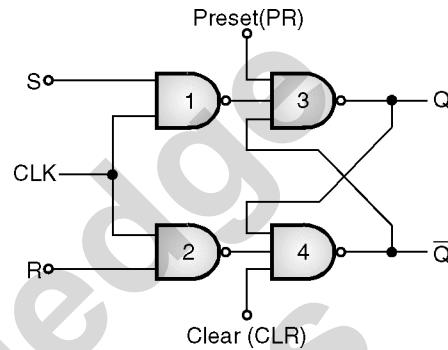
6.12 Preset and Clear Inputs :

- In the flip flops discussed so far when the power switch is turned on, the state of outputs Q and \bar{Q} can be anything that means either $Q = 0$, $\bar{Q} = 1$ or $Q = 1$, $\bar{Q} = 0$. It is not predictable.
- But this uncertainty cannot be tolerated in certain applications.
- In some applications it is necessary to initially set or reset the flip flops precisely.
- This can be practically achieved by adding two more inputs to a flip flop, called **preset (PR)** and **clear (CLR)** inputs.
- These inputs are called as **direct** or **asynchronous inputs** because we can apply them any time between clock pulses without thinking about their synchronization with the clock.
- That means applying PR or CLR inputs is not related to the clock in any way.

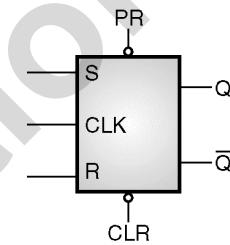
6.12.1 S-R Flip-Flop with Preset and Clear Inputs :

Logic diagram and symbol :

- The S-R flipflop with preset and clear is shown in Fig. 6.12.1(a) and its symbol is shown in Fig. 6.12.1(b).



(a) S-R FF with preset and clear



(b) Symbol

(C-609) Fig. 6.12.1

- Note that both these inputs are active low inputs. This is indicated by the bubbles on these inputs in Fig. 6.12.1(b).

Operation :

Case I : PR = CLR = 1

If PR = CLR = 1 then both are inactive and the S-R FF operates as per the truth table of the conventional SR flip-flop.

Case II : PR = 0 and CLR = 1 (Preset condition)

- As PR = 0, the output of gate 3 of Fig. 6.12.1(a) will be 1.
- That means $Q = 1$.
- Therefore all the three inputs to the gate 4 will be 1 and \bar{Q} will become 0.
- Thus making PR = 0 will set the flip flop. Note that with PR = 0 and CLR = 1 the flipflop is set irrespective of the status of S, R or clock inputs.



- Therefore it is said that the PR input has higher priority than S, R or CLK inputs.

Case III : PR = 1, CLR = 0 (Clear or Reset condition)

- If PR = 1 and CLR = 0, then the output of gate-4 i.e. $\bar{Q} = 1$.
- Therefore all the three inputs to gate-3 will be 1 and hence Q = 0.
- Thus making CLR = 0 will reset the flip flop. Note that with CLR = 0 and PR = 1, the FF is reset irrespective of the status of S, R or clock inputs.
- Therefore it is said that CLR has higher priority than S, R or CLK inputs.

Case IV : PR = 0, CLR = 0

- This condition should never be used, because it leads to an uncertain state.
- The operation of SR flip flop with preset and clear inputs is summarized in Table 6.12.1.
- The don't care conditions (X) marked in the CLK column indicate that PR and CLR inputs have higher priority than clock input.

(C-609(a))Table 6.12.1 : Summary of operation of SR FF

Inputs			Output	Operation performed
CLK	PR	CLR		
1	1	1	Q_{n+1}	Normal S-R flipflop
x	0	1	1	FF is set
x	1	0	0	FF is reset

6.12.2 Synchronous Preset and Clear Inputs :

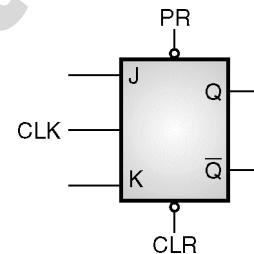
- Preset and clear inputs can be of two types :
 1. Asynchronous with clock.
 2. Synchronous with clock.
- We have already discussed the asynchronous preset and clear inputs.
- The operation of synchronous preset and clear inputs can be understood from the truth table given in Table 6.12.2.

(C-8075) Table 6.12.2 : Truth table for S-R FF with synchronous preset and clear

Inputs					Outputs	
P _r	C _r	CLK	S	R	Q_{n+1}	\bar{Q}_{n+1}
X	X	0	X	X	NC	NC
0	0	1	X	X	RACE	RACE
0	1	1	X	X	0	1
1	0	1	X	X	1	0
1	1	1	0	1	NC	NC
1	1	1	0	1	0	1
1	1	1	1	0	1	0
1	1	1	1	1	RACE	RACE

6.12.3 JK Flip Flop with Preset and Clear Inputs :

- A JK flip flop with preset and clear inputs is shown in Fig. 6.12.2. These are the synchronous preset and clear terminals.
- Both these inputs are active low and have higher priority than all the other inputs.



(C-610) (a) Symbol of a JK FF with preset and clear inputs

Inputs			Output	Operation performed
CLK	PR	CLR		
1	1	1	Q_{n+1}	Normal JK flipflop
x	0	1	1	FF is set
x	1	0	0	FF is reset

(C-610(a))(b) Summary of operation of JK FF

Fig. 6.12.2

6.12.4 Applications of JK Flip Flop :

1. Shift register.
2. Counters.

6.13 Various Representations of Flip Flops :

- There are different ways in which a flip flop can be represented.



- Each representation is suitable for a different application.
- Various representations of flip flops are :
 1. Characteristic equations.
 2. Flip flop as Finite State Machine (FSM).
 3. Flip flop excitation tables.

6.13.1 Characteristic Equations :

- We have already introduced the concept of characteristic equations earlier in this chapter and we have written the characteristic equations for various flip flops.

6.14 Excitation Table of Flip-Flop :

- Till now we have written the truth tables of various flip flops.
- The truth tables are also known as the **characteristic tables**.
- But while designing the sequential circuits, sometimes the present and next state of a circuit are given and we are expected to find the corresponding input condition.
- We need to use the **excitation tables** of flipflops to do this. These tables are different from the characteristic tables.
- Present state is the state prior to application of clock pulse and the next state means the state after application of clock pulse.

6.14.1 Excitation Table of SR Flip Flops :

- For example the outputs of an S-R FF before clock pulse are $Q_n = 0$ and $\bar{Q}_n = 1$ and it is expected that these outputs should remain unchanged after application of clock. Then what must be the values of inputs S_n and R_n to achieve this ?
- Refer to the truth table of SR FF to answer this question.
- The answer is, for the following two conditions the outputs remain unchanged at $Q = 0$ and $\bar{Q} = 1$.

(C-6571) Table 6.14.1 : Truth table of SR flip flop

	CLK	S_n	R_n	Q_{n+1}	\bar{Q}_{n+1}
Row 1	↑	0	0	Q_n	\bar{Q}_n
Row 2	↑	0	1	0	1
	↑	1	0	1	0
	↑	1	1	Race	Race

Condition 1 :

S_n and $R_n = 0 \rightarrow$ Refer first row of Table 6.14.1.

Condition 2 :

$S_n = 0 R_n = 1 \rightarrow$ Refer second row of Table 6.14.1.

- From the two conditions mentioned above we conclude that S_n input should be equal to 0 and R_n input can be 0 or 1 (don't care) if we want to maintain $Q = 0$ and $\bar{Q} = 1$ before and after clock.
- Similarly we can find the input conditions (Values of S and R inputs) for all the possible situations that may exist on the output side.
- The table containing all these output situations and the corresponding input conditions is called as the "excitation table" of a flip flop.
- The excitation table of SR flip flop is shown in Table 6.14.2.

C-6581) Table 6.14.2 : Excitation table of SR flip flop

	Present state of Q output	Next state of Q output	S_n input	R_n input
Case I	0	0	0	X
Case II	0	1	1	0
Case III	1	0	0	1
Case IV	1	1	X	0

Description of excitation table of SR FF :

We have already discussed case I.

Case II : Q should change from 0 to 1

- This is nothing but the set condition.
- Hence $S_n = 1$ and $R_n = 0$ should be the inputs.

Case III : Q should change from 1 to 0

- This is nothing but the reset condition.
- Hence S_n should be 0 and R_n should be 1.

Case IV : Q should be 1. No change

- To satisfy this requirement, we have two possible input conditions :

Condition 1 : $S_n = R_n = 0$ i.e. No change in output.

Condition 2 : $S_n = 1$ and $R_n = 0$.

- From these conditions we conclude that S_n can be either 0 or 1 i.e. don't care and $R_n = 0$.
- Hence the inputs corresponding to this case is $S_n = \times$ and $R_n = 0$.
- Similarly we can write the excitation tables for the other flip flops.

6.14.2 Excitation Table of D Flip Flop :

- Excitation table of a D flip flop is given by Table 6.14.3.

(C-7828) Table 6.14.3 : Excitation table of a D flip flop

Q output		Input
Present state	Next state	D_n
0	0	0
0	1	1
1	0	0
1	1	1

6.14.3 Excitation Table of JK Flip Flop :

- Excitation table of a JK flip flop is given by Table 6.14.4.

(C-6211) Table 6.14.4 : Excitation table of JK flip flop

	Q output		Inputs	
	Present state	Next state	J_n	K_n
Case I	0	0	0	\times
Case II	0	1	1	\times
Case III	1	0	\times	1
Case IV	1	1	\times	0

Don't care conditions result due to toggling

- Refer to Case II of Table 6.14.4. For the change in Q output from 0 to 1 we have the following two input conditions :

Condition I :

$J_n = 1$ and $K_n = 0$ i.e. set condition.

Condition II :

$J_n = 1$ and $K_n = 1$ i.e. toggle.

- Hence $J_n = 1$ and $K_n = \times$ (0 or 1) corresponding to case II.
- Similarly for case III also, the don't care condition corresponds to toggle mode.

6.14.4 Excitation Table of T Flip Flop :

- Excitation table of T flip flop is shown in Table 6.14.5.

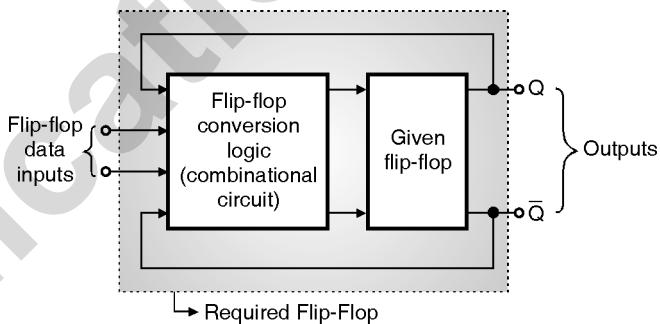
(C-7829) Table 6.14.5 : Excitation table of T flip-flop

Q output		Input
Present state	Next state	T_n
0	0	0
0	1	1
1	0	1
1	1	0

6.15 Conversion of Flip Flops :

Concept :

- The conversion from one type of flip flop to the other (say SR FF to JKFF), needs a systematic approach using the excitation tables and K map simplifications.
- Fig. 6.15.1 shows a generalized model for conversion from one flip flop to the other.



(C-611) Fig. 6.15.1 : General model used to convert one type of FF to the other

- As shown in Fig. 6.15.1 the required flip flop is actually a combination of the given flip flop and a combinational logic circuit using gates. (Such a combinational circuit is called as the flip flop conversion logic).
- The inputs to FF conversion logic are, the flip flop data inputs and the outputs of given flip-flop i.e. the given FF and the desired FF.
- The conversion logic is designed by combining the excitation tables of both the flip flops.
- The truth table of the conversion logic has data inputs and Q and \bar{Q} outputs of the given FF as inputs whereas the inputs of the given FF are the outputs of the truth table.



- Then we draw the K map for each output and obtain the simplified expressions. The conversion logic is then implemented using gates.

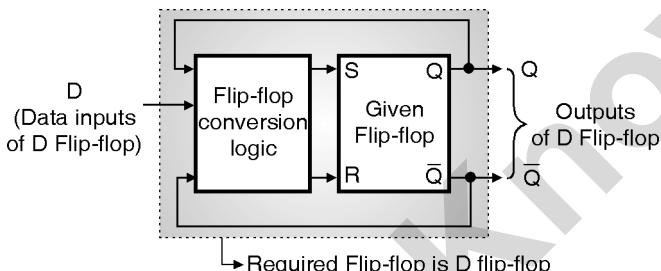
6.15.1 Conversion from S-R Flip Flop to D Flip Flop :

SPPU : May 12

University Questions

Q. 1 Convert the basic SR-flip-flop (SR-FF) into D FF.
(May 12, 2 Marks)

- Refer Fig. 6.15.2. Here the given FF is SR FF and the required FF is D FF.
- The truth table for the conversion logic is shown in Table 6.15.1. The inputs are D and Q whereas outputs are S and R.
- The truth table is prepared by combining the excitation tables of D F/F and SR FF.



(C-612) Fig. 6.15.2

(C-6387) Table 6.15.1 : Truth table for SR to D FF conversion

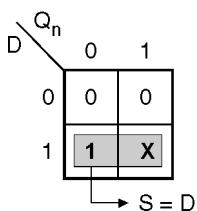
Inputs			Outputs	
D	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	X
1	0	1	1	0
0	1	0	0	1
1	1	1	X	0

Entries from excitation table of D FF →

Entries from excitation table of SR FF →

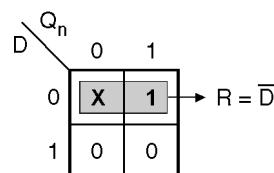
- Now write the K maps for the S and R outputs as shown in Figs. 6.15.3(a) and (b).

For S output



(a) K map for S

For R output

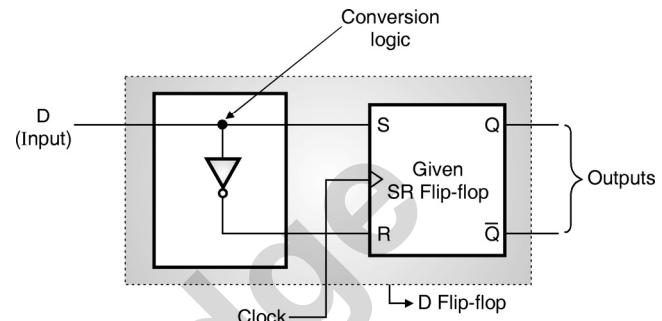


(b) K map for R

(C-613) Fig. 6.15.3

Logic diagram :

- The logic diagram for SR FF to D FF is shown in Fig. 6.15.4.



(C-614) Fig. 6.15.4 : SR flip flop to D flip flop

6.15.2 Conversion of JK FF to T FF :

SPPU : Dec. 09, May 17

University Questions

Q. 1 Convert J-K flip-flop into T-FF. Show the truth table.
(Dec. 09, 4 Marks)

Q. 2 Design flip-flop conversion logic to convert JK flip-flop to T flip-flop.
(May 17, 6 Marks)

Step 1 : Write the truth table for conversion :

- The required truth table is obtained from the excitation tables of JK and T flip flops as follows :

(C-6388) Table 6.15.2 : Truth table for JK to T FF conversion

Inputs			Outputs	
T	Present state of Q	Next state of Q	J	K
0	0	0	0	X
1	0	1	1	X
1	1	0	X	1
0	1	1	X	0

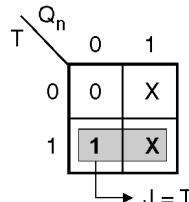
← Excitation table of T FF →

← Excitation table of JK FF →

Step 2 : K maps and simplification :

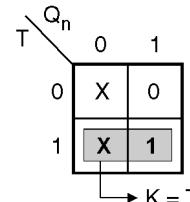
- The K maps for outputs J and K are shown in Fig. 6.15.5.

For J output



(a) K map for J output

For K output

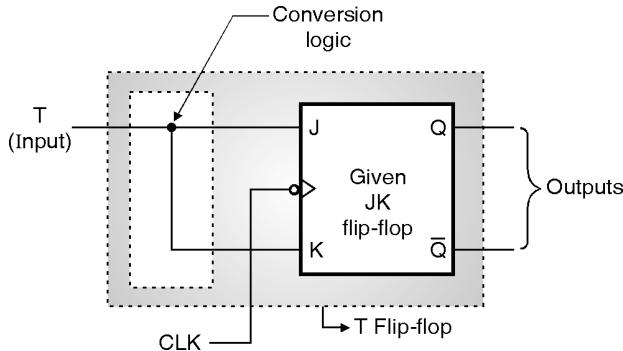


(b) K map for K output

(C-615) Fig. 6.15.5

Step 3 : Draw the logic diagram :

- The logic diagram is shown in Fig. 6.15.6.



(C-616) Fig. 6.15.6 : Logic diagram for conversion of JK FF to T FF

6.15.3 SR Flip Flop to T Flip Flop :

SPPU : May 12, Dec. 16, Dec. 18

University Questions

- Q. 1** Convert the basic SR-flip-flop (SR-FF) into T-FF
(May 12, 2 Marks)
- Q. 2** Explain the difference between asynchronous and synchronous counter and convert SR flip-flop into T flip-flop. Show the design. **(Dec. 16, 6 Marks)**
- Q. 3** Design and implement T flip-flop using SR flip-flop.
(Dec. 18, 6 Marks)

- The stepwise conversion process is as follows :

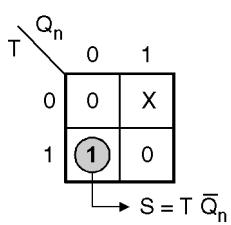
Step 1 : Write the truth table :

(C-8069) Table 6.15.3 : Truth table for SR FF to T FF

Inputs		Outputs		
T	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	X
1	0	1	1	0
1	1	0	0	1
0	1	1	X	0

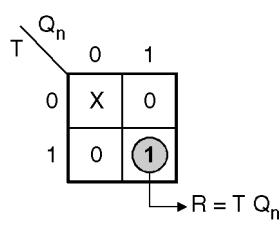
Step 2 : Write the K maps and obtain the expressions for S and R :

For S output



(a) K map for S

For R output

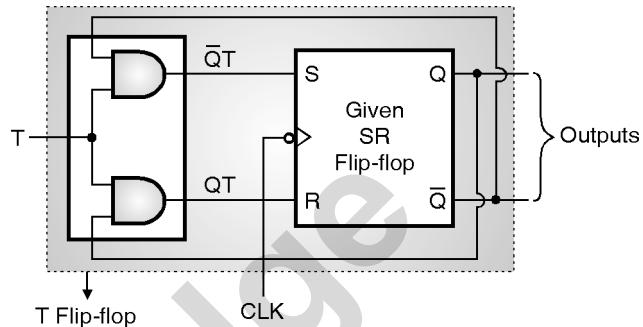


(b) K map for R

(C-617) Fig. 6.15.7

Step 3 : Draw the logic diagram :

- The logic diagram is shown in Fig. 6.15.8.



(C-618) Fig. 6.15.8 : Conversion from SR flip flop to T flip flop

6.15.4 SR Flip Flop to JK Flip Flop :

SPPU : May 12, Dec. 14, May 15

University Questions

- Q. 1** Convert the basic SR-flip-flop (SR-FF) into JK-FF.
(May 12, 2 Marks)
- Q. 2** Design JK flip-flop using SR flip-flop.
(Dec. 14, 6 Marks)
- Q. 3** How will you convert the basic SR-flip-flop (SR-FF) into JK flip-flop ?
(May 15, 6 Marks)

Step 1 : Write the truth table for SR to JK :

- The truth table for SR to JK flip flop conversion is shown in Table 6.15.4.

(C-6389) Table 6.15.4 : Truth table for SR to JK FF conversion

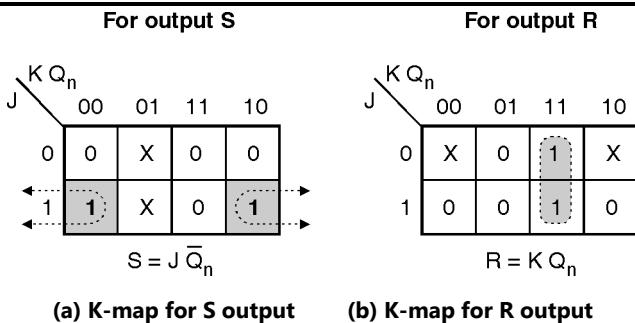
Inputs			Outputs		
J	K	Present state Q_n	Next state Q_{n+1}	S	R
0	0	0	0	0	X
0	1	0	0	0	X
1	0	0	1	1	0
1	1	0	1	1	0
0	1	1	0	0	1
1	1	1	0	0	1
0	0	1	1	X	0
1	0	1	1	X	0

Excitation table of JK FF

Excitation table of SR FF

Step 2 : K maps and simplification :

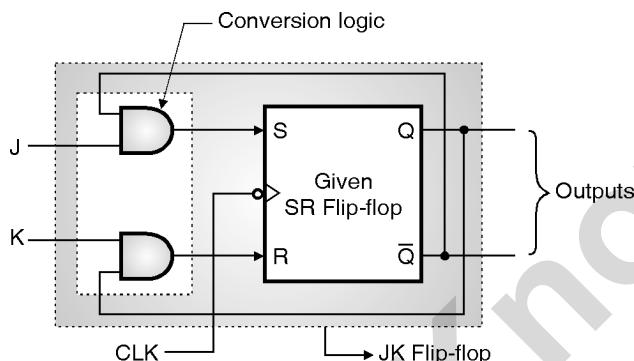
- K maps for S and R outputs are shown in Fig. 6.15.9.



(C-619) Fig. 6.15.9

Step 3 : Logic diagram :

- The logic diagram of SR to JK flip flop is given in Fig. 6.15.10.



(C-620) Fig. 6.15.10 : SR to JK flip flop conversion

6.15.5 Conversion of D Flip Flop to T Flip Flop :

SPPU : Dec. 13

University Questions

- Q. 1** What is race around condition ? How it can be avoided ? Convert D flip-flop to T flip-flop.
(Dec. 13, 6 Marks)

Step 1 : Write the truth table for D to T FF conversion :

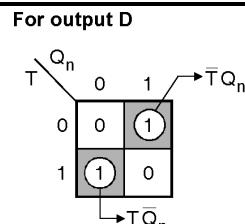
- The truth table is as follows :

(C-8044) Table 6.15.5 : Truth table for D to T FF conversion

Inputs		Outputs	
T	Present state Q _n	Next state Q _{n+1}	D
0	0	0	0
1	0	1	1
1	1	0	0
0	1	1	1

Step 2 : K map and simplification and logic diagram :

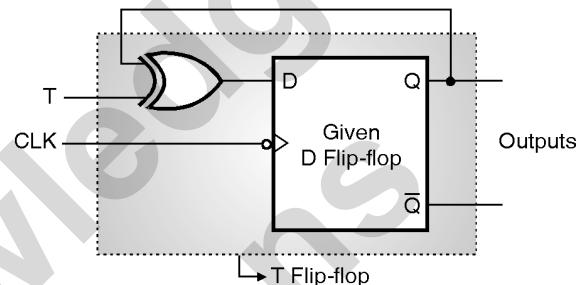
- K map is shown in Fig. 6.15.11(a) and logic diagram is shown in Fig. 6.15.11(b).



$$\therefore D = T \bar{Q}_n + \bar{T} Q_n$$

$$\therefore D = T \oplus Q_n$$

(a) K map and simplification



(b) Logic diagram

(C-621) Fig. 6.15.11 : D flip flop to T flip flop

6.15.6 T Flip Flop to D Flip Flop Conversion :**Step 1 : Write the truth table :**

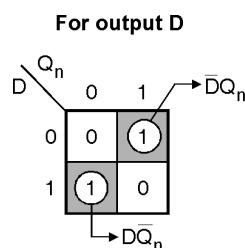
- The truth table is as given in Table 6.15.6.

(C-7831) Table 6.15.6 : Truth table for T FF to D FF conversion

Inputs		Outputs	
D	Previous state Q _n	Next state Q _{n+1}	T
0	0	0	0
1	0	1	1
0	1	0	1
1	1	1	0

Step 2 : K maps simplification and logic diagram :

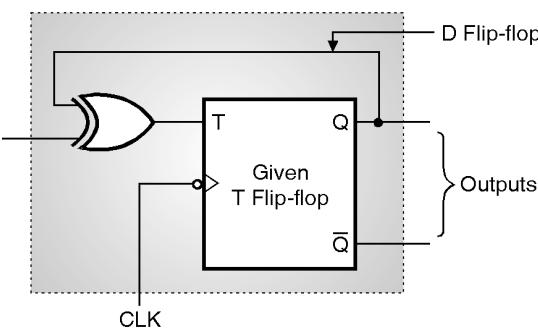
- The K map is shown in Fig. 6.15.12(a) and the logic diagram is given in Fig. 6.15.12(b).



$$\therefore T = D \bar{Q}_n + \bar{D} Q_n$$

$$\therefore T = D \oplus Q_n$$

(C-622) Fig. 6.15.12(a) : K map for D output



(C-622) Fig. 6.15.12(b) : Logic diagram of T to D flip flop conversion

6.15.7 JK Flip Flop to D Flip Flop Conversion :

SPPU : Dec. 09, May 14

University Questions

- Q. 1** Convert J-K flip-flop into D-FF. Show the truth table. **(Dec. 09, 4 Marks)**
- Q. 2** Explain the difference between asynchronous and synchronous counter and convert J-K flip flop into D-FF. Show the design. **(May 14, 6 Marks)**

Step 1 : Write the truth table for JK to D conversion :

- The truth table is as follows :

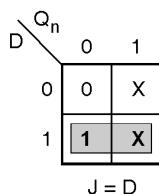
(C-6390) Table 6.15.7 : Truth table for JK to D flip flop conversion

Inputs			Outputs	
D	Previous state Q_n	Next state Q_{n+1}	J	K
0	0	0	0	X
1	0	1	1	X
0	1	0	X	1
1	1	1	X	0

← Excitation table of D FF →
← Excitation table of JK FF →

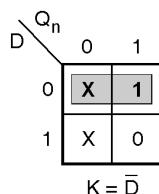
Step 2 : K maps and simplification :

For J output



(a) K map for J output

For K output

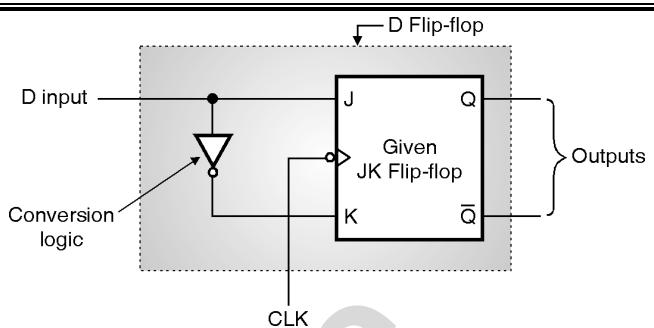


(b) K map for K output

(C-623) Fig. 6.15.13

Step 3 : Draw the logic diagram :

- The logic diagram for JK flip flop to D flip flop is shown in Fig. 6.15.14.



(C-624) Fig. 6.15.14 : Logic diagram for conversion from JK FF to D FF

6.15.8 JK Flip Flop to SR Flip Flop Conversion :

Conversion : SPPU : May 10, Dec. 13, Dec. 19

University Questions

- Q. 1** Design SR flip-flop using JK flip-flop. **(May 10, 4 Marks)**
- Q. 2** Explain the difference between combinational and sequential circuit. Design S-R flip-flop using J-K flip-flop. **(Dec. 13, 6 Marks)**
- Q. 3** Compare combinational circuits with sequential circuits. Convert JK Flip-Flop into SR flip-flop. **(Dec. 19, 6 Marks)**

Step 1 : Write the truth table for JK to SR :

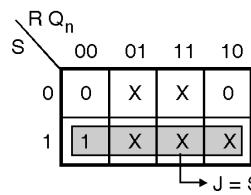
- The truth table is shown in Table 6.15.8.

(C-8045) Table 6.15.8 : Truth table for JK to SR conversion

Inputs				Outputs	
S	R	Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	0	0	X
0	1	0	0	0	X
1	0	0	1	1	X
1	0	0	1	1	X
0	1	1	0	X	1
0	1	1	0	X	1
0	0	1	1	X	0
1	0	1	1	X	0

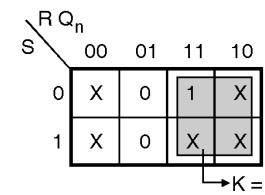
Step 2 : K-maps and simplifications :

For J output



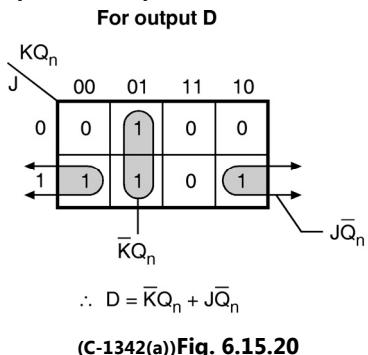
(a)

For K output

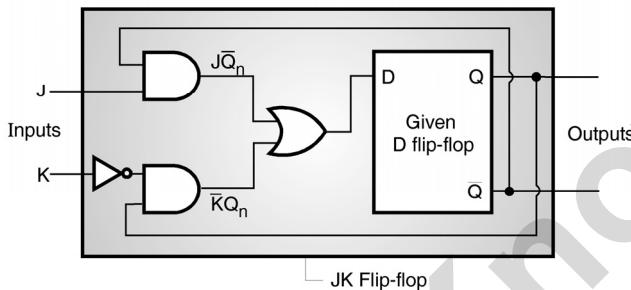


(b)

(C-625) Fig. 6.15.15

**Step 2 : K maps and simplification :****Step 3 : Logic diagram :**

- The logic diagram for D FF to JK FF is shown in Fig. 6.15.21.

**6.16 Applications of Flip Flops :**

- Some of the important applications of the flip flops are :
 - Elimination of keyboard debounce.
 - As a memory element.
 - In various types of registers.
 - In counters/timers.
 - As a delay element.

6.17 Study of Flip-Flop ICs :**6.17.1 SN74LS74A : Dual D-Type Positive Edge-triggered Flip-Flop Low Power Schottky :****Description :**

- The SN54LS/74LS74A is a dual edge-triggered flip-flop which utilizes Schottky TTL circuitry to produce high speed D-type flip-flops. Each flip-flop has individual clear and set inputs, as well as complementary Q and \bar{Q} outputs.

Mode select – truth table

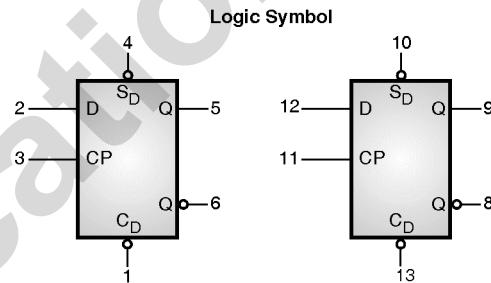
Operating Mode	Inputs		Outputs		
	\bar{S}_D	\bar{C}_D	D	Q	\bar{Q}
Set	L	H	X	H	L
Reset (Clear)	H	L	X	L	H
*Undetermined	L	L	X	H	H
Load "1" (Set)	H	H	h	H	L
Load "0" (Reset)	H	H	/	L	H

H, h = HIGH Voltage Level

L, l = LOW Voltage Level

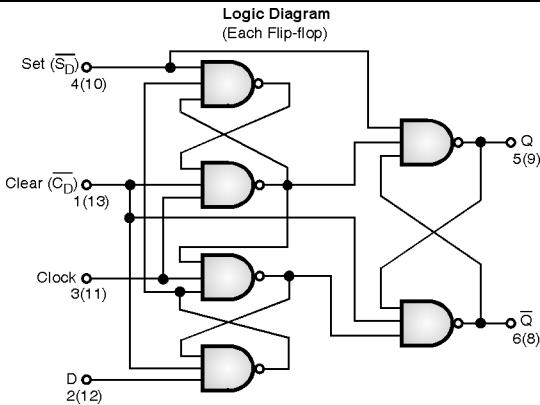
X = Don't care

- i, h (q) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.



(C-1722) Fig. 6.17.1

- Information applied at input D is transferred to the Q output on the positive-going edge of the clock pulse. Clock triggering occurs at a voltage level of the clock pulse and is not directly related to the transition time of the positive-going pulse. When the clock input is at either the HIGH or the LOW level, the D input signal does not have any effect on the Q and \bar{Q} outputs.
- Now consider the mode select truth table indicates that both outputs will be HIGH while both \bar{S}_D AND \bar{C}_D are LOW, but the output states are unpredictable if \bar{S}_D and \bar{C}_D go HIGH simultaneously. If the levels at the set and clear are near V_{IL} maximum then we cannot guarantee to meet the minimum level for V_{OH} .



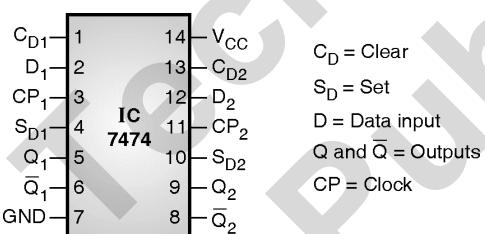
(C-3624) Fig. 6.17.2

Guaranteed operating ranges :

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54	4.5	5.0	5.5	V
		74	4.75	5.0	5.25	
T_A	Operating Ambient Temperature Range	54	-55	25	125	°C
		74	0	25	70	
I_{OH}	Output Current – High	54,			-0.4	mA
		74				
I_{OL}	Output Current – Low	54			4.0	mA
		74			8.0	

Pin configuration :

- Fig. 6.17.3 shows the pin configuration of IC 7474.



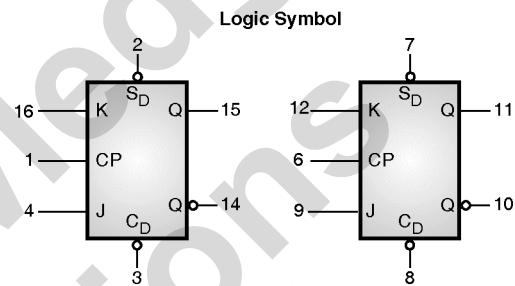
(C-3625) Fig. 6.17.3 : Pin configuration of IC 7474

6.17.2 SN74LS76A : Dual JK Flip-Flop with Set and Clear Low Power Schottky :**Description :**

- The SN54LS/74LS76A offers individual J, K, Clock Pulse, Direct Set and Direct Clear inputs. These dual flip-flops are designed so that when the clock goes HIGH, the inputs are enabled and data will be accepted.
- The logic Level of the J and K inputs will perform according to the truth table as long as minimum set-up times are observed. Input data is transferred to the outputs on the HIGH-to-LOW clock transitions.

Mode Select – Truth table

Operating Mode	Inputs			Outputs		
	\bar{S}_D	\bar{C}_D	J	K	Q	\bar{Q}
Set	L	H	X	X	H	L
Reset (Clear)	H	L	X	X	L	H
*Undetermined	L	L	X	X	H	H
Toggle	H	H	H	h	\bar{q}	q
Load "0" (Reset)	H	H	L	h	L	H
Load "1" (Set)	H	H	H	l	H	L
Hold	H	H	L	l	q	\bar{q}



V_{CC} = Pin 5
GND = Pin 13
J Suffix - Case 620 - 08 (Ceramic)
N Suffix - Case 648 - 05 (Plastic)

(C-1723) Fig. 6.17.4

- Both outputs will be HIGH while both \bar{S}_D and \bar{C}_D are LOW, but the 7 output states are unpredictable if \bar{S}_D and \bar{C}_D go HIGH simultaneously.

H, h = HIGH Voltage Level

L, l = LOW Voltage Level

X = immaterial

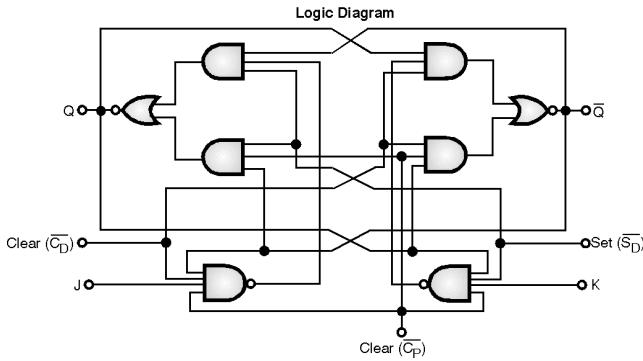
- $l, h (q) =$ Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the HIGH-to-LOW clock transition.

Guaranteed operating ranges :

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54	4.5	5.0	5.5	V
		74	4.75	5.0	5.25	
T_A	Operating Ambient Temperature Range	54	-55	25	125	°C
		74	0	25	70	
I_{OH}	Output Current – High	54,			-0.4	mA
		74				
I_{OL}	Output Current – Low	54			4.0	mA
		74			8.0	



- Logic diagram and pin configuration of IC 7476 is as shown in Fig. 6.17.5.



(a) Logic diagram

	1	16	K ₁
S _{D1}	2	15	Q ₁
C _{D1}	3	14	Q̄ ₁
J ₁	4	13	GND
V _{CC}	5	12	K ₂
CP ₂	6	11	Q ₂
S _{D2}	7	10	Q̄ ₂
C _{D2}	8	9	J ₂

(b) Pin configuration

(C-3626) Fig. 6.17.5

6.18 Analysis of Clocked Sequential Circuits :

- The behaviour of a clocked sequential circuit is dependent on the inputs, the outputs and the state of its flip-flops.
- The outputs as well as the next state both are function of inputs and present state.
- The analysis of the given clocked sequential circuit includes writing the state table and drawing the state diagram for the given circuit.
- In this section, we will introduce the concepts such as state diagram, state table, state equation and input equations and the step by step analysis procedure.

6.18.1 State Table :

- We use the truth table for explaining the relation between its inputs and outputs of a combinational circuit.
- But it is not suitable to use the truth table to describe the input output relation of a sequential circuit.
- Instead we use the state table to describe the input output relation of the sequential circuits.

- The basic building block of a sequential circuit is a flip flop. The outputs Q_A, Q_B ... etc. of such flip flops will be used as **state inputs** to a sequential circuit. They are also called as **state variable**.
- In addition to this "x" represents an external input and Y represents the output of the sequential circuit.
- Y will be dependent on the state variables (Q_A, Q_B, ...) and the external input x.
- The general format of a state table is shown in Table 6.18.1.

(C-7833) Table 6.18.1 : General format of state table

Present state	Next state		Output Y		
	x = 0	x = 1	x = 0	x = 1	
Q _A	Q _B	Q _A	Q _B	Q _A	Q _B
0	0	0	0	0	1
0	1	1	1	0	1
1	0	1	0	0	0
1	1	1	0	1	1

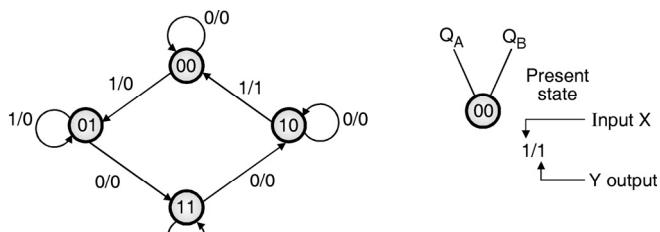
Definition :

- The state table is defined as the table that tells us about the relation between the present state, next state, external inputs and output of a sequential circuit.

6.18.2 State Diagram :

Definition :

- A state diagram is the graphical representation of a sequential circuit, which consists of states, state transitions and actions.
- State diagrams depict the permitted states and transitions as well as the events that effect these transition.
- The information available in the state table is represented graphically using the state diagram.
- The state diagram is drawn by using the state table as a reference. Such a state diagram is shown in Fig. 6.18.1.



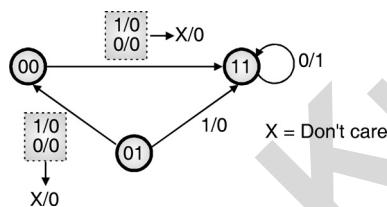
(C-653) Fig. 6.18.1 : State diagram



- The circle represents the present state. The arrows between the circles define the state transition say from 00 to 01 or 01 to 11.
- If there is directed line connecting the same circle then it means that the next state is same as the present state.
- The lines joining the circles are labeled with a pair of binary numbers with a “ / ” in between. For example the line joining 00 and 01 is labeled as 1/0.
- Note that 00 to 01 transition takes place when $x = 1$ and $y = 0$ (see row-1 of the state table). Hence 1 in 1/0 corresponds to x and 0 corresponds to y .

Don't care condition in the state diagram :

- Sometimes the same next state is reached for more than one present states.
- This is called as don't care condition in the state diagram, as shown in Fig. 6.18.2.



(C-654) Fig. 6.18.2 : Don't care condition in state diagram

6.18.3 State Equation :

Definition :

- State equation is an algebraic equation. The left side of this equation represents the next state of the flip flops.
- And the right hand side of this equation specifies the present state conditions which make the next state equal to 1.
- For example consider the following state expression.

(C-6235)

$$Q_{A(n+1)} = (\bar{Q}_A Q_B + Q_A \bar{Q}_B + Q_A Q_B)x + Q_A \bar{Q}_B x \dots$$

Next state
Combinations of present state and inputs which make the next state equal to 1.

- State equation is also called as **application equation**.
- We can derive the state equation from the state table using the K maps. The state equation is the Boolean function with time included into it.
- If the right hand side part is zero and we apply a clock pulse, then the next state will be equal to zero.

Ex. 6.18.1 : For the clocked D FF write the state table, draw the state diagram and write the state equation.

Soln. :

Step 1 : Write the truth table :

- Table P. 6.18.1(a) represents the truth table for a clocked D flip flop.

(C-6212) Table P. 6.18.1(a) : Truth table of a clocked D FF

Inputs		Outputs	
CLK	D	Q_{n+1}	\bar{Q}_{n+1}
0	X	Q_n	\bar{Q}_n
1	X	Q_n	\bar{Q}_n
↓	X	Q_n	\bar{Q}_n
↑	0	0	1
↑	1	1	0

No change in output
Q output follows

Step 2 : Write the state table :

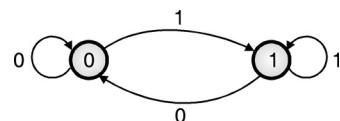
- Table P. 6.18.1(b) represents the state table for the clocked D FF.

(C-7834) Table P. 6.18.1(b) : State table of a clocked D FF

Present state Q_n	Next state Q_{n+1}	
	D = 0	D = 1
0	0	1
1	0	1

Step 3 : State diagram :

- State diagram of clocked D FF is shown in Fig. P. 6.18.1.



(C-659(a)) Fig. P. 6.18.1 : State diagram of a clocked D FF

Step 4 : Write excitation table :

- The excitation table for D FF is given in Table P. 6.18.1(c).

(C-7835) Table P. 6.18.1(c) : Excitation table for D FF

Present state Q_n	Next state Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

**Step 5 : Write the state equation :**

- The K-map for output Q_{n+1} is shown in Fig. P. 6.18.1(a).

For Q_{n+1}

D	Q_n	0	1
0	0	0	
1	1	1	

∴ State equation
 $Q_{n+1} = D$

(C-656) Fig. P. 6.18.1(a) : K-map and state equation

Ex. 6.18.2 : For the clocked JK FF write the state table, draw the state diagram and write the state equation.

Soln. :**Step 1 : Write the truth table :**

- Table P. 6.18.2(a) represents the truth table for a clocked JK flip flop.

(C-6213) Table P. 6.18.2(a) : Truth table of JK FF

Inputs			Output
CLK	J	K	Q_{n+1}
0	X	X	Q_n (NC)
1	X	X	Q_n (NC)
↓	X	X	Q_n (NC)
↑	0	0	Q_n (NC)
↑	0	1	0
↑	1	0	1
↑	1	1	\bar{Q}_n

No change in output

Step 2 : Write the state table :

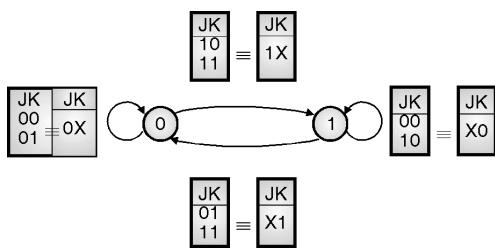
- Table P. 6.18.2(b) represents the state table for a clocked JK FF.

(C-8070) Table P. 6.18.2(b) : State table of clocked JK FF

Present state Q_n	Next state Q_{n+1}			
	JK = 00	JK = 01	JK = 10	JK = 11
0	0	0	1	1
1	1	0	1	0

Step 3 : Draw the state diagram :

- State diagram is as shown in Fig. P. 6.18.2(a).



(C-657) Fig. P. 6.18.2(a) : State diagram of a JK flip flop

Step 4 : Write the excitation table :

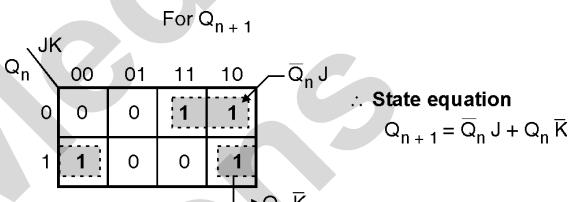
- The excitation table of JK FF is as shown in Table P. 6.18.2(c).

(C-7840) Table P. 6.18.2(c) : Excitation table for JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Step 5 : Write the state equation :

- The K-map for output Q_{n+1} is shown in Fig. P. 6.18.2(b).



(C-658) Fig. P. 6.18.2(b) : K-map state equation

Ex. 6.18.3 : For a toggle FF write the state table, draw the state diagram and write the state equation.

Soln. :**Step 1 : Write the truth table :**

- Table P. 6.18.3(a) gives the truth table for a T FF.

(C-8071) Table P. 6.18.3(a) : Truth table for a T FF

CLK	T	Q_{n+1}
0	X	Q_n
1	X	Q_n
↑	X	Q_n
↓	0	Q_n
↓	1	\bar{Q}_n

Step 2 : Write the state table :

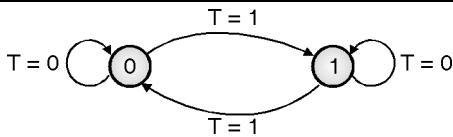
- The state table is shown in Table P. 6.18.3(b).

(C-8072) Table P. 6.18.3(b) : State table for a T FF

Present state Q_n	Next state Q_{n+1}	
	T = 0	T = 1
0	0	1
1	1	0

Step 3 : Draw the state diagram :

- The state diagram is shown in Fig. P. 6.18.3(a).



(C-659) Fig. P. 6.18.3(a) : State diagram of a T FF

Step 4 : Write the excitation table :

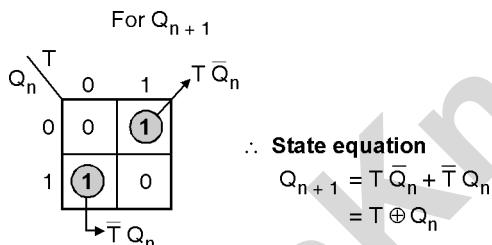
- Table P. 6.18.3(c) represents the excitation table of T FF.

(C-7842) Table P. 6.18.3(c) : Excitation table for T FF

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 5 : Write the state equation :

- The K-map for Q_{n+1} output is shown in Fig. P. 6.18.3(b). The state equation can be obtained from this K-map.



(C-660) Fig. P. 6.18.3(b)

6.19 Design of Clocked Synchronous State Machine using State Diagram :

- Design of a clocked sequential circuits will start from the set of given specifications and it will end with drawing a logic diagram.
- The stepwise design procedure is as follows :

Step 1 : A state diagram or timing diagram or some other information is given, which describes the behaviour of the circuit that is to be designed.

Step 2 : Draw the state table.

Step 3 : The number of states can be reduced by state reduction methods.

Step 4 : Assign binary values to each state for the states in steps 2 and 3 using state assignment technique.

Step 5 : Determine the number of flip-flops required and assign a letter symbol to each one.

Step 6 : Decide the type of FF to be used.

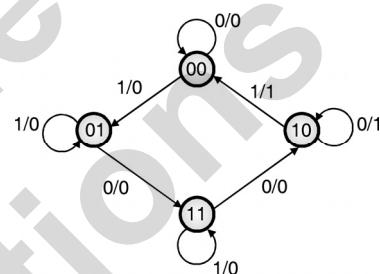
Step 7 : Derive the circuit excitation table and output table from the state table.

Step 8 : Obtain the expressions for the circuit output and flip-flop inputs.

Step 9 : Draw the logic diagram.

- The design procedure of clocked sequential circuits will be clear by solving the following example.

Ex. 6.19.1 : For the state diagram given in Fig. P. 6.19.1(a) draw the clocked sequential circuit using T flip-flops.



(C-685) Fig. P. 6.19.1(a) : Given state diagram

Soln. :

Step 1 : Write the state table :

- The state table is written from the given state diagram as shown in Table P. 6.19.1(a).

(C-7837) Table P. 6.19.1(a) : State table

Present state	Next state				Output	
	X = 0		X = 1		X = 0	X = 1
A	B	A _{n+1}	B _{n+1}	A _{n+1}	B _{n+1}	
0	0	0	0	0	1	0
0	1	1	1	0	1	0
1	0	1	0	0	0	1
1	1	1	0	1	1	0

Step 2 : Number of flip-flops :

- As seen from the state table, there are no equivalent states. So there won't be any reduction in the state diagram.
- The circuit goes through four states. Hence we need to use 2 flip-flops.

Step 3 : Write the circuit excitation table :

- The circuit excitation table is shown in Table P. 6.19.1(b). The type of FF used is T type FF.



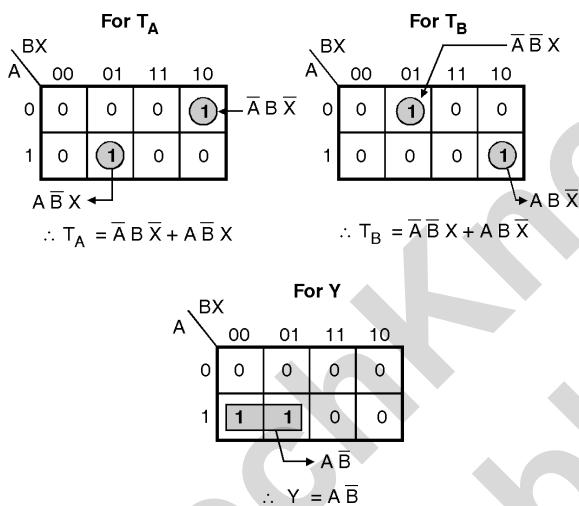
(C-7760) Table P. 6.19.1(b) : Circuit excitation table

Present state		Input X	Next state		FF Inputs		Output Y
A	B		A _{n+1}	B _{n+1}	T _A	T _B	
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	1	0	0	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0
1	1	1	1	1	0	0	0

- The shaded portion of the circuit excitation table corresponds to the excitation table of a T FF.

Step 4 : K maps and simplifications :

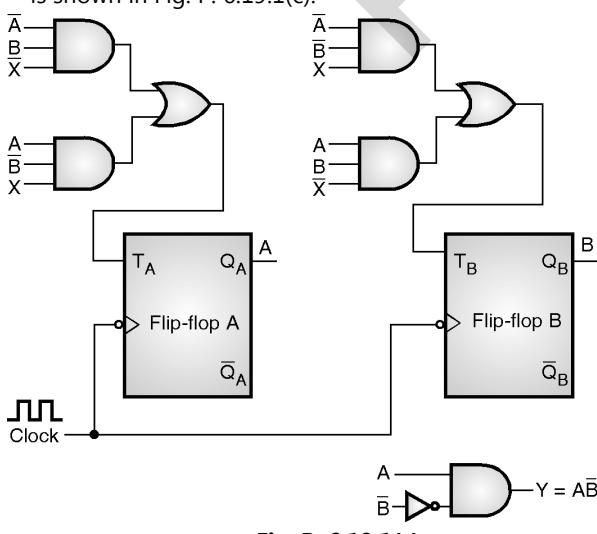
- Fig. P. 6.19.1(b) shows the K-maps and corresponding simplified expressions for T_A T_B i.e. inputs of the two T flip-flops and the Y output.



(C-686) Fig. P. 6.19.1(b) : K-maps and simplifications

Step 5 : Draw the logic diagram :

- The logic diagram for required clocked sequential circuit is shown in Fig. P. 6.19.1(c).

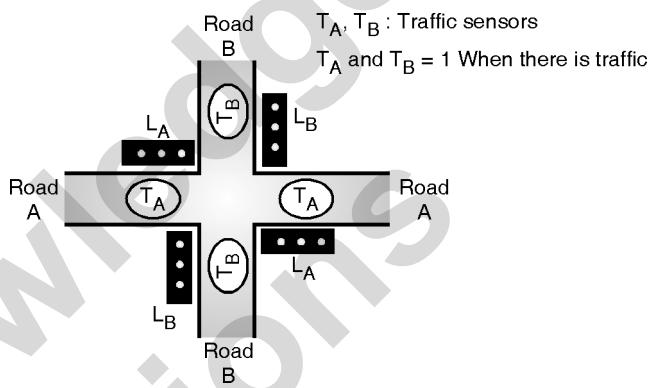


(C-687) Fig. P. 6.19.1(c)

6.20 Case Study : Use of Sequential Logic Design in a Simple Traffic Light Controller :

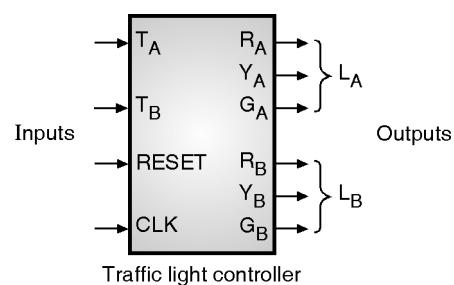
Traffic Light Controller :

- A traffic light controller is to be designed which controls the traffic lights L_A and L_B shown in Fig. 6.20.1(a) located at the intersection of two roads, on the basis of the status of the traffic sensors T_A and T_B.



(C-6097) Fig. 6.20.1(a) : Traffic lights to be designed

- Thus the inputs to the traffic light controller are the outputs of the traffic sensors, CLOCK and RESET as shown in Fig. 6.20.1(b).
- The outputs of the traffic light controller are the driving signals for the lights L_A and L_B as shown in Fig. 6.20.1(b).



(C-6098) Fig. 6.20.1(b) : Traffic light controller block diagram

Operation and state diagram :

Initially (State A) :

- Initially after resetting the controller, the status of outputs is as follows :

Initially : L _A = Green i.e. G _A = 1	State
L _B = Red i.e. R _B = 1	

(C-6099)



- If $T_A = 1$ which indicates traffic on road A then the state remains the same i.e. state A as shown in Fig. 6.20.1(c).
- But if $T_A = 0$ (No traffic on road A), irrespective of the value of T_B , the controller outputs will get modified as follows and the controller moves to state B.

$T_A = 0$	$L_A = \text{Yellow}$ i.e. $Y_A = 1$	$\} \text{State}$
$T_B = X$	$L_B = \text{Red}$ i.e. $R_B = 1$	

(C-6100)

- This is as shown in Fig. 6.20.1(c).

State B :

- After some time in state B, the controller will move to the next state i.e. state C which is defined as follows :

$L_A = \text{Red}$ i.e. $R_A = 1$	$\} \text{State}$
$L_B = \text{Green}$ i.e. $G_B = 1$	

(C-6101)

State C :

- The controller will continue to be in state C as long as $T_B = 1$ which indicates that there is traffic on road B.
- But if $T_B = 0$ (No traffic on road B), irrespective of the value of T_A , the controller outputs will change in the following manner and it will move to the next state D as shown in Fig. 6.20.1(c).

$T_A = X$	$L_A = \text{Red}$ i.e. $R_A = 1$	$\} \text{State}$
$T_B = 0$	$L_B = \text{Yellow}$ i.e. $Y_B = 1$	

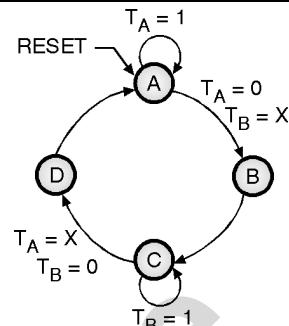
(C-6102)

State D :

- After some time in state D, the controller will automatically move to the initial state A as shown in Fig. 6.20.1(c).

(C-8310)

State	Outputs
A	$L_A = \text{Green}$, $L_B = \text{Red}$
B	$L_A = \text{Yellow}$, $L_B = \text{Red}$
C	$L_A = \text{Red}$, $L_B = \text{Green}$
D	$L_A = \text{Red}$, $L_B = \text{Yellow}$



(C-6103) Fig. 6.20.1(c) : State diagram of a traffic light controller

State table :

- We can derive the state table from the state diagram.
- The four states requires 2 flipflops (D-type). The state table of this controller is as shown in Table 6.20.1(a).

(C-8311) Table 6.20.1(a) : State table

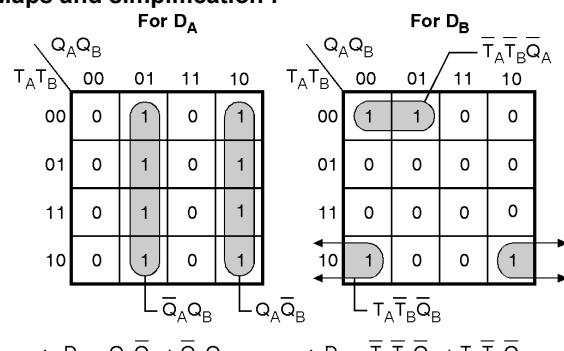
Present state		Inputs		Next state		Outputs	
Q_A	Q_B	A_A	T_B	Q_A	Q_B	L_A	L_B
A (00)	0	X		B (01)		Yellow	Red
A (00)	1	X		A (00)		Green	Red
B (01)	X	X		C (10)		Red	Green
C (10)	X	1		C (10)		Red	Green
C (10)	X	0		D (11)		Red	Yellow
D (11)	X	X		A (00)		Green	Red

Circuit excitation table :

(C-8312) Table 6.20.1(b) : Circuit excitation table

Present state		Inputs		Next state		Flip flop input		Outputs					
Q_A	Q_B	T_A	T_B	Q_A	Q_B	D_A	D_B	R_A	Y_A	G_A	R_B	Y_B	G_B
0 0(A)	0	X	0	1(B)	0	0	1	0	1	0	1	0	0
0 0(A)	1	X	0	0(A)	0	0	0	0	0	0	1	1	0
0 1(B)	X	X	1	0(C)	1	0	1	0	0	0	0	0	1
1 0(C)	X	1	1	0(C)	1	0	1	0	1	0	0	0	1
1 0(C)	X	0	1	1(D)	1	1	1	1	0	0	0	1	0
1 1(D)	X	X	0	0(A)	0	0	0	0	0	1	1	0	0

K-Maps and simplification :

(a) K-map for D_A (b) K-map for D_B

(C-6104) Fig. 6.20.2 : K-maps (Contd..)



For R_A output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	0	1	0	1	
01	0	1	0	1	
11	0	1	0	1	
10	0	1	0	1	

$$R_A = Q_A \bar{Q}_B + \bar{Q}_A Q_B$$

$$\therefore R_A = Q_A \oplus Q_B = D_A \quad \dots(3)$$

(c) K-map for R_A

For Y_A output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	1	0	0	0	0
01	1	0	0	0	0
11	0	0	0	0	0
10	0	0	0	0	0

$$Y_A = \bar{T}_A \bar{Q}_A \bar{Q}_B \quad \dots(4)$$

(d) K-map for Y_A

For G_A output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	0	0	1	0	0
01	0	0	1	0	0
11	1	0	1	0	0
10	1	0	1	0	0

$$G_A = Q_A Q_B + T_A \bar{Q}_A \bar{Q}_B \quad \dots(5)$$

(e) K-map for G_A

For R_B output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	1	0	1	0	0
01	1	0	1	0	0
11	1	0	1	0	0
10	1	0	1	0	0

$$R_B = Q_A Q_B + \bar{Q}_A \bar{Q}_B$$

$$R_B = Q_A \oplus Q_B \quad \dots(6)$$

(f) K-map for R_B

For Y_B output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	0	0	0	0	1
01	0	0	0	0	0
11	0	0	0	0	0
10	0	0	0	0	1

$$Y_B = \bar{T}_B Q_A \bar{Q}_B \quad \dots(7)$$

(g) K-map for Y_B

For G_B output					
$T_A T_B$	$Q_A Q_B$	00	01	11	10
00	0	1	0	0	0
01	0	1	0	0	1
11	0	1	0	0	1
10	0	1	0	0	0

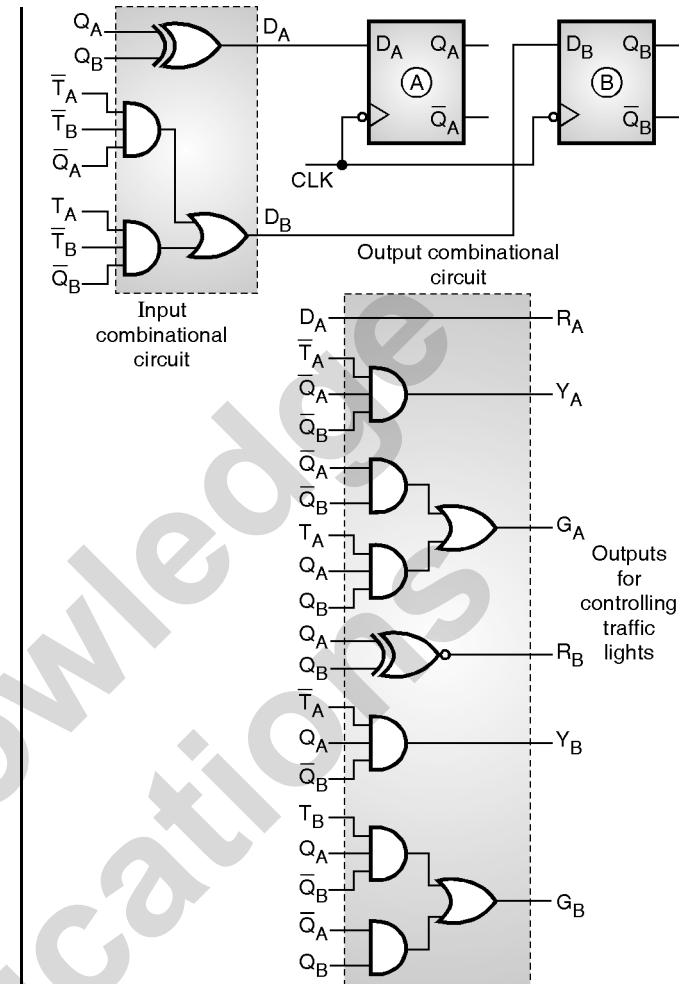
$$G_B = T_B Q_A \bar{Q}_B + \bar{Q}_A Q_B \quad \dots(8)$$

(h) K-map for G_B

(C-6104) Fig. 6.20.2 : K-maps

Realization :

- The traffic light controller is as shown in Fig. 6.20.3.



(C-6105) Fig. 6.20.3 : Realization of the traffic controller

Review Questions

- Mention the types of digital systems. Explain with block diagram their operating principle.
- What is a flip-flop ?
- State and explain the triggering methods used for flip-flops.
- What is the function of preset and clear inputs in flip-flop ?
- Explain with truth table the working of clocked RS flip-flop.
- State the disadvantages of RS flip-flop. How can they be avoided ?
- Explain with diagram the working of D type flip-flop. Give its truth table.



- | | |
|--|---|
| <p>Q. 8 Give reason why D flip-flop is called as data latch ?</p> <p>Q. 9 Draw the circuit using logic gates of a T-type flip-flop. Draw its symbol and write its truth table.</p> <p>Q. 10 Explain S-R flip flop using NOR gates.</p> <p>Q. 11 Describe how two cross-coupled NAND gates form a R-S flip-flop ? Write its truth table.</p> <p>Q. 12 What are the various types of flip-flops ?</p> <p>Q. 13 Draw the circuit of SR flip-flop using NAND gate.</p> <p>Q. 14 Draw the schematic diagram of JK flip-flop and describe its working. Write down its truth table.</p> <p>Q. 15 What is race around condition ?</p> <p>Q. 16 Draw the circuit of J-K flip-flop using NAND gate.</p> <p>Q. 17 Draw a neat diagram of master slave J-K flip-flop. Explain how race around condition is avoided using master slave J-K flip-flop ?</p> <p>Q. 18 Explain the working of the master slave JK flip-flop.</p> <p>Q. 19 Can a flip flop be used as a memory ? If so how many bits can be stored by R-S flip flop ?</p> <p>Q. 20 Explain T flip-flop.</p> <p>Q. 21 Explain the following flip-flops :
1. Clocked SR.
2. JK with preset and clear.
3. Master Slave JK.</p> | <p>4. D type and T type.</p> <p>Q. 22 Design a conversion logic to convert a JK flip-flop to a D flip-flop.</p> <p>Q. 23 Write a short note on race around condition in JK flip-flop.</p> <p>Q. 24 Draw neat circuit diagram of clocked JK flip-flop using NAND gates. Give its truth table and explain race around condition.</p> <p>Q. 25 What is race around condition ? How does it gets eliminated in master slave JK FF ? Explain.</p> <p>Q. 26 Explain how JK FF is converted into :
1. D FF 2. T FF</p> <p>Q. 27 Carry out the following flip flop conversions (Conversion tables and K-maps expected) :
1. S-R to D
2. D to S-R
3. J-K to S-R</p> <p>Q. 28 If \bar{Q} output of a D-type Flip-Flop is connected to D input, it acts as a toggle switch. State whether true or false ? Justify your answer.</p> <p>Q. 29 What is the basic difference between pulse-triggered and edge-triggered flip-flops ?</p> |
|--|---|

□□□

Unit 3

Chapter

7 Counters

Syllabus

Application of flip-flops : Counters - Asynchronous, Synchronous and modulo n counters, Study of 7490 modulus n counter ICs & their applications to implement mod counters.

Chapter Contents

7.1 Introduction	7.8 Synchronous Counters
7.2 Asynchronous / Ripple Up Counters	7.9 Modulo – N Synchronous Counters
7.3 Asynchronous Down Counters	7.10 UP / DOWN Synchronous Counter
7.4 UP / DOWN Counters	7.11 Lock Out Condition
7.5 Modulus of the Counter (MOD-N Counter)	7.12 Bush Diagram
7.6 Ripple Counter IC 7490 (Decade Counter)	7.13 Applications of Counters
7.7 Problems Faced by Ripple Counters	



7.1 Introduction :

Definition :

- The digital circuit used for counting pulses is known as counter. It is a sequential circuit.
- Counter is the most widely used application of flip-flops. It is a group of flip-flops with a clock signal applied.
- Counters count the number of clock pulses. Therefore with some modifications we can use them for measuring frequency or time period.

7.1.1 Types of Counters :

SPPU : May 08

University Questions

Q. 1 What do you mean by binary ripple counter ?

(May 08, 2 Marks)

- Counters are basically of two types :
 1. Asynchronous or ripple counters.
 2. Synchronous counters.

1. Asynchronous or ripple counters :

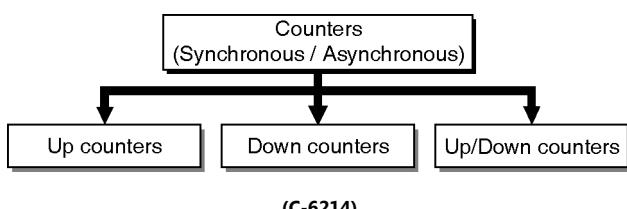
- For these counters the external clock signal is applied to one flip flop and then the output of preceding flip-flop is connected to the clock of next flip-flop.

2. Synchronous counters :

- In synchronous counters all the flip-flops receive the external clock pulse is applied to all the flip-flops simultaneously.
- Ring counter and Johnson counter are the examples of synchronous counters.

7.1.2 Classification of Counters :

- Depending on the way in which the counter outputs change, the synchronous or asynchronous counters are classified as follows :



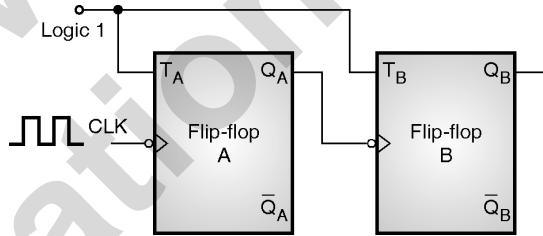
- **Up counters** are the counters that count from small to big count. Their output goes on increasing as they receive clock pulses.

- For example, the output of an up counter will be 0-1-2-3...
- **Down counters** are the counters that count from large to small count. Their output goes on decreasing as they receive clock pulses.
- For example, the output of a down counter will be 7-6-5-4-3...
- **Up/Down counter** is the combination of up counter and down counter.

7.2 Asynchronous / Ripple Up Counters :

Logic diagram :

- Fig. 7.2.1 shows the logic diagram of a 2-bit ripple up counter.



(C-771) Fig. 7.2.1 : A two bit asynchronous binary up counter

- The number of flip-flops used is 2. Note that the number of bits will always be equal to the number of flip-flops. Thus a 4 bit counter will use four flip-flops.
- The toggle (T) flip-flops are being used. But we can use the JK flip-flops also with J and K connected permanently to logic 1.
- External clock is applied to the clock input of flip-flop A which is the LSB flip-flop and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

Operation of the counter :

- Initially let both the flip-flops be in reset condition.
 $\therefore Q_B \ Q_A = 00$

On the first negative going clock edge :

- As soon as the first falling edge of the clock hits FF-A, it will toggle as $T_A = 1$. Hence Q_A will become equal to 1.
- Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B.



- There is no change in the status of Q_B because FF-B is a negative edge triggered FF.
- Therefore after the first clock pulse the counter outputs are

$$Q_B Q_A = 01 \quad \dots \text{After the first CLK pulse}$$

At the second falling edge of clock :

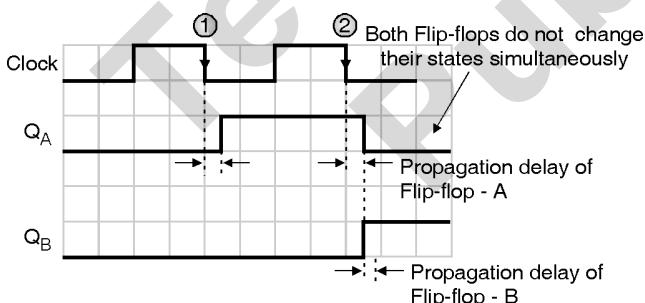
- On the arrival of second falling clock edge, FF-A toggles again and Q_A changes from 1 to 0.
- $\therefore Q_A = 0$ Corresponding to 2nd negative clock edge.
- This change in Q_A (from 1 to 0) acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will change from 0 to 1.

$$\therefore Q_B = 1$$

- Hence after the second clock pulse the counter outputs are

$$Q_B Q_A = 10 \quad \dots \text{After the second CLK pulse}$$

- Note that both the outputs are changing their state.
- But both the changes do not take place simultaneously. Q_A will change first from 1 to 0 and then Q_B will change from 0 to 1.
- This is due to the propagation delay of FF-A. So both flip-flops will never trigger at the same instant.
- Therefore the counter is called as an asynchronous counter. This is shown in Fig. 7.2.2.



(C-772) Fig. 7.2.2 : FFs do not change their state simultaneously

At the third falling edge of clock :

- On arrival of the third falling edge, FF-A toggles again and Q_A becomes 1 from 0.
- Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1.

$$\therefore Q_B Q_A = 11 \quad \dots \text{After the third CLK pulse}$$

At the 4th negative clock edge :

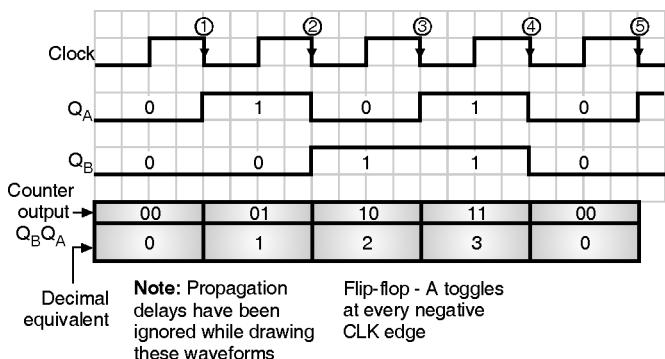
- On the 4th falling clock edge, FF-A toggles and Q_A changes from 1 to 0.
- This negative going change in Q_A acts as a negative clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0.

$$\therefore Q_B Q_A = 00 \quad \dots \text{After the fourth CLK pulse}$$

- So the counter has reached the original state. The counter operation will now repeat.
- Table 7.2.1 summarizes the operation of the counter and Fig. 7.2.3 shows the timing waveforms.

(C-6843) Table 7.2.1 : Summary of operation of a 2-bit binary ripple up counter

Clock	Counter outputs		State number	Decimal equivalent of counter output
	Q_B (MSB)	Q_A (LSB)		
Initially	0	0	-	0
1 st (↓)	0	1	1	1
2 nd (↓)	1	0	2	2
3 rd (↓)	1	1	3	3
4 th (↓)	0	0	4	0



(C-773) Fig. 7.2.3 : Timing diagram for a 2 bit ripple up counter

Why is it called counter ?

- See Fig. 7.2.3. The decimal count corresponds to the number of clock pulses, which counter has received.
- Thus this circuit counts the clock pulses. Hence it is called as counter.

Number of states :

- As seen from Table 7.2.1, this counter has four distinct states of output namely 00, 01, 10 and 11. In general the number of states = 2^n where n is equal to the number of flip-flops.

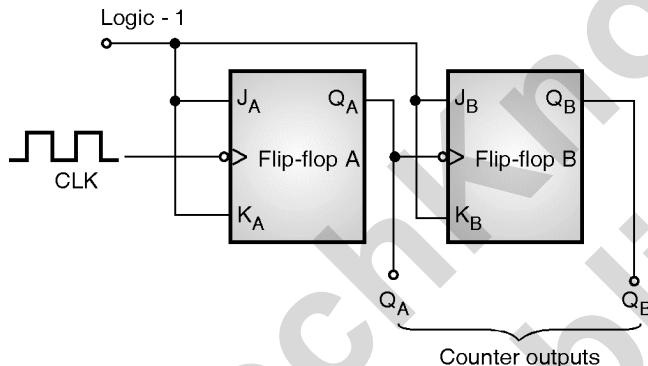
Maximum count :

- As seen from Table 7.2.1, the maximum count is 3 (decimal) i.e. 11 binary.
Maximum count = $3 = 2^2 - 1$
- In general the maximum count = $(2^n - 1)$, where n = Number of flip-flops.

7.2.1 Two Bit Asynchronous Up Counter using JK Flip-Flops :

Logic diagram :

- A 2 bit asynchronous counter up using JK flip-flops is shown in Fig. 7.2.4.



(C-774) Fig. 7.2.4 : Two bit ripple up counter using JK flip-flops

- Note that the J and K inputs of both the flip-flops are connected to logic 1 so actually JK flip-flops are converted into T flip-flops.
- The operation of this circuit is exactly same as that of the counter using the T flip-flops.

7.2.2 3 Bit Asynchronous Up Counter :

SPPU : May 07, Dec. 09, Dec. 12

University Questions

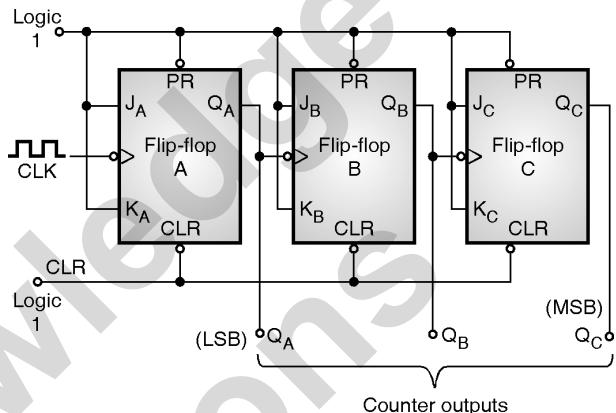
- Q. 1** Draw and explain 3-bit asynchronous up-counter. Also draw the necessary timing diagram.

(May 07, Dec. 12, 6 Marks)

- Q. 2** Draw 3-bit asynchronous counter. Explain timing diagram for the same. **(Dec. 09, 8 Marks)**

Logic diagram :

- We can apply all the basic concepts which were introduced for the 2-bit ripple up counter to the 3-bit ripple up counter.
- Fig. 7.2.5 shows the logic diagram of a 3-bit ripple up counter. Since it is a 3-bit counter, we need to use 3-flip-flops.



(C-775) Fig. 7.2.5 : 3-bit ripple up counter

- Operation of the 3-bit ripple up counter takes place in exactly similar manner as that of a 2-bit counter.

Truth table :

- Table 7.2.2 summarizes the operation of the 3-bit asynchronous up counter.

(C-776(a)) Table 7.2.2 : Summary of operation of a 3-bit ripple up counter

Clock	Flip-flop outputs			State	Decimal equivalent
	Q_C (MSB)	Q_B	Q_A (LSB)		
Initially	0	0	0	1	0
1 st (↓)	0	0	1	2	1
2 nd (↓)	0	1	0	3	2
3 rd (↓)	0	1	1	4	3
4 th (↓)	1	0	0	5	4
5 th (↓)	1	0	1	6	5
6 th (↓)	1	1	0	7	6
7 th (↓)	1	1	1	8	7
8 th (↓)	0	0	0	1	0

- Note that the asynchronous preset and clear terminals are also being used.
- Both of them are active low inputs. Hence for the normal output of the counter preset and clear terminals should be connected to logic 1.

Number of states :

- Number of states = $2^n = 2^3 = 8$.



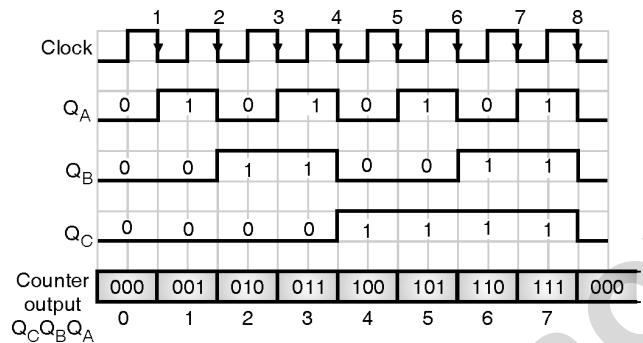
- The 3 bit ripple up counter can have 8 distinct states i.e. $Q_C Q_B Q_A$ can take up values from 000, 001, 010,110, 111.

Maximum count :

- Maximum count = $2^n - 1 = 8 - 1 = 7$. Refer Table 7.2.2. The maximum count is $Q_C Q_B Q_A = 1\ 1\ 1$ i.e. decimal 7. Note that Q_C is treated as MSB and Q_A as LSB.

Timing diagram :

- The timing diagram of a 3-bit ripple up counter are as shown in Fig. 7.2.6.



(C-776) Fig. 7.2.6 : Timing diagram for a 3-bit ripple up counter

7.2.3 4 Bit Asynchronous up Counter :

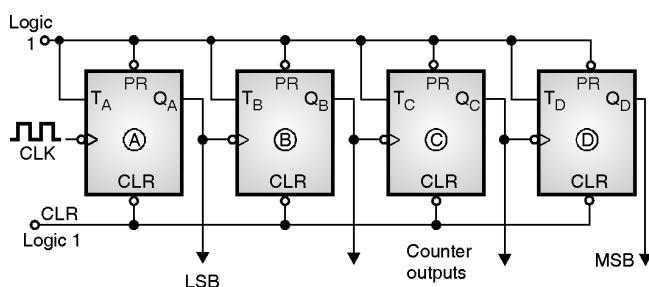
SPPU : May 08, May 11

University Questions

- Q. 1** Draw and explain 4-bit binary up counting with this concept. Also draw the necessary timing diagram. Is there any frequency division concept in it ? Comment on frequency generated at the output of each flip-flop. **(May 08, 4 Marks)**
- Q. 2** Draw 4-bit asynchronous counter. Also explain timing diagram for the same. **(May 11, 8 Marks)**

Logic diagram :

- Fig. 7.2.7(a) shows the circuit diagram of a 4 bit asynchronous counter using the T flip flops.



(C-777) Fig. 7.2.7(a) : 4 bit asynchronous up counter

- Since it is a 4 bit ripple up counter, we need to use four flip flops.
- Initially all the flipflops have zero output.

$$\therefore Q_D Q_C Q_B Q_A = 0000.$$

- All the flip flop are negative edge triggered. CLK signal is applied to the clock input of FF-A whereas Q outputs of every F/F is applied to the clock input of next F/F.
- For example Q_A to CLK of FF-B, Q_B to CLK of FF-C and so on.

Truth table :

- Table 7.2.3 shows the truth table for 4 bit asynchronous up counter. Its output passes through 16 states from 0000 i.e. $(0)_10$ to $(1111)_10$ i.e. $(15)_10$.

(C-6844) Table 7.2.3 : Truth table for a 4-bit asynchronous up counter

Clock	FF outputs				Decimal
	Q_D	Q_C	Q_B	Q_A	
Initially	0	0	0	0	0
1 st (\downarrow)	0	0	0	1	1
2 nd (\downarrow)	0	0	1	0	2
3 rd (\downarrow)	0	0	1	1	3
:	:	:			:
14 (\downarrow)	1	1	1	0	14
15 (\downarrow)	1	1	1	1	15
16 (\downarrow)	0	0	0	0	0

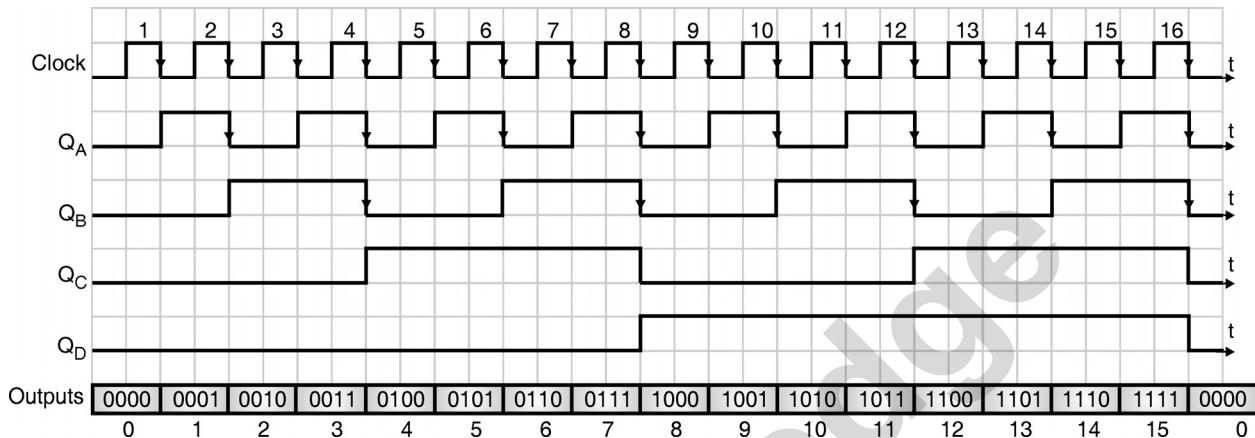
- After 1111, the output again become 0000 and the operation repeats itself.
- Fig. 7.2.7(b) shows the timing diagram for the 4-bit asynchronous up counter.
- Q_D acts as MSB of the output whereas Q_A acts as the LSB.

Number of states :

- The number of state through which the output of a 4 bit up counter passes is 16 (from 0 to 15).

Maximum count :

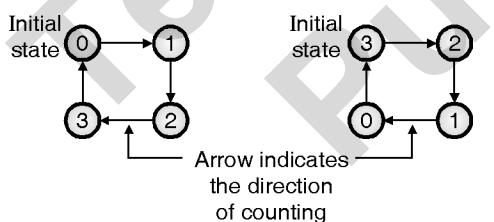
- The maximum count is 15 i.e. 1111. Thus a 4-bit ripple up counter will count from 0000 to 1111.

**Timing diagram :**

(C-778) Fig. 7.2.7(b) : Waveforms of a 4 bit asynchronous up counter

7.2.4 State Diagram of a Counter :

- The state diagram of a counter represents the states of a counter graphically.
- For example for a 2-bit up counter the state diagram is shown in Fig. 7.2.8(a) and for a 2-bit down counter the state diagram is shown in Fig. 7.2.8(b).
- The number written inside a circle represents the state number, whereas the arrow shows the direction of counter (up or down).
- Note that in the counters only the state is important. Hence in the state diagram we have not shown any input or output conditions.



(C-779)Fig. 7.2.8 : State diagram

7.3 Asynchronous Down Counters :**7.3.1 3- Bit Asynchronous Down Counter :****Truth table and state diagram :**

- All the counters discussed so far have counted upwards from zero. So they can be called as **up counters**.

- But the counters which can count in the **downward** direction i.e. from the maximum count to zero are called **down counters**.

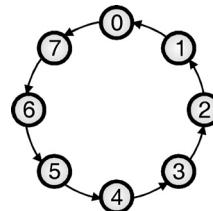
- The countdown sequence for a 3-bit asynchronous down counter is as follows :

(C-6970) Table 7.3.1 : Truth table of 3 bit down counter

CLK	Output			Decimal Count
	Q _C	Q _B	Q _A	
Maximum count	0	0	0	0
↓	1	1	1	7
↓	1	1	0	6
↓	1	0	1	5
↓	1	0	0	4
↓	0	1	1	3
↓	0	1	0	2
↓	0	0	1	1
↓	0	0	0	0

Direction of counting ↓

Repeats



(C-2762) Fig. 7.3.1 : State diagram

- Thus counting takes place as follows :

$$Q_C \ Q_B \ Q_A = 111, 110, 101, 100, 011, 010, 001, 000.$$

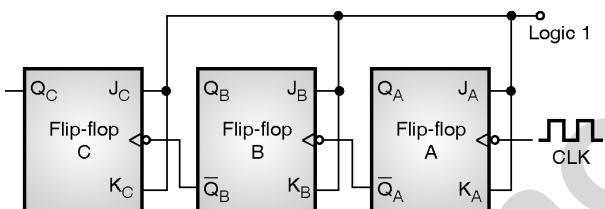
- From this sequence it is evident that FF-A should toggle at every negative going clock edge but FF-B should change its state only at those instants when Q_A changes from LOW (0) to HIGH (1) and Q_C should change only when Q_B changes from LOW to HIGH.



- Thus in a down counter, each FF except the first one (FF-A) should toggle when the output of its preceding flip-flop changes from LOW to HIGH.
- If all the FFs are negative edge triggered i.e. responding to the negative CLK edge, then we can place an inverter in front of every CLK input or we can drive the CLK input of next FF from the \bar{Q} output of the preceding FF and not from the Q outputs as shown in Fig. 7.3.2.

Logic diagram :

- A 3-bit asynchronous down counter is shown in Fig. 7.3.2. The clock input is applied directly to the clock input of FF-A. But \bar{Q}_A is connected to clock of FF-B, \bar{Q}_B to clock of FF-C and so on.



(C-781) Fig. 7.3.2 : A 3-bit asynchronous down counter

Operation :

- Initially let all the flip-flops be in the reset condition.
 $\therefore Q_C Q_B Q_A = 0 0 0$
- As soon as the first falling clock pulse arrives, FF-A toggles. So Q_A becomes 1 and \bar{Q}_A changes from 1 to 0.
- The negative going change in \bar{Q}_A acts as a clock to FF-B. Hence FF-B will change its state. So Q_B becomes 1 and \bar{Q}_B changes from 1 to 0.
- This negative going change in \bar{Q}_B acts as a clock to FF-C. Hence FF-C will change its state. So Q_C becomes 1 and \bar{Q}_C becomes 0.
- Thus after the first clock pulse the output of counter are,
 $Q_C Q_B Q_A = 1 1 1 \dots \text{After the } 1^{\text{st}} \text{ CLK pulse}$
- Corresponding to the second falling clock edge, FF-A toggles. Q_A becomes 0 and \bar{Q}_A becomes 1. This positive going change in \bar{Q}_A does not alter the state of FF-B. So Q_B remains 1 and \bar{Q}_B remains 0. So there is no change in the state of FF-C. Hence after the second clock pulse the counter outputs are,

$$Q_C Q_B Q_A = 1 1 0 \dots \text{After the } 2^{\text{nd}} \text{ CLK pulse}$$

- The down counting will thus take place. Similarly the counter will count down to pass through the states 101, 100, 011, 010, 001 and 000. The operation repeats itself thereafter.

7.4 UP / DOWN Counters :

- We have designed the up counters and the down counters separately.
- But in practice both these modes are generally combined together and an UP/DOWN counter is formed.
- A mode control (M) input is also provided to select either up count or down count mode of operation.
- A combinational circuit is required to be designed and used between each pair of flip-flops in order to achieve the up/down operation.

Types of up/down counters :

- The up/down counters are of two types :
 1. UP/DOWN ripple counters.
 2. UP/DOWN synchronous counters.

7.4.1 UP/DOWN Ripple Counters :

- In the up/down ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used.
- The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from Q or \bar{Q} output of the previous FF.

UP counting mode (M = 0) :

- The CLK signal is applied directly to the clock input of the LSB flip-flop.
- For the remaining flip-flops, the Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M = 0).

DOWN counting mode (M = 1) :

- The clock signal is applied directly to the clock input of the LSB flip-flop. For the remaining flip-flop, the \bar{Q} output of the preceding FF is connected to the clock of the next FF.



- This will operate the counter in the down counting mode. For down counting mode the mode select input M is kept at logic 1 (M = 1).

7.5 Modulus of the Counter (MOD-N Counter) :

SPPU : Dec. 09, Dec. 10

University Questions

Q. 1 What is MOD counter ? (Dec. 09, 2 Marks)

Q. 2 What is the advantage of MOD counter ?

(Dec. 10, 2 Marks)

Definition :

- Modulus (MOD) of a counter represents the number of states through which the counter progresses during its operation. It is denoted by N.
- Thus MOD-N counter means the counter progresses through N states.
- A MOD-4 counter will have 4 states. A MOD-6 counter will have 6 states etc.
- Thus a 3 bit counter which has 8 states is a MOD 8 counter, and a 4 bit counter is a MOD-16 counter.
- In general m number of flip-flops are required to construct mod-n counter, where $N \leq 2^m$.
- We can design a modulo counter with the help of the basic ripple counter structure and a combinational logic called reset logic.
- The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter where

$$\text{MOD number} = 2^n$$

- So we can conclude that **modulus** of a counter represents the **number of states** through which the counter progresses during its operation.
- Can we have a modulo-5 counter using a 3-bit ripple counter ? That means can we restrict the number of states to only 5 instead of 8 under normal conditions ?
- The answer is yes. We can design such modulo counters with the help of the basic ripple counter structure and a combinational logic called reset logic.

- Table 7.5.1 shows the relation between 2, 3 and 4 bit counters and their modulus.

(C-8049) Table 7.5.1

Counter type	Modulus
2 bit up or down	MOD-4
3 bit up or down	MOD-8
4 bit up or down	MOD-16

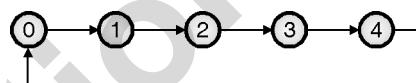
7.5.1 Design of Asynchronous MOD Counters :

Ex. 7.5.1 : Design MOD-5 asynchronous counter, and also draw the waveforms.

Soln. :

Step 1 : Draw the state diagram :

- The state diagram of MOD-5 ripple counter is as shown in Fig. P. 7.5.1(a).



(C-794) Fig. P. 7.5.1(a) : State diagram of a MOD-5 ripple counter

Step 2 : Write truth table for the reset logic :

- Table P. 7.5.1 shows the truth table for the reset logic.

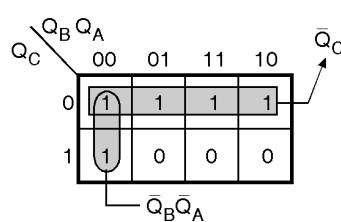
(C-6216) Table P. 7.5.1 : Truth table for the reset logic

State	Flip-flop outputs			Output Y of reset logic
	Q _C	Q _B	Q _A	
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Valid states Invalid states

Step 3 : K-map :

For output Y



Expression for Y
 $Y = \bar{Q}_C + \bar{Q}_B \bar{Q}_A$

(C-795) Fig. P. 7.5.1(b) : K-map and simplification

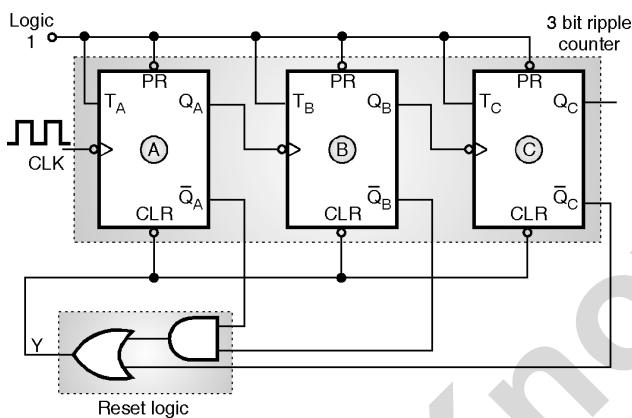
- The K map is as shown in Fig. P. 7.5.1(b).



- The states 0 through 4 are valid states and the output Y of reset logic (Y) is inactive (0) for them.
- The states 5, 6 and 7 are invalid states. If counter enters into any one of these states that Y = 1 (active) and will reset all the flip-flops.

Step 4 : Logic diagram :

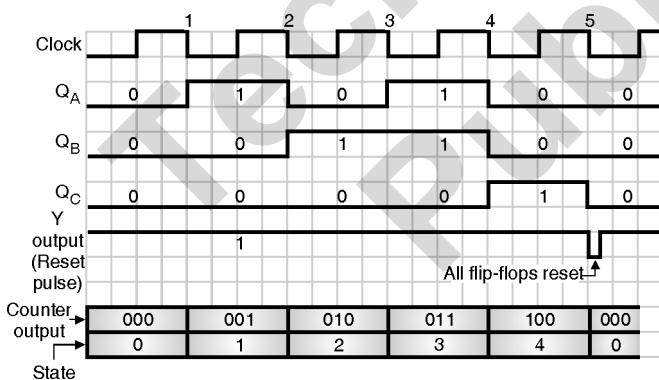
- The logic diagram of a MOD-5 ripple counter is shown in Fig. P. 7.5.1(c).



(C-796) Fig. P. 7.5.1(c) : Logic diagram of MOD-5 ripple counter

Step 5 : Timing diagram :

- The timing diagram is as shown in Fig. P. 7.5.1(d).



(C-797) Fig. P. 7.5.1(d) : Timing diagram of MOD-5 ripple counter

Ex. 7.5.2 : For MOD-11 asynchronous up counter :

- Draw circuit diagram. Use T flip-flop.
- Write truth table.
- Draw timing diagram.
- If the output frequency is 11 kHz what is the clock input ?

Soln. :

Step 1 : Write the truth table :

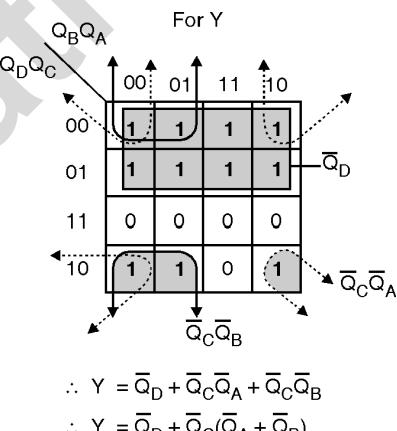
(C-6268) Table P. 7.5.2

Q _D	Q _C	Q _B	Q _A	Y output
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	0

↑
Invalid states
↓

Step 2 : Draw the K map :

- The K map is shown in Fig. P. 7.5.2(a).

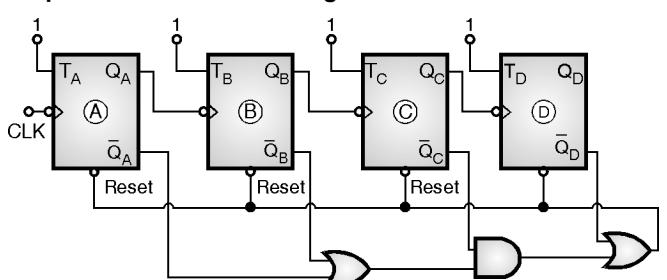


(C-801) Fig. P. 7.5.2(a) : K map

- The simplified equation for Y output of the reset logic is as follows :

$$Y = \overline{Q}_D + \overline{Q}_C (\overline{Q}_A + \overline{Q}_B)$$

Step 3 : Draw the circuit diagram :



(C-802) Fig. P. 7.5.2(b) : MOD 11 counter



7.5.2 Frequency Division Taking Place in Asynchronous Counters :

SPPU : May 05, May 08, Dec. 11

University Questions

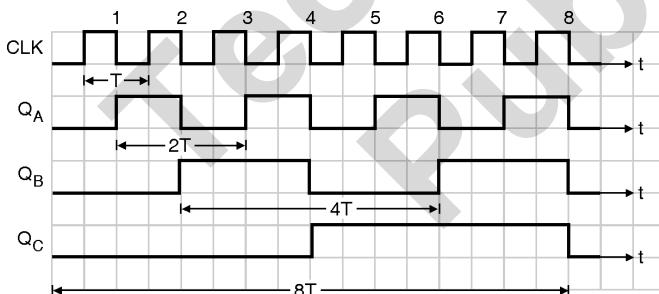
Q. 1 Assume 16 MHz clock source in a system. How will you divide this frequency by a factor 8 ? Explain your logic with suitable circuit diagram.

(May 05, Dec. 11, 8 Marks)

Q. 2 What do you mean by binary ripple counter ? Draw and explain 4-bit binary up counting with this concept. Also draw the necessary timing diagram. Is there any frequency division concept in it ? Comment on frequency generated at the output of each flip-flop.

(May 08, 10 Marks)

- In the chapter on flip-flops, we have seen that a flip-flop in toggle mode divides the clock frequency by 2.
- That means the frequency of Q or \bar{Q} output waveform of a toggle flip-flop is exactly half of the clock frequency.
- The concept of frequency division is applicable to the counters as well because we use flip-flops in the toggle mode for counters.
- Refer to the timing waveforms of a 3-bit asynchronous up counter and observe the frequencies of Q_A , Q_B and Q_C waveforms with respect to the clock frequency.



(C-807) Fig. 7.5.1 : Timing waveforms of a 3-bit asynchronous up counter

Conclusions :

- Let one cycle period of the clock signal be T sec. Hence the clock frequency $f_{CLK} = (1/T)$ Hz.
- Output of the least significant flip-flop (i.e. FF-A) has a one cycle period of (2 T) as shown in Fig. 7.5.1. Hence,

$$\text{Frequency of } Q_A \text{ output} = f_A = \frac{1}{2T} = \frac{f_{CLK}}{2}$$

- The one cycle period of Q_B is 4 T. Hence the frequency of Q_B is given by,

$$f_B = \frac{1}{4T} = \frac{f_{CLK}}{4} \quad \dots \text{since } \frac{1}{T} = f_{CLK}$$

- Similarly the frequency of Q_C output is given by

$$f_C = \frac{1}{8T} = \frac{f_{CLK}}{8}$$

Note : In any counter, the signal at the output of last FF (i.e. MSB) will have a frequency equal to the input clock frequency divided by the MOD number of the counter.

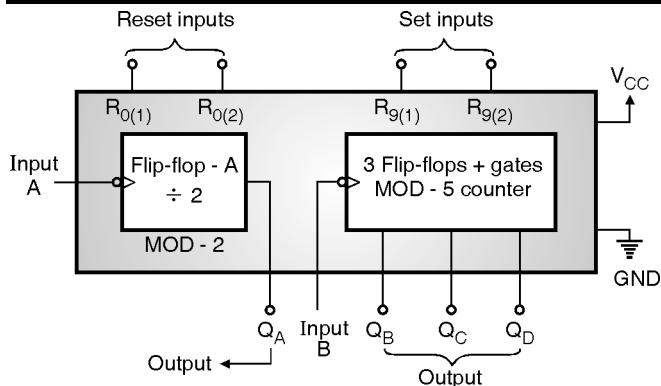
7.5.3 Disadvantages of Ripple Counters :

- Every flip-flop has its own propagation delay. In ripple counter the output of the previous FF is used as clock for the next FF.
- Hence the propagation delay goes on accumulating. For a 3-bit ripple counter the propagation delay of the first FF gets added to that of the second FF to decide the transition time for the third stage.
- This accumulated time delay is the main problem with the ripple counters, because the propagation delay goes on increasing with increase in number of flip-flops.
- This will put a limitation on the maximum clock frequency.

7.6 Ripple Counter IC 7490 (Decade Counter) :

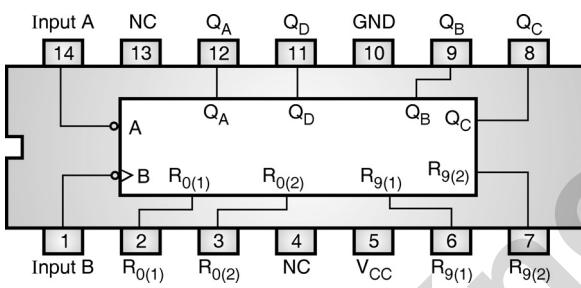
Internal structure :

- IC 7490 is a TTL MSI decade counter. It contains four master slave flip-flops and a few logic gates to provide a divide-by-two counter and a three stage binary counter which provides a divide by 5 counter. (MOD-5), as shown in Fig. 7.6.1.



(C-1374) Fig. 7.6.1 : The basic internal structure of IC 7490

- IC 7490 is a MOD-10 or decade counter. It is a 14 pin IC and its pin configuration as shown in Fig. 7.6.2.



(C-1375) Fig. 7.6.2 : Pin configuration of IC 7490

Description :

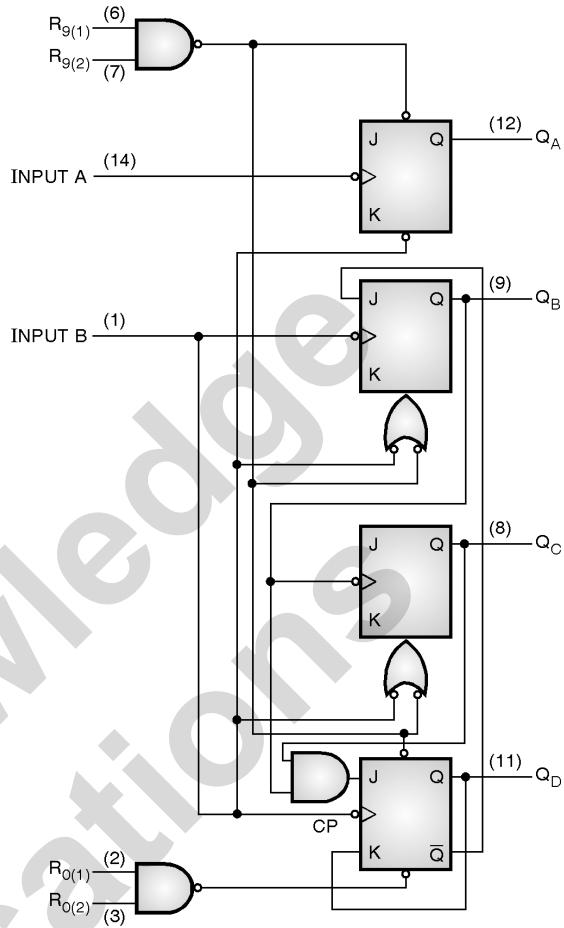
- Table 7.6.1 explains the pin configuration of IC 7490.

Table 7.6.1 : Pin name and description of IC 7490

Pin name	Description
Input B	This is clock input to the internal MOD-5 ripple counter, which is negative edge triggered.
R ₀₍₁₎ , R ₀₍₂₎	Gated zero reset inputs.
V _{CC}	+ 5 V DC
R ₉₍₁₎ , R ₉₍₂₎	These are gated set to nine inputs.
Q _D , Q _C , Q _B	Outputs of internal MOD-5 counter with Q _D as MSB.
GND	Logic ground. Acts as a reference.
Q _A	Output of internal mod-2 counter or FF-A.
Input A	Clock input to FF-A which is negative edge triggered.

7.6.1 The Internal Diagram of IC 7490 :

- A simplified internal diagram of IC 7490 is shown in Fig. 7.6.3.



(C-1376) Fig. 7.6.3 : A simplified internal diagram of IC 7490

Function table :

- The reset/count function table of IC 7490 is shown in Table 7.6.2.

(C-6217) Table 7.6.2 : Reset/count truth table

Reset inputs				Output			
R ₀₍₁₎	R ₀₍₂₎	R ₉₍₁₎	R ₉₍₂₎	Q _D	Q _C	Q _B	Q _A
1	1	0	X	0	0	0	0
1	1	X	0	0	0	0	0
X	X	1	1	1	0	0	1
X	0	X	0	COUNTER			
0	X	0	X	COUNTER			
0	X	X	0	COUNTER			
X	0	0	X	COUNTER			

→ Reset to 0000
 → Set to 1001
 If CLK applied it will count or step from previous to next step

Conclusion :

1. If both the reset inputs R₀₍₁₎ and R₀₍₂₎ are at logic 1 then all the flip-flops will be reset and the output is given by $Q_D\ Q_C\ Q_B\ Q_A = 0\ 0\ 0\ 0$
2. If both the preset inputs R₉₍₁₎ and R₉₍₂₎ are at logic 1 then the counter output is set to decimal 9.



$$\therefore Q_D Q_C Q_B Q_A = 1001$$

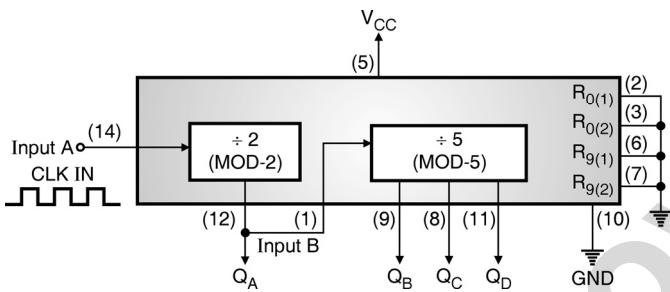
3. If any one pin of $R_{0(1)}, R_{0(2)}$ and one of $R_{9(1)}, R_{9(2)}$ is low, then the counter will be in the count mode.

Ex. 7.6.1 : Demonstrate the use of IC 7490 as a decade counter and explain its operation.

Dec. 04, 3 Marks

Soln. :

- The connection diagram for IC 7490 as decade counter is shown in Fig. P. 7.6.1.



(C-1377) Fig. P. 7.6.1 : 7490 as decade counter

- Note that the Q_A output of FF-A is externally connected to the input B which is the clock input of the internal mod-5 ripple counter.
- Hence Q_A will toggle on every falling edge of the clock input whereas the outputs $Q_D Q_C Q_B$ of the MOD-5 counter will increment from 000 to 100 on the low going change of Q_A output.
- Table P. 7.6.1 summarizes the operation of the 7490 as decade counter. Due to cascading of MOD-2 and MOD-5 counters, the overall configuration becomes a MOD - 10 i.e. decade counter configuration.
- The reset inputs $R_{0(1)}, R_{0(2)}$ and the preset inputs $R_{9(1)}, R_{9(2)}$ are connected to ground so as to make them inactive.
- As shown in Table P. 7.6.1 the counter counts from 0000 to 1001 i.e. from 0 to 9. After 1001 the MOD-5 counter resets to 000 and Q_A changes to 0. Hence the next count after 1001 is 0000 and recycling begins.

(C-6218) Table P. 7.6.1 : Summary of operation of IC 7490 decade counter

Q_D	Q_C	Q_B	Outputs of MOD-5 counter		CLK	Count
			Q_A	Output		
0	0	0	0	0	0	0
0	0	0	1	1 (↓)	1	1
0	0	1	0	2 (↓)	2	2
0	0	1	1	3 (↓)	3	3
0	1	0	0	4 (↓)	4	4
0	1	0	1	5 (↓)	5	5
0	1	1	0	6 (↓)	6	6
0	1	1	1	7 (↓)	7	7
1	0	0	0	8 (↓)	8	8
1	0	0	1	9 (↓)	9	9

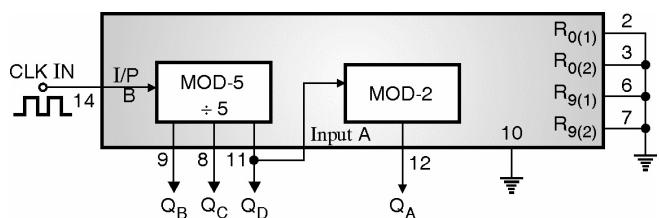
FF-A toggles on every falling clock edge. It acts as clock for MOD-5 counter.
MOD-5 counter outputs remain unchanged if Q_A changes from 0 to 1. But MOD-5 counter outputs increment by 1 every time, Q_A changes from 1 to 0

7.6.2 Other Applications of IC 7490 :

- Some of the other applications of IC 7490 are as follows:
 - Symmetrical Bi-quinary divide by ten counter.
 - Divide by two (MOD-2) and divide by 5 (MOD-5) counter.

7.6.3 Symmetrical Bi-quinary Divide by Ten Counter :

- In this application the clock input is applied to the B input and Q_D output is connected to A input as shown in Fig. 7.6.4.
- The output is obtained at Q_A output. It is a perfect square wave with a 50% duty cycle at a frequency equal to $f_{CLK} / 10$.



(C-1378) Fig. 7.6.4 : Symmetrical bi-quinary divide by ten counter

**Operation :**

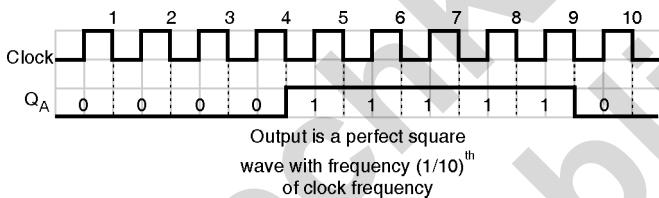
- The operation of this counter has been summarized in Table 7.6.3.

(C-6219) Table 7.6.3 : Summary of operation of symmetrical bi-quinary divide by 10 counter

Clock pulse number	Outputs				Count
	(LSB) Q _B	Q _C	Q _D	(MSB) Q _A	
0	0	0	0	0	0
1 (↓)	1	0	0	0	1
2 (↓)	0	1	0	0	2
3 (↓)	1	1	0 Transition	0	3
4 (↓)	0	0	1	0 Toggle	4
5 (↓)	0	0	0	1	8
6 (↓)	1	0	0	1	9
7 (↓)	0	1	0	1	10
8 (↓)	1	1	0	1	11
9 (↓)	0	0	1	1	12

Output waveform :

- The waveform at Q_A output is shown in Fig. 7.6.5. It shows that Q_A output is a perfect square wave.



(C-1379) Fig. 7.6.5 : Waveforms showing a perfect square wave output

Ex. 7.6.2 : Draw MOD 6 counter using IC 7490. Write truth table.

Dec. 04, 3 Marks

Soln. :

Step 1 : Truth table :

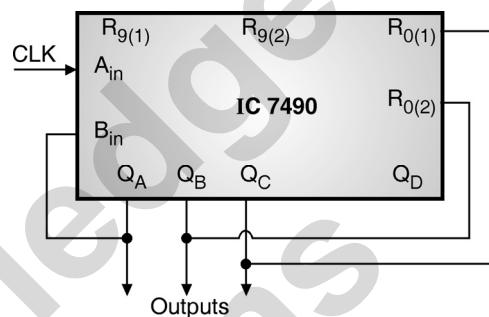
- Table P. 7.6.2 shows the truth table for a MOD 6 counter.

(C-7764) Table P. 7.6.2

Q _C	Q _B	Q _A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
0	0	0

Step 2 : Draw the circuit :

- The total number of states is 6.
- Note that Q_C and Q_B are connected to R₀₁ and R₀₂ respectively.
- So when the counter output is (6)₁₀ i.e. Q_C Q_B Q_A = 110 then the counter will reset to 000.

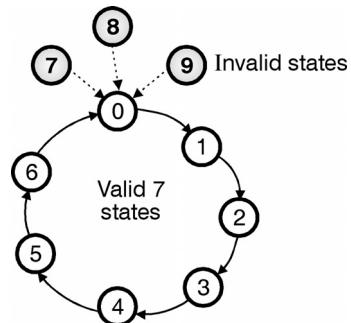


(C-1380) Fig. P. 7.6.2 : MOD 6 counter using IC 7490

Ex. 7.6.3 : Design a MOD-7 counter using 7490.

Soln. :

- MOD - 7 counter counts through the 7 states as shown in Fig. P. 7.6.3(a).



(C-1381) Fig. P. 7.6.3(a) : State diagram for MOD-7 counter

- From all the invalid states (7, 8, 9) and from 6 the counter should reset.

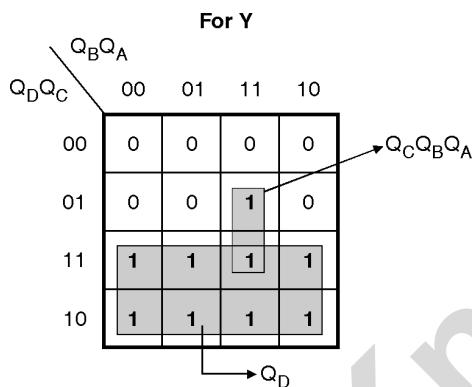
Design of reset logic :

- The output of the reset logic should be 1 corresponding to all the invalid states. The reset logic output is applied to R₀₍₁₎ and R₀₍₂₎.
- The truth table for the reset logic is shown in Table P. 7.6.3 and the K-map is shown in Fig. P. 7.6.3(b).



(C-6220) Table P. 7.6.3 : Truth table of the reset logic

Q_D	Q_C	Q_B	Q_A	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1



(C-1382) Fig. P. 7.6.3(b) : K-map for the output of reset logic

- Note that for all the states beyond 1001, we have entered a logic 1 in the K-map treating all those states as invalid states.

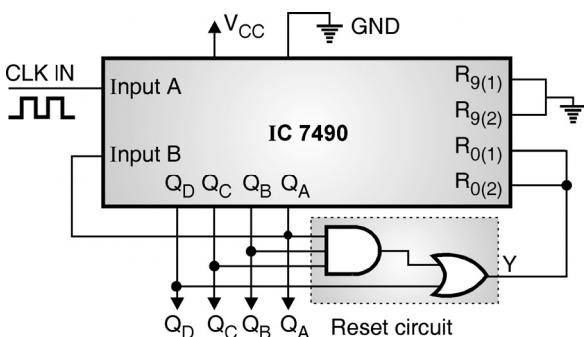
Expression for Y :

- From the K-map, the simplified expression for Y is given by,

$$Y = Q_C Q_B Q_A + Q_D$$

Logic diagram :

- The logic diagram of the MOD-7 counter is shown in Fig. P. 7.6.3(c).



(C-1383) Fig. P. 7.6.3(c) : MOD -7 counter using IC 7490

Ex. 7.6.4 : Draw basic internal architecture of IC 7490.
Design a divide by 20 counter using same.

Dec. 11, 8 Marks

Soln. :

Soln. : Solve it yourself.

Ex. 7.6.5 : Explain the internal diagram of IC 7490.
Design MOD 7 and MOD 98 counter using 7490.

May 12, 8 Marks

Soln. :

- Refer section 7.6.1 for internal diagram of IC 7490. Refer Ex. 7.6.3 for MOD-7 counter.

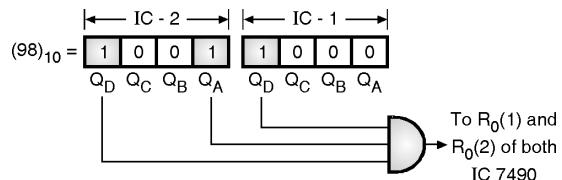
MOD 98 counter :

Step 1 : Number of ICs required :

- Up to MOD - 100, two IC 7490s will be sufficient. Hence to implement a divide by 98 counter, we have to use two decade counter ICs.

Step 2 : Design of reset logic :

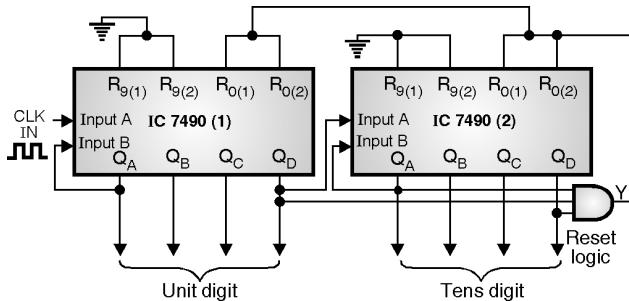
- To design the reset logic, earlier we used to draw the K-maps.
- The 3, 4 or at the most 5 variable K-maps are practically possible to handle.
- But here, there will be 8 FFs and so there will be 8-variables. So use of K-maps is practically impossible.
- Hence we will simplify the reset logic design as follows :
- A divide by 98 counter counts through 98 states from 0 to 97 and the counter should reset as soon as the count becomes 98.
- So in order to reset the counter at 98, connect the Q outputs which are equal to 1 in the count of 98 to an AND gate as shown in Fig. P. 7.6.5 and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



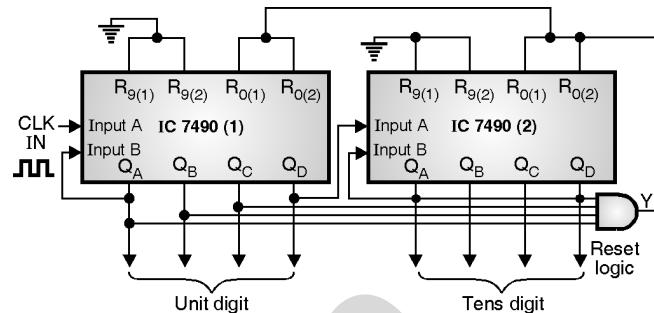
(C-6005) Fig. P. 7.6.5

Step 3 : Draw the logic diagram :

- The logic diagram of the MOD-98 counter is shown in Fig. P. 7.6.5(a).



(C-3602) Fig. P. 7.6.5(a)



(C-3604) Fig. P. 7.6.6(b)

Ex. 7.6.6 : Design the following using IC7490 :

1. MOD 97 counter
2. MOD 45 counter.

Dec. 12, 8 Marks, May 17, 6 Marks

Soln. :

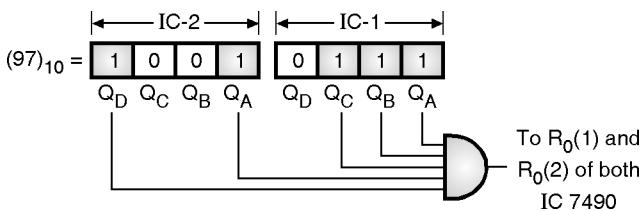
1. MOD 97 counter :

Step 1 : Number of ICs required :

- Upto MOD - 100, two IC 7490s will be sufficient.

Step 2 : Design of reset logic :

- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 97, connect the Q outputs which are equal to 1 in the count of 97 to an AND gate as shown in Fig. P. 7.6.6(a) and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-6006) Fig. P. 7.6.6(a) : Reset logic

Step 3 : Draw the logic diagram :

- The logic diagram of the MOD-97 counter is shown in Fig. P. 7.6.6(b).

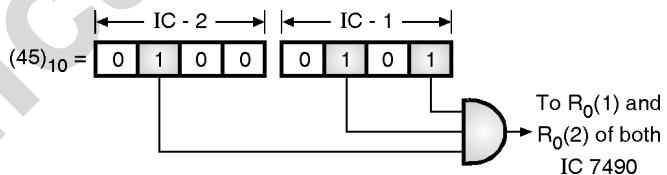
2. MOD 45 counter :

Step 1 : Number of ICs required :

- Upto MOD - 100, two IC 7490s will be sufficient

Step 2 : Design of reset logic :

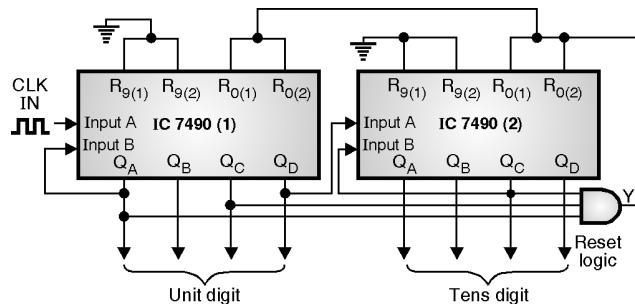
- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 45, connect the Q outputs which are equal to 1 in the count of 45 to an AND gate as shown in Fig. P. 7.6.6(c) and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-6007) Fig. P. 7.6.6(c) : Reset logic

Step 3 : Draw the logic diagram :

- The logic diagram of the MOD-45 counter is shown in Fig. P. 7.6.6(d).



(C-3606) Fig. P. 7.6.6(d)

Ex. 7.6.7 : Design and draw logic diagram of Mod-82 counter using IC7490. **Dec. 14, 6 Marks**

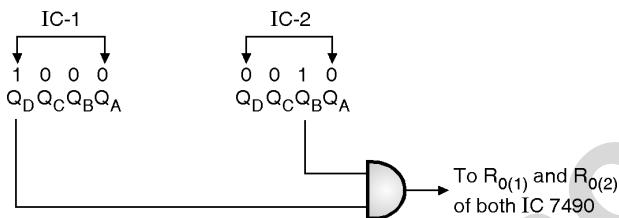
Soln. :

Step 1 : Number of ICs required :

Upto MOD - 100, two IC 7490s will be sufficient.

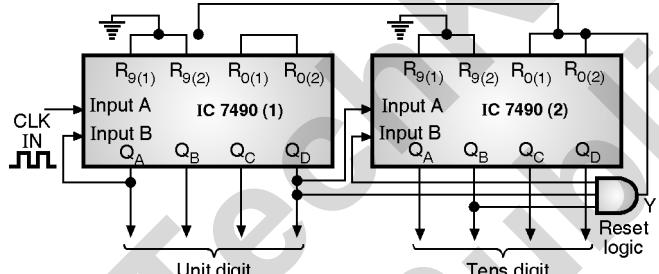
Step 2 : Design of reset logic :

- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 82, connect the Q outputs which are equal to 1 in the count of 82 to an AND gate as shown in Fig. P. 7.6.7(a) and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-4939) Fig. P. 7.6.7(a) : Reset logic

Step 3 : Draw the logic diagram :



(C-4940) Fig. P. 7.6.7(b) : MOD-82 using IC 7490

The logic diagram of the MOD-82 counter is shown in Figs. 7.6.7(b).

Ex. 7.6.8 : Design a MOD-11 counter using IC7490. Show states with the help of timing diagram.

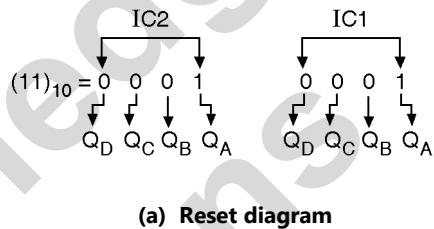
May 15, 7 Marks

Soln. :

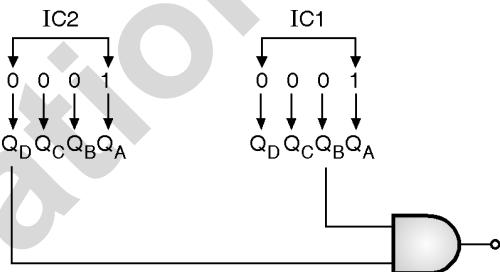
MOD-11 counter using IC7490 :

Step 1 : To design a divide by 11 (MOD-11) counter we have to use two IC 7490 counter ICs.

Step 2 : Design of reset logic. Both ICs should reset as soon as the count is equal to 11 decimal.



(a) Reset diagram

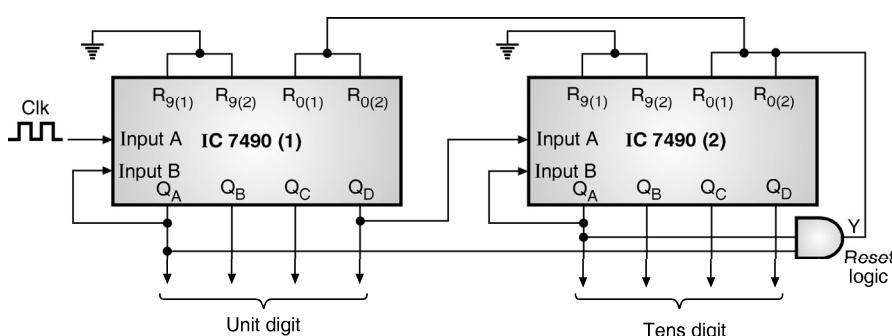


(b) To $R_{0(1)}$ and $R_{0(2)}$ of both IC7490

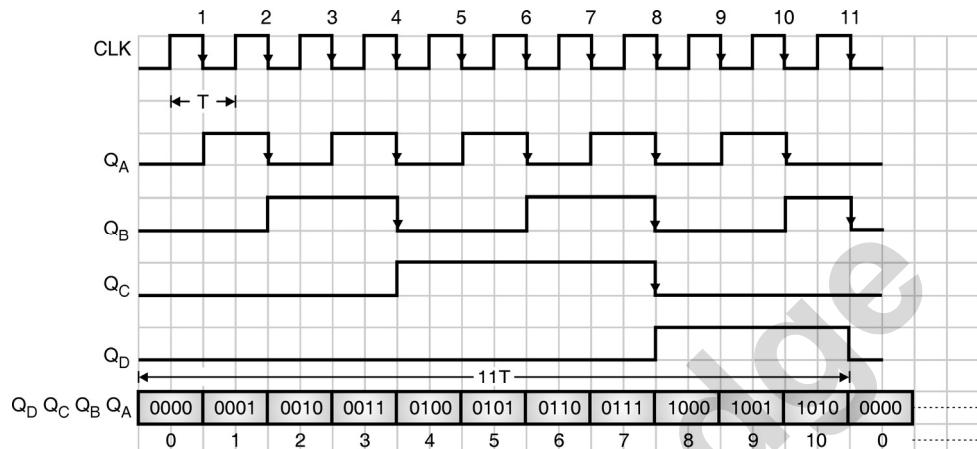
(C-5129) Fig. P. 7.6.8

So in order to reset the counter at 11, connect the Q output which are equal to '1' in the count of 11 to an AND gate as shown in Fig. P. 7.6.8(b).

Step 3 : The logic diagram of the MOD-11 is Fig. P. 7.6.8(c).



(C-5130) Fig. P. 7.6.8(c)

Timing diagram :

(C-803) Fig. P. 7.6.8(d)

Ex. 7.6.9 : Design and draw logic diagram of Mod 72 counter using IC 7490.

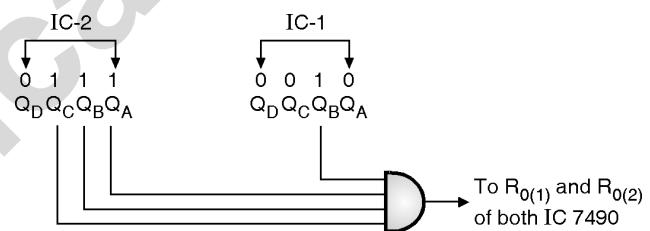
Dec. 16, 6 Marks

Soln. :**Step 1 : Number of ICs required :**

Upto MOD - 100, two IC 7490s will be sufficient.

Step 2 : Design of reset logic :

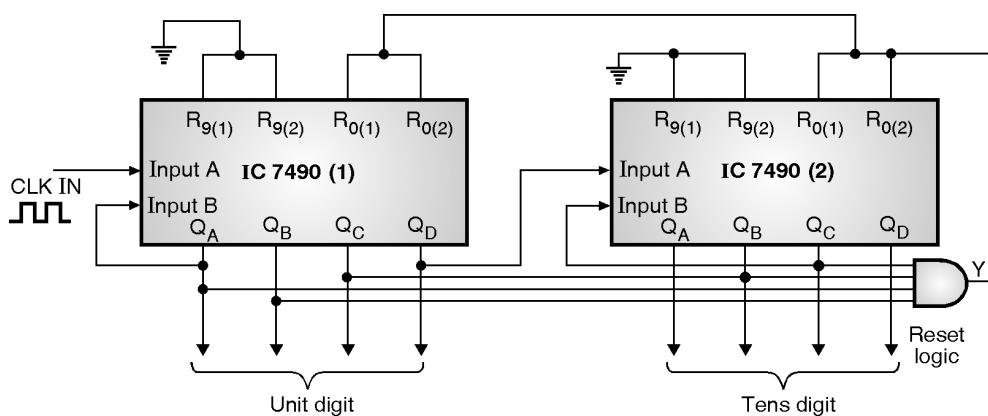
- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 72, connect the Q outputs which are equal to 1 in the count of 72 to an AND gate as shown in Fig. P. 7.6.9(a) and then connect the AND output to the R₀₍₁₎ and R₀₍₂₎ i.e. reset inputs of both the ICs.



(C-5713) Fig. P. 7.6.9(a) : Reset logic

Step 3 : Draw the logic diagram :

- The logic diagram of the MOD-72 counter is shown in Fig. P. 7.6.9(b)



(C-5709) Fig. P. 7.6.9(b)



Ex. 7.6.10 : What is Mod counter ? Explain MOD-26 counter using IC 7490. Draw design for the same.

Dec. 17, 6 Marks

Soln. :

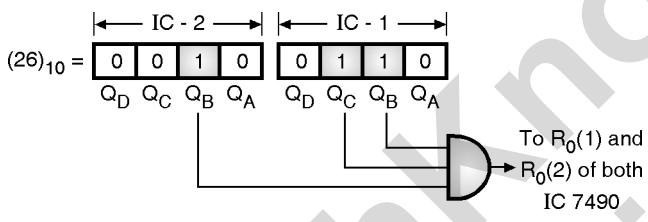
Refer section 7.5 for definition of Mod counter.

Step 1 : Number of ICs required :

Upto MOD - 100, two IC 7490s will be sufficient.

Step 2 : Design of reset logic :

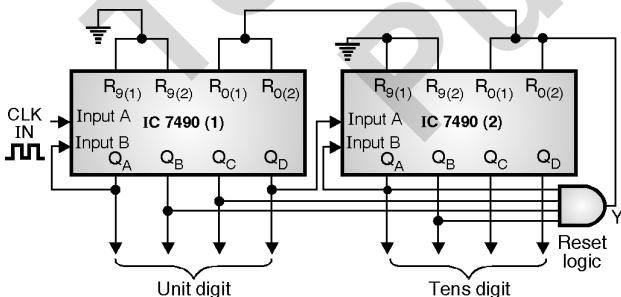
- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 26, connect the Q outputs which are equal to 1 in the count of 26 to an AND gate as shown in Fig. P. 7.6.10 and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-6331) Fig. P. 7.6.10

Step 3 : Draw the logic diagram :

- The logic diagram of the MOD-26 counter is shown in Fig. P. 7.6.10(a).



(C-6322) Fig. P. 7.6.10(a) : MOD-26 counter

Ex. 7.6.11 : Design and draw MOD 56 counter using IC 7490 and explain its operation.

May 18, 6 Marks

Soln. :

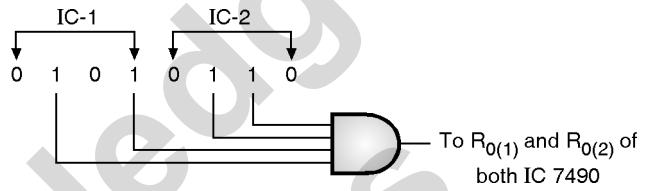
Step 1 : Number of ICs required :

Upto MOD - 100, two IC 7490s will be sufficient.

Step 2 : Design of reset logic :

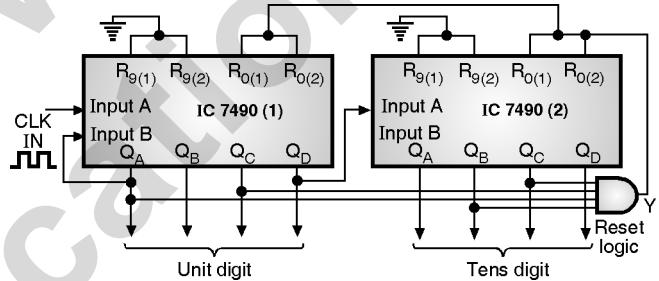
- Refer previous example for the procedure to design the reset logic.

So in order to reset the counter at 56, connect the Q outputs which are equal to 1 in the count of 56 to an AND gate as shown in Fig. P. 7.6.11(a) and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-7151) Fig. P. 7.6.11(a) : Reset logic

Step 3 : Draw the logic diagram :



(C-7152) Fig. P. 7.6.11(b) : MOD-56 using IC 7490

- The logic diagram of the MOD-56 counter is shown in Fig. P. 7.6.11(b).

Ex. 7.6.12 : Design MOD 93 counter using IC 7490.

May 19, 6 Marks

Soln. :

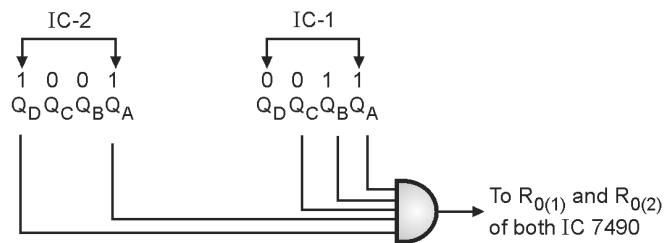
1. MOD 93 counter :

Step 1 : Number of ICs required :

- Upto MOD - 100, two IC 7490s will be sufficient..

Step 2 : Design of reset logic :

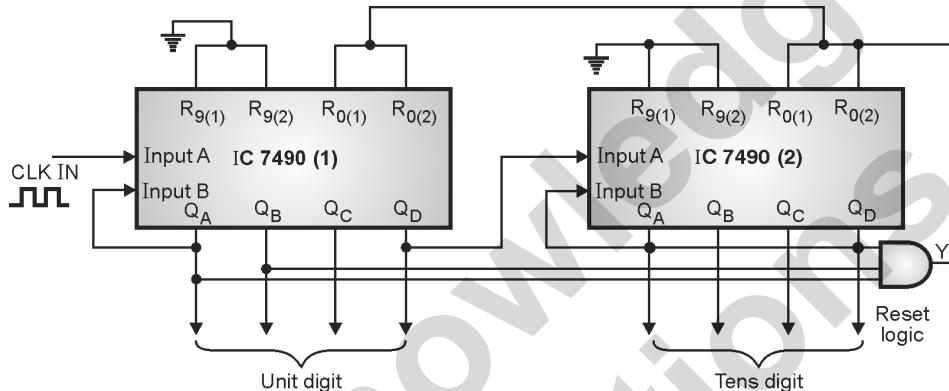
- Refer previous example for the procedure to design the reset logic.
- So in order to reset the counter at 93, connect the Q outputs which are equal to 1 in the count of 93 to an AND gate as shown in Fig. 2(a) and then connect the AND output to the $R_{0(1)}$ and $R_{0(2)}$ i.e. reset inputs of both the ICs.



(C-3603) Fig. P. 7.6.12(a) : Reset logic

Step 3 : Draw the logic diagram :

The logic diagram of the MOD-93 counter is shown in Fig. P. 7.6.12(b).



(C-7935) Fig. P. 7.6.12(b)

7.7 Problems Faced by Ripple Counters :

- The two major problems associated with the ripple counters are as follows :

 1. Generation of unwanted short duration pulses called glitch.
 2. Propagation delay.

7.8 Synchronous Counters :

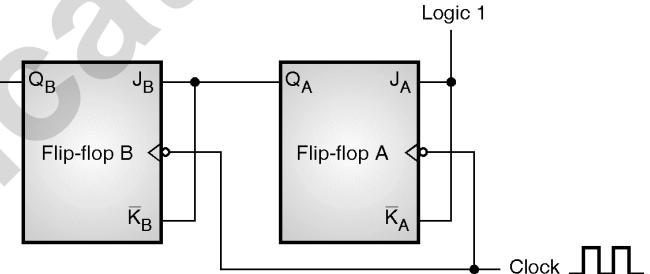
Definition :

- If the "clock" pulses are applied to all the flip-flops connected in a counter simultaneously, then such a counter is called as synchronous counter.
- These counters are also known as parallel counters. The state of all the flip flops will change simultaneously in the synchronous counter.

7.8.1 2-Bit Synchronous up Counter :

Logic diagram :

- A 2-bit or MOD-4 synchronous counter is shown in Fig. 7.8.1.



(C-815) Fig. 7.8.1 : A 2-bit (MOD-4) synchronous counter

- The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A .
- Hence FF-B will toggle if $Q_A = 1$ and there won't be any state change if $Q_A = 0$, at the instant when the negative clock edge is applied.

Operation :

- Initially let both the FFs be in the reset state. Let FF-A be the LSB flip-flop and FF-B be the MSB flip-flop.
 $\therefore Q_B \ Q_A = 0 \ 0 \dots$ Initially

At the 1st negative clock edge :

- As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1.



- But at the instant of application of negative clock edge, $Q_A = 0 \therefore J_B = K_B = 0$. Therefore FF-B will not change its state. So Q_B will remain 0.
 $\therefore Q_B Q_A = 01 \dots$ After the first clock pulse

At the 2nd negative clock edge :

- At the instant when we apply the second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0.
- But at this instant Q_A was 1. So $J_B = K_B = 1$ and FF-B also will toggle. Hence Q_B changes from 0 to 1.
 $\therefore Q_B Q_A = 10 \dots$ After the second clock pulse

Next negative clock edges :

- Similarly on application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B.
 $\therefore Q_B Q_A = 11 \dots$ After the third clock pulse
- On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0. Hence
 $\therefore Q_B Q_A = 00 \dots$ After the fourth clock pulse
- This is the original state. The operation of counter will repeat after this. The operation is summarised in Table 7.8.1 and the timing diagram is shown in Fig. 7.8.2.
- In this way the 2-bit synchronous counter has four distinct states namely $Q_B Q_A = 00, 01, 10$ and 11 .
- The maximum count of a 2-bit counter is $(11)_2$ or $(3)_{10}$.

(C-6224) Table 7.8.1 : Summary of operation of a 2-bit synchronous counter

Clock	Counter outputs	
	Q_B (MSB)	Q_A (LSB)
Initially	0	0
1 st (↓)	0	1
2 nd (↓)	1	0
3 rd (↓)	1	1
4 th (↓)	0	0

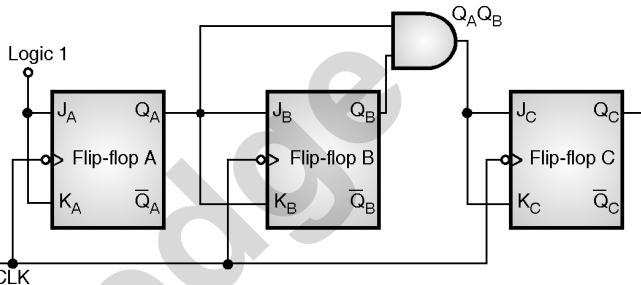
Clock	1	2	3	4	
Q_A	0	1	0	1	0
Q_B	0	0	1	1	0
$Q_B Q_A$ count	00	01	10	11	00
	0	1	2	3	Repeat

(C-816) Fig. 7.8.2 : Timing diagram for a 2-bit synchronous counter

7.8.2 3-Bit Synchronous Binary up Counter :

Logic diagram :

- We can extend the principle of operation of a 2-bit synchronous counter to a 3-bit counter shown in Fig. 7.8.3.



(C-817) Fig. 7.8.3 : A 3-bit synchronous binary counter

- FF-A acts as a toggle FF since $J_A = K_A = 1$.
- Q_A output of FF-A is applied to J_B as well as K_B . Hence if $Q_A = 1$ at the instant of triggering, then FF-B will toggle but if $Q_A = 0$ then FF-B will not change its state.
- Q_A and Q_B are ANDed and the output of AND gate is applied to J_C and K_C .
- Hence when Q_A and Q_B both are simultaneously high, then $J_C = K_C = 1$ and FF-C will toggle. Otherwise there is no change in the state of FF-C.

Operation :

- Initially all the FFs are in their reset state.
 $\therefore Q_C Q_B Q_A = 000$

1st clock pulse :

- FF-A toggles and Q_A changes to 1 from 0. But since $Q_A = 0$ at the instant of application of 1st falling clock edge, $J_B = K_B = 0$ and Q_B does not change state.
 $\therefore Q_B$ remains 0.
- Similarly Q_C also does not change state.

$$\therefore Q_C = 0.$$

$$\therefore Q_C Q_B Q_A = 001 \dots \text{After } 1^{\text{st}} \text{ clock pulse}$$

2nd clock pulse :

- FF-A toggles and Q_A becomes 0.
- But at the instant of application of 2nd falling clock edge Q_A was equal to 1. Hence $J_B = K_B = 1$. Hence FF-B will toggle and Q_B becomes 1.
- Output of AND gate is 0 at the instant of negative clock edge. So $J_C = K_C = 0$. Hence Q_C remains 0.



$\therefore Q_C Q_B Q_A = 0 1 0$...After the 2nd clock pulse

3rd clock pulse :

- After the 3rd clock pulse, the outputs are $Q_C Q_B Q_A = 0 1 1$.

4th clock pulse :

- Note that $Q_B = Q_A = 1$. Hence output of AND gate = 1 and $J_C = K_C = 1$, at the instant of application of 4th negative edge of the clock.
- Hence on application of this clock pulse, FF-C will toggle and Q_C changes from 0 to 1.
- FF-A toggles as usual and Q_A becomes 0.
- Since Q_A was equal to 1 earlier, FF-B will also toggle to make $Q_B = 0$.

$\therefore Q_C Q_B Q_A = 1 0 0$...After the 4th clock pulse

- Thus the counting progresses.
- After the 7th clock pulse the output is 111 and after the 8th clock pulse, all the flip-flops toggle and change their outputs to 0. Hence $Q_C Q_B Q_A = 0 0 0$ after the 8th pulse and the operation repeats.
- Table 7.8.2 summarizes operation of the three bit synchronous counter.

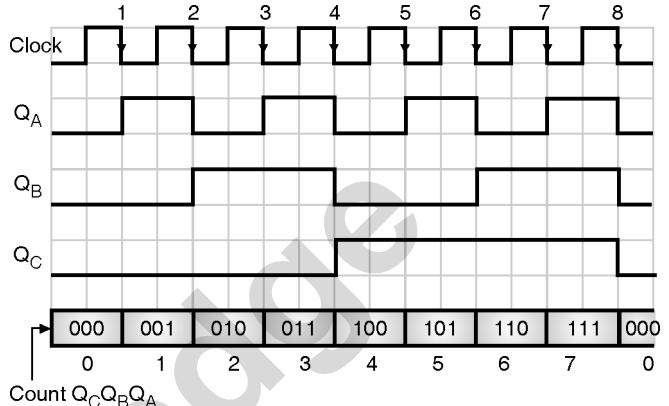
(C-6225) Table 7.8.2 : Summary of operation of a 3-bit synchronous counter

Clock	Q_C	Q_B	Q_A
0	0	0	0
1 st (\downarrow)	0	0	1
2 nd (\downarrow)	0	1	0
3 rd (\downarrow)	0	1	1
4 th (\downarrow)	1	0	0
5 th (\downarrow)	1	0	1
6 th (\downarrow)	1	1	0
7 th (\downarrow)	1	1	1

Timing diagram :

- Timing diagram for a 3-bit synchronous counter is shown in Fig. 7.8.4.
- The number of states through which this counter progresses is 8 namely $Q_C Q_B Q_A = 000, 001, 010, 011, 100, 101, 110, 111$.
- The maximum count is $(111)_2$ or $(7)_{10}$.

- Note that the waveforms of synchronous counter are exactly same as those of an asynchronous counter.



(C-818) Fig. 7.8.4 : Timing diagram for a 3-bit synchronous counter

7.8.3 Design of the 3 Bit Synchronous Counter :

SPPU : May 12, Dec. 19

University Questions

- Q. 1** Design 3-bit synchronous up-counter using MS JK-flip-flop. **(May 12, 6 Marks)**
- Q. 2** Design 3-bit Synchronous up counter with JK flip-flops. **(Dec. 19, 6 Marks)**

- Let us now design a 3 bit synchronous counter first using the T flip flops and then using JK flip flops.

Design using T flip flops :

Steps to be followed :

- Step 1 :** Decide the number of flip flops.
- Step 2 :** Write the excitation table of T flip flop.
- Step 3 :** Write the excitation table of the counter.
- Step 4 :** From the circuit excitation table write K-maps and obtain simplified equations.
- Step 5 :** Draw the logic diagram.

Step 1 : Decide number of FFs :

- A 3 bit counter goes through 8 states. So it needs three flip flops.

Step 2 : Excitation table of T FFs :

- Table 7.8.3(a) shows the excitation table of T FF.

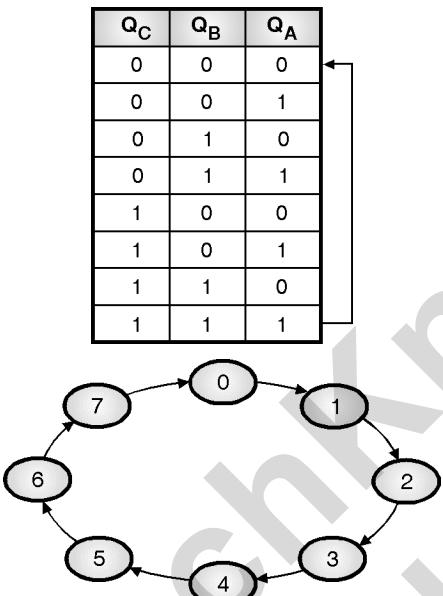
(C-7842) Table 7.8.3(a) : Excitation table of a T FF

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 3 : State diagram and circuit excitation table :

- Count sequence for a 3 bit up counter is given in Table 7.8.3(b) and Fig. 7.8.5 shows the corresponding state diagram.

(C-6226) Table 7.8.3(b)



(C-819) Fig. 7.8.5 : State diagram

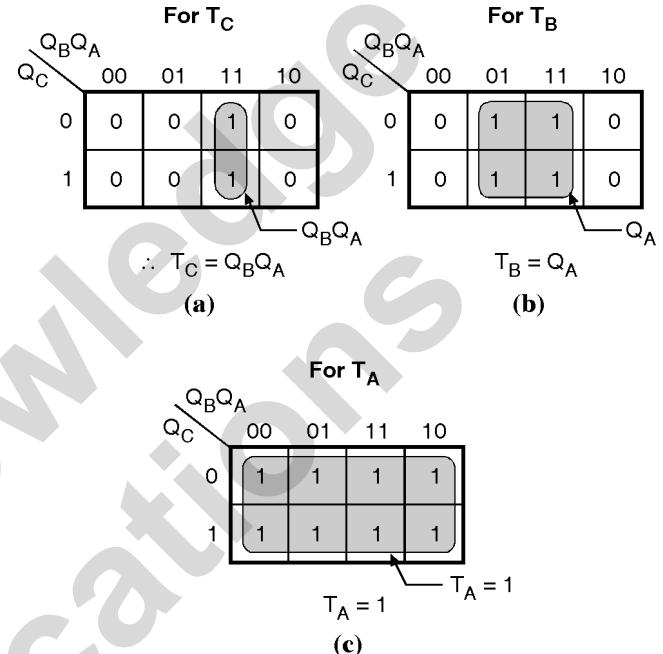
- Table 7.8.3(c) shows the circuit excitation table.

(C-7838) Table 7.8.3(c) : Circuit excitation table

Present state			Next state			Flip flop input		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	1	1

- Table 7.8.3(c) has been written by referring to the present and next state of an output and the required value to input is written as per the excitation table of T FF.

- For example consider the shaded columns of Table 7.8.3(c). i.e. Q_C , Q_{C+1} and T_C
- Consider the first row. $Q_C = 0$, $Q_{C+1} = 0$. So as per the excitation table of a T FF T_C should be 0. Similarly all other entries are made.

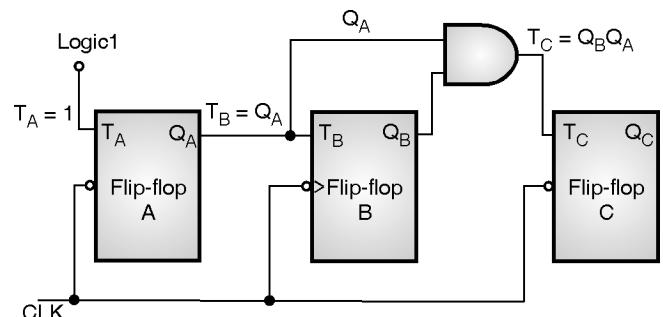
Step 4 : Write K maps and obtain simplified equations :

(C-1506) Fig. 7.8.6 : K-maps for different FF inputs

- Refer Figs. 7.8.6(a), (b), (c) for the K-maps corresponding to all the FF inputs. The simplified equations for T_A , T_B and T_C also are shown.

Step 5 : Draw the logic diagram :

- By using the simplified equations for T_A , T_B and T_C we can draw the logic diagram for the 3 bit synchronous shown in Fig. 7.8.7.



(C-1507) Fig. 7.8.7 : Logic diagram of a 3-bit synchronous counter



Design using JK flip flops :

Step 1 : Number of flip flops :

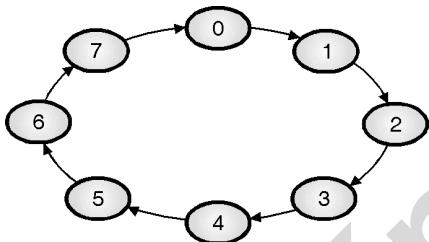
- For a 3 bit counter, we need 3 flip-flops.

Step 2 : Excitation table of JK FF :

(C-7761) Table 7.8.4 : Excitation table of JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Step 3 : State diagram and circuit excitation table :



(C-1508) Fig. 7.8.8 : State diagram

(C-7050) Table 7.8.5 : Circuit excitation table

Present state	Next state			Flip flop inputs								
	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0 0 0	0	0	1	0	0	1	0	X	0	X	1	X
0 0 1	0	1	0	0	1	0	0	X	1	X	X	1
0 1 0	0	1	1	0	1	1	0	X	X	0	1	X
0 1 1	1	0	0	1	0	0	1	X	X	1	X	1
1 0 0	1	0	1	1	0	1	X	0	0	X	1	X
1 0 1	1	1	0	1	1	0	X	0	1	X	X	1
1 1 0	1	1	1	0	1	1	X	0	X	0	1	X
1 1 1	0	0	0	0	0	0	X	1	X	1	X	1

Step 4 : K maps and simplified expressions for all FF inputs :

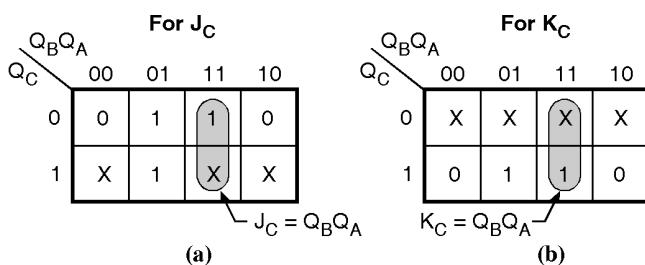
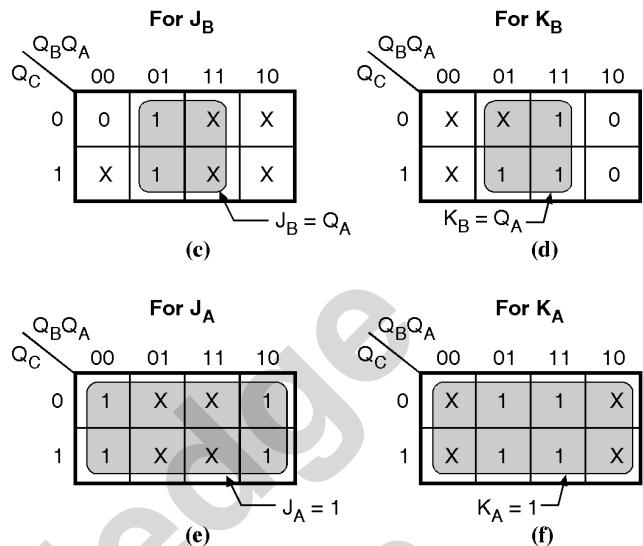


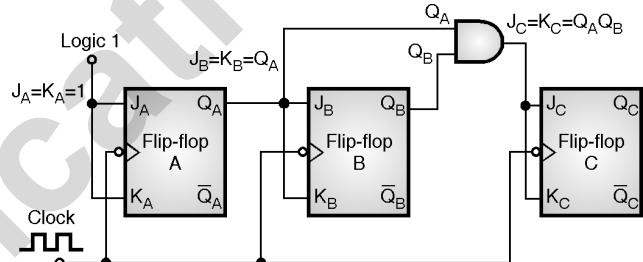
Fig. 7.8.9 (cont...)



(C-1509) Fig. 7.8.9 : K-maps and simplified equations for different FF inputs

Step 5 : Logic diagram :

- Fig. 7.8.10 shows the logic diagram of a 3 bit synchronous counter using JK flip-flops.



(C-1510) Fig. 7.8.10 : 3 bit synchronous counter using JK FFs

7.8.4 Four Bit Synchronous Up Counter :

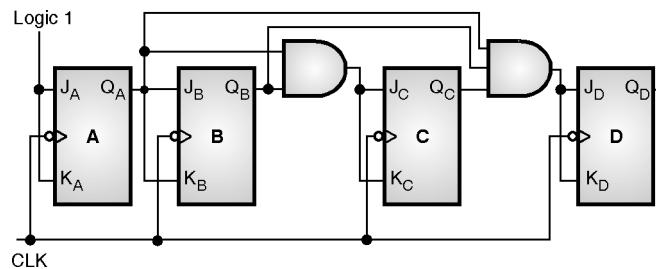
SPPU : Dec. 10

University Questions

- Q. 1** Draw a 4-bit synchronous counter. Also explain timing diagram for the same. (Dec. 10, 10 Marks)

Logic diagram :

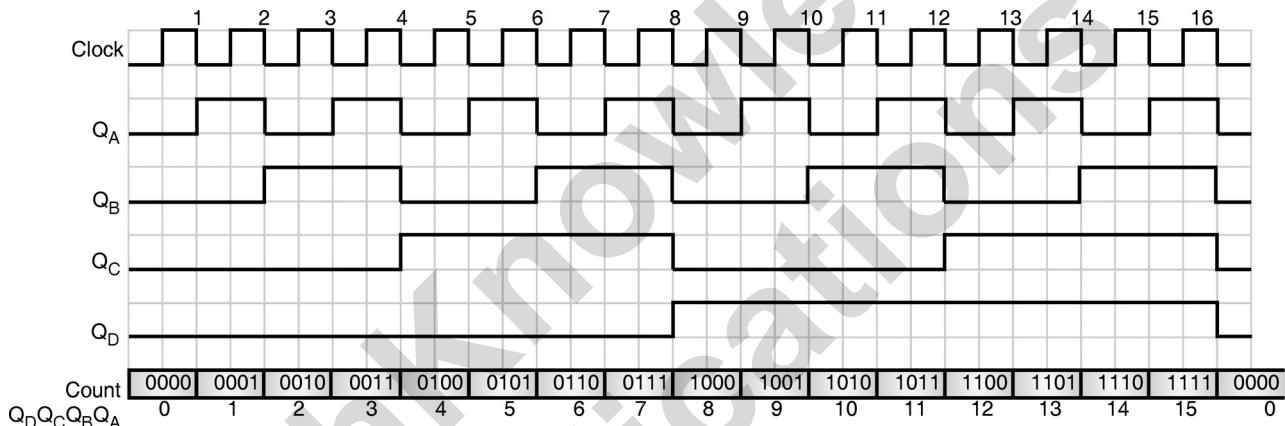
- All the concepts of a synchronous counter are extended to design a 4-bit synchronous counter shown in Fig. 7.8.11.
- Note that J_D and K_D of the FF-D are connected to the output of an AND gate and the inputs of this AND gate come from the outputs of the three preceding FFs.
- Operating principle of this counter is same as that of the 3-bit counter discussed earlier.



(C-1391) Fig. 7.8.11 : A four bit synchronous counter

Timing diagram :

- The timing diagram of a four bit synchronous counter is as shown in Fig. 7.8.12.
- Note that the principle of frequency division is applicable to the synchronous counters as well.



(C-1392) Fig. 7.8.12 : Timing diagram of a 4-bit synchronous counter

7.9 Modulo – N Synchronous Counters :

SPPU : May 06, Dec. 06

University Questions**Q. 1** What is MOD counter ?

(May 06, Dec. 06, 5 Marks)

- We have already discussed the MOD-N asynchronous counters. Now let us discuss the Modulo-N synchronous counters.
- The steps to be followed to design a MOD-N synchronous counter are as follows.

Steps to be followed :**Step 1 :** Decide the number of FFs and type of FF to be used.**Step 2 :** Write the circuit excitation table.**Step 3 :** From the circuit excitation table write down the K-maps and obtain simplified expressions for the outputs.**Step 4 :** Draw the logic diagram and timing diagram.**7.9.1 Synchronous Decade Counter :**

- For the design of a synchronous decade counter follow the steps given below :

Step 1 : Write the excitation table for T FF and circuit excitation table :

- Excitation table for T FF is shown in Table 7.9.1.

(C-7842) Table 7.9.1 : Excitation table for T flip-flop

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

- The circuit excitation table is shown in Table 7.9.2.

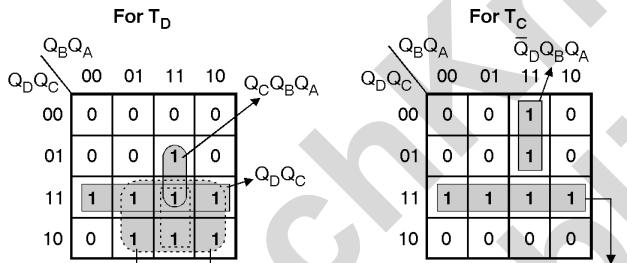


(C-7839) Table 7.9.2 : Circuit excitation table

Present state				Next state				Flip flop inputs			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	0	0	0	1	0	1	0
1	0	1	1	0	0	0	0	1	0	1	1
1	1	0	0	0	0	0	0	1	1	0	0
1	1	0	1	0	0	0	0	1	1	0	1
1	1	1	0	0	0	0	0	1	1	1	0
1	1	1	1	0	0	0	0	1	1	1	1

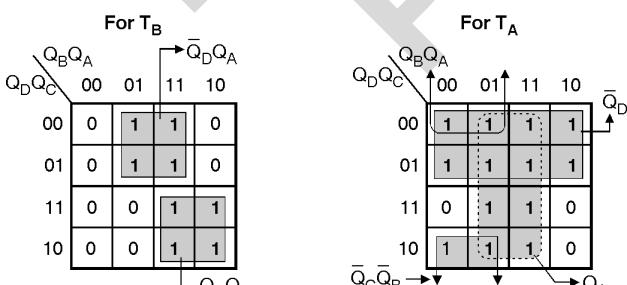
Step 2 : K-maps and simplifications :

- K-maps for T_D , T_C , T_B , T_A and their simplified expressions are given in Figs. 7.9.1(a), (b), (c) and (d).



$$T_D = Q_C Q_B Q_A + Q_D Q_C + Q_D Q_B + Q_D Q_A \quad T_C = \bar{Q}_D Q_B Q_A + Q_D Q_C$$

$$\therefore T_D = Q_C Q_B Q_A + Q_D (Q_C + Q_B + Q_A)$$

(a) K map for T_D 

$$\therefore T_B = Q_D Q_B + \bar{Q}_D Q_A$$

(b) K map for T_C

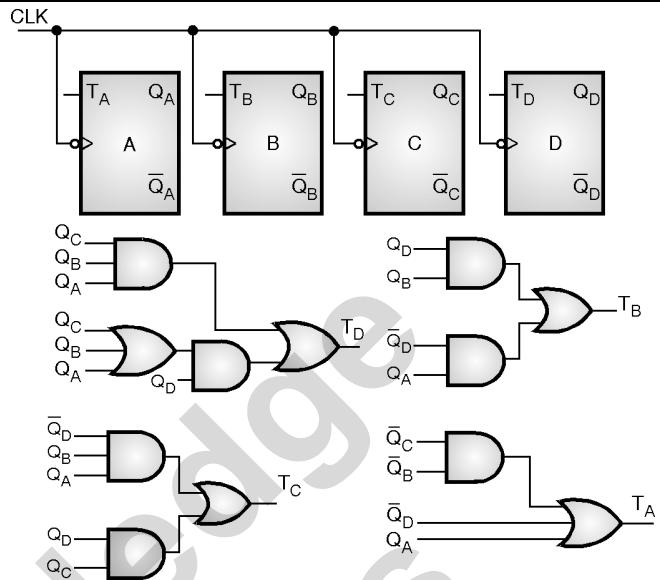
$$T_A = \bar{Q}_D + \bar{Q}_C \bar{Q}_B + Q_A$$

(d) K map for T_A

(C-847) Fig. 7.9.1

Step 3 : Draw the logic diagram :

- Fig. 7.9.2 shows the logic diagram of a synchronous decade counter.



(C-848) Fig. 7.9.2 : Logic diagram of a synchronous decade counter

Ex. 7.9.1 : Design a synchronous counter for the sequence shown in Fig. P. 7.9.1(a).

May 06, 8 Marks

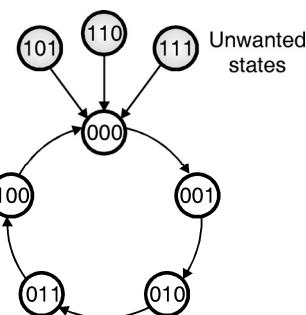
(C-6227) Table P. 7.9.1(a) : Desired sequence

Q_C	Q_B	Q_A
0	0	0 ←
0	0	1
0	1	0
0	1	1
1	0	0

Soln. :

Step 1 : Determine the desired number of FFs :

- From the given sequence the number of FFs is equal to 3. This is a MOD-5 synchronous counter since the number of states is 5.
- The state diagram is shown in Fig. P. 7.9.1(a) which shows that 101, 110, 111 are unwanted states.



(C-849) Fig. P. 7.9.1(a) : State diagram



Step 2 : Write the excitation table and state table :

- The type of FF used is JK flip-flop. The excitation table for a JK FF is as shown in Table P. 7.9.1(b).
- We have already seen how to write the excitation table for JK FF.

Excitation table of JK FF :

- Table P. 7.9.1(b) exhibits the excitation table of a JK FF.

(C-7840) Table P. 7.9.1(b) : Excitation table of JK FF

Present state Q_n	Next state Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Circuit excitation table :

- The circuit excitation table is as shown in Table P. 7.9.1(c).

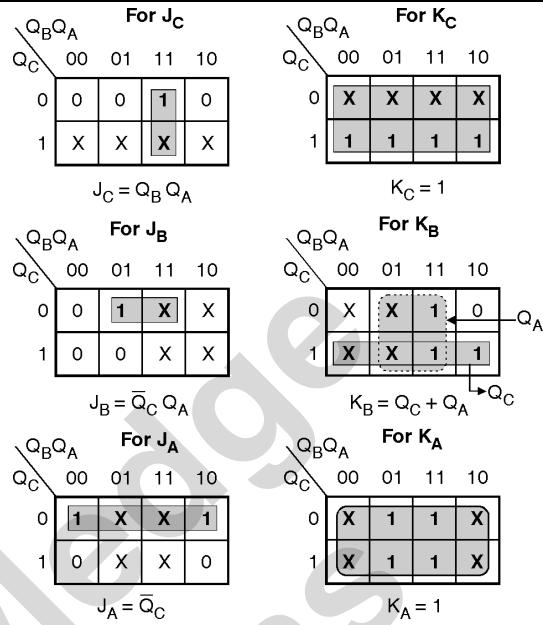
(C-7841) Table P. 7.9.1(c) : Circuit excitation table

Present state			Next state			Flip flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
1	0	0	0	0	1	0	x	0	x	1	x
1	0	1	0	1	0	0	x	1	x	x	1
1	1	0	0	1	1	0	x	x	x	0	1
1	1	1	1	0	0	1	x	x	x	1	x
0	0	0	0	0	0	x	1	0	x	0	x
0	0	1	0	0	0	x	1	0	x	x	1
0	1	0	0	0	0	x	1	x	1	0	x
0	1	1	0	0	0	x	1	x	1	x	1

- Refer to the shaded portion of the circuit excitation table. This is nothing but the excitation table of FF-C. The J_C and K_C values have been decided based on Q_C and Q_{C+1} .
- Similarly the entries for J_B and K_B are based on Q_B and Q_{B+1} whereas those for J_A and K_A are based on Q_A and Q_{A+1} .

Step 3 : K-maps and simplifications :

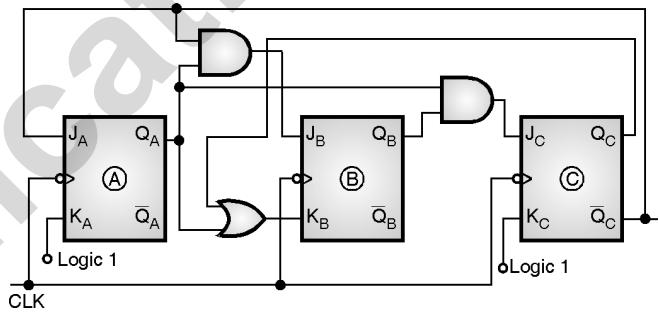
- K-maps for the J and K inputs of all the FFs and the corresponding simplified equations are shown in Fig. P. 7.9.1(b).
- Note that the inputs to this combinational circuit are Q_A , Q_B , Q_C and outputs are J_A , K_A through J_C , K_C .



(C-850) Fig. P. 7.9.1(b) : K-maps and simplifications

Step 4 : Logic diagram :

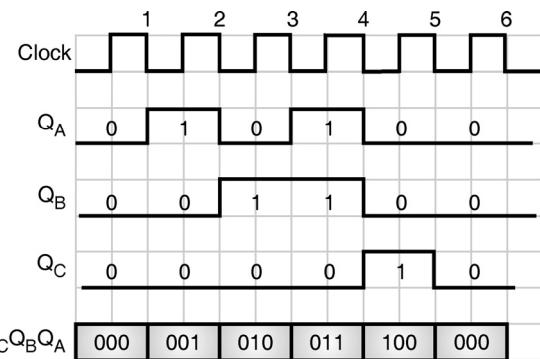
- The logic diagram of MOD-5 synchronous counter is shown in Fig. P. 7.9.1(c).



(C-851) Fig. P. 7.9.1(c) : Logic diagram of MOD-5 synchronous counter

Step 5 : Draw the timing diagram :

- The timing diagram of MOD-5 synchronous counter is shown in Fig. P. 7.9.1(d).



(C-852) Fig. P. 7.9.1(d) : Timing diagram of MOD-5 counter

Ex. 7.9.2 : Design a MOD-5 synchronous counter using T flip-flops.

Soln. :

Step 1 : Decide number of FFs :

- Since the number of states is 5 we need to use 3 flip-flops.

Step 2 : Excitation tables :

- The excitation table of a T flip-flop is shown in Table P. 7.9.2(a).

(C-7842) Table P. 7.9.2(a) : Excitation table of a T flip-flop

Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Circuit excitation table :

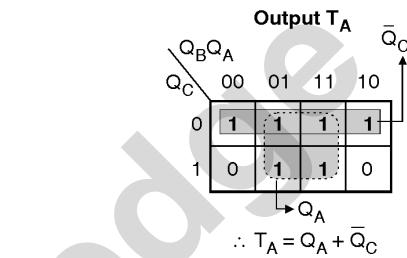
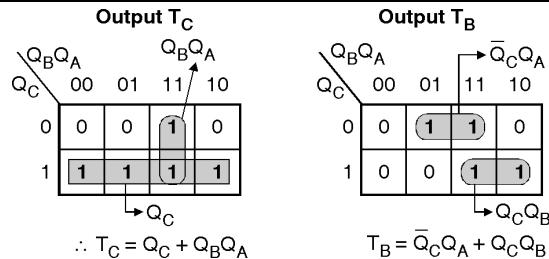
- The circuit excitation table is shown in Table P. 7.9.2(b).
- Refer to the shaded portion of the circuit excitation table.
- This is nothing but the excitation table of FF-C. The T_C values are decided based on Q_C and Q_{C+1} .
- Similarly the entries for T_B are based on Q_B and Q_{B+1} whereas those for T_A are based on Q_A and Q_{A+1} .

(C-7843) Table P. 7.9.2(b) : Circuit excitation table

Present state	Next state			Flip flop inputs		
	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0 0 0	0	0	1	0	0	1
0 0 1	0	1	0	0	1	1
0 1 0	0	1	1	0	0	1
0 1 1	1	0	0	1	1	1
1 0 0	0	0	0	1	0	0
1 0 1	0	0	0	1	0	1
1 1 0	0	0	0	1	1	0
1 1 1	0	0	0	1	1	1

Step 3 : K-maps and simplifications :

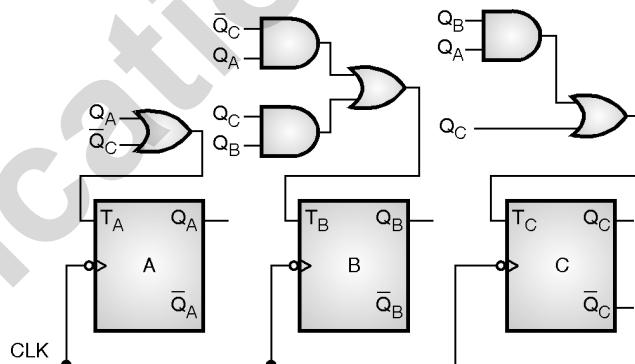
- K-maps for the T_A , T_B and T_C inputs of the three FFs and the corresponding simplified equations are shown in Fig. P. 7.9.2(a).



(C-853) Fig. P. 7.9.2(a) : K-maps and simplifications

Step 4 : Draw the logic diagram :

- The logic diagram of MOD-5 synchronous counter using T FFs is shown in Fig. P. 7.9.2(b).



(C-854) Fig. P. 7.9.2(b) : MOD-5 synchronous counter using T flip-flops

7.10 UP / DOWN Synchronous Counter :

- The operating principle of a synchronous up down counter is same as that of the up/down ripple counters. But the design procedure and realization are different.
- The mode control (M) is used for selecting the mode of operation.
- The steps to be followed for the design are as follows :

Steps to be followed :

Step 1 : Write the circuit excitation table.

Step 2 : Write K-maps and obtain the simplified expressions.

Step 3 : Draw the logic diagram.

7.10.1 3-bit Up/Down Synchronous Counter :

SPPU : Dec. 08, May 10

University Questions

Q. 1 Design and implement 3 bit up/down synchronous counter using MS-JK flip-flop with its truth table.
Also draw timing diagram. (Dec. 08, 12 Marks)

Q. 2 Explain with a neat diagram working of 3-bit up-down synchronous counter. Draw necessary timing diagram. (May 10, 10 Marks)

- The number of FFs to be used is three. We will use three toggle flip-flops.
- Let upcounting takes place with $M = 0$ and down counting takes place for $M = 1$.

Step 1 : Write the circuit excitation table :

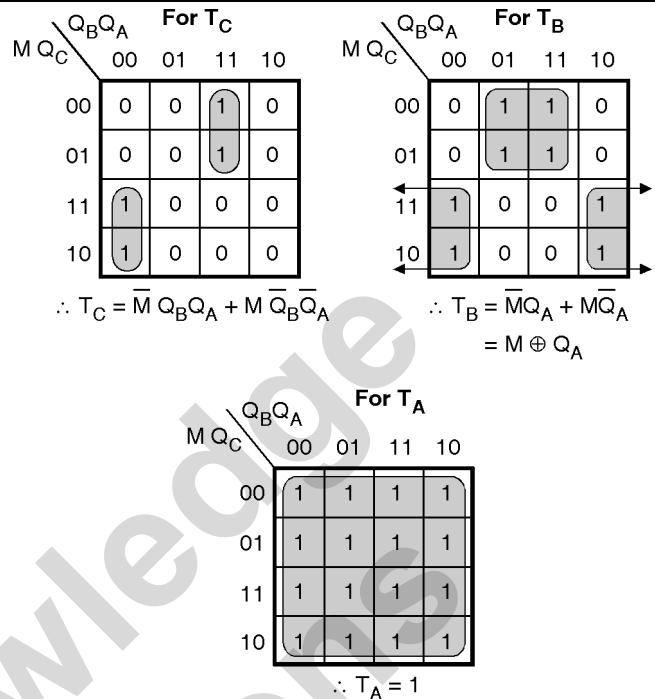
- The circuit excitation table is shown in Table 7.10.1.

(C-6228) Table 7.10.1 : Excitation table for a 3-bit up/down synchronous counter

Mode control M	Present state			Next state			Flip-flop inputs		
	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

Step 2 : K-maps and simplified equations :

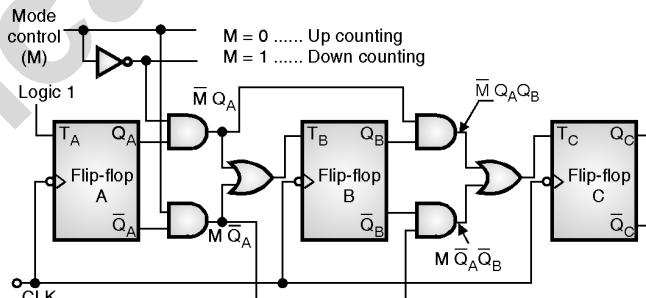
- The K-maps and simplified equations for T_C , T_B and T_A are shown in Fig. 7.10.1.



(C-855) Fig. 7.10.1 : K-maps and simplified expressions

Step 3 : Draw the logic diagram :

- The logic diagram for a 3-bit synchronous up down counter is shown in Fig. 7.10.2.



(C-856) Fig. 7.10.2 : Logic diagram of a 3-bit synchronous up down counter

7.10.2 Advantages of Synchronous Counter :

- All the FFs receive the clock signal simultaneously i.e. at the same instant, the propagation delay problem is reduced to a great extent. Recall that the propagation delay is one of the problems of the ripple counter.
- The total propagation delay between the instant of application of clock edge and the instant at which the MSB output changes is equal to sum of propagation delay of one FF and that of one AND gate.



- \therefore Total propagation delay = t_d of one FF + t_d of one gate
 3. This is much less than the propagation delay of an asynchronous (ripple) counter.

Due to reduced propagation delay, the synchronous counters can operate at a much higher clock frequency than the asynchronous counters.

7.10.3 Comparison of Synchronous and Asynchronous Counters :

SPPU : May 12, Dec. 12,
May 14, Dec. 16, May 18

Sr. No.	Parameter	Asynchronous counter	Synchronous counter
4.	Propagation delay	P.D. = $n \times (t_d)$ where n is number of FFs and t_d is p.d. per FF.	P.D. = $(t_d)_{FF} + (t_d)_{gate}$. It is much shorter than that of asynchronous counter.
5.	Maximum frequency of operation	Low because of the long propagation delay.	High due to shorter propagation delay.

University Questions

- Q. 1 What is the difference between synchronous counters and asynchronous counters ?

(May 12, Dec. 12, 4 Marks)

- Q. 2 Explain the difference between asynchronous and synchronous counter and convert J-K flip flop into D-FF. Show the design. (May 14, 6 Marks)

- Q. 3 Explain the difference between asynchronous and synchronous counter and convert SR flip-flop into T flip-flop. Show the design. (Dec. 16, 6 Marks)

- Q. 4 Compare Asynchronous counter with Synchronous counter. Design MOD.11 up counter using IC 74191. (May 18, 6 Marks)

- Comparison of synchronous and asynchronous counters is given in Table 7.10.2.

Table 7.10.2 : Comparison of synchronous and asynchronous counters

Sr. No.	Parameter	Asynchronous counter	Synchronous counter
1.	Circuit complexity	Logic circuit is simple	With increase in number of states, the logic circuit becomes complicated.
2.	Connection pattern	Output of the preceding FF, is connected to clock of the next FF.	There is no connection between output of preceding FF and CLK of next one.
3.	Clock input	All the FFs are not clocked simultaneously.	All FFs receive clock signal simultaneously.

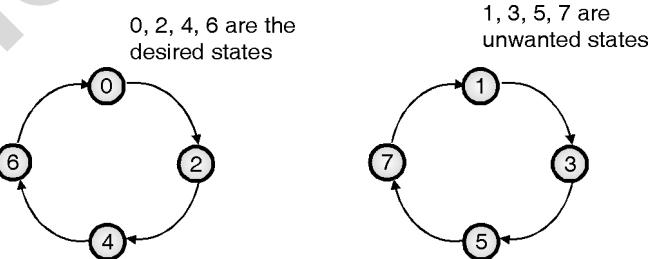
7.11 Lock Out Condition :

SPPU : Dec. 08

University Questions

- Q. 1 What is lock out condition ? (Dec. 08, 3 Marks)

- A counter is supposed to follow the sequence of only the desired states as shown in Fig. 7.11.1(a).
- If it enters into an unused or unwanted state, then it is expected to return back to a desired state.
- Instead if the next state of an unwanted state is again an unwanted state as shown in Fig. 7.11.1(b) then the counter is said to have gone into the lockout conditions.



(a) State diagram showing the desired sequence (b) State diagram showing the lockout condition

(C-831) Fig. 7.11.1

7.11.1 Bushless Circuit :

SPPU : Dec. 08

University Questions

- Q. 1 How lock out condition avoided ?

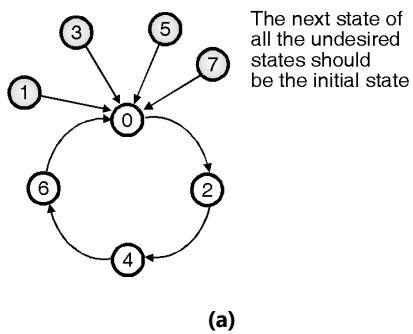
(Dec. 08, 3 Marks)

Definition :

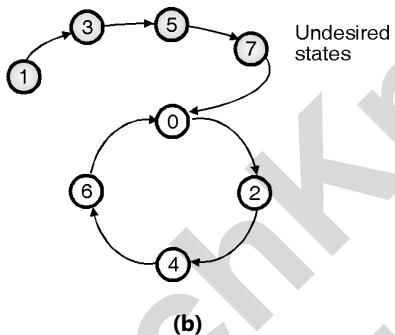
- The sequential circuit which enters into the lockout condition is called as the bushless circuit.

How to avoid lockout situation ?

- In order to avoid the lockout condition, we have to use some additional logic circuit to ensure that the next state of the counter is its initial state, if it enters into an undesired state.
- Thus the next state of each unwanted state should be the initial state as shown in Fig. 7.11.2(a).



(a)



(b)

(C-832) Fig. 7.11.2 : Ways to avoid lockout condition

- However it is not necessary to terminate all the undesired states into the initial state as shown in Fig. 7.11.2(a).
- It will be sufficient to terminate only one unused state into the initial state as shown in Fig. 7.11.2(b).
- If the circuit enters into say state "3" then its next states will be "5" and "7". As "7" has next state of "0" which is the desired state, the lockout condition is avoided.

Sequential counter design using JK and T flip flops :

Ex. 7.11.1 : By using suitable flip-flops design a counter to go through the following states.

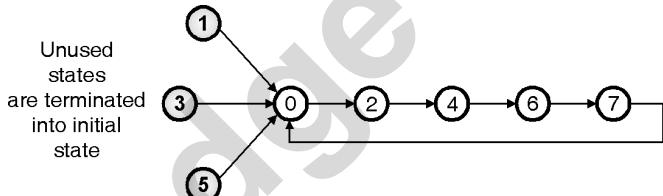
0 – 2 – 4 – 6 – 7 – 0

Avoid the lockout condition.

Soln. :

Step 1 : Draw the state diagram :

- The state diagram is shown in Fig. P. 7.11.1(a). It shows that the next state of all the unused states (1, 3 and 5) is forced to be the initial state (0).
- This is essential in order to avoid the lockout condition.

**(C-833) Fig. P. 7.11.1(a) : State diagram of desired counter**

Step 2 : Number of flip-flops and type of flip-flop :

- Since the highest state is 7 i.e. 111 we need to use three flip-flops. Let us use JK flip-flops.

Step 3 : Write the state table :

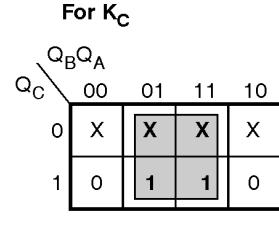
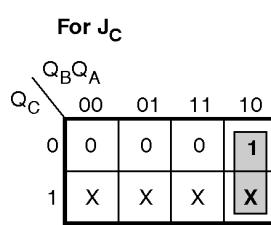
- Table P. 7.11.1(a) shows the state table for the required counter.

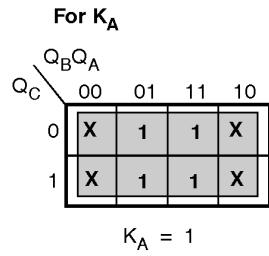
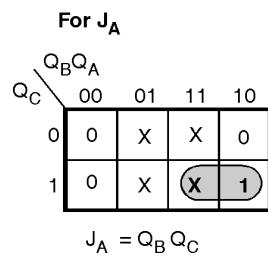
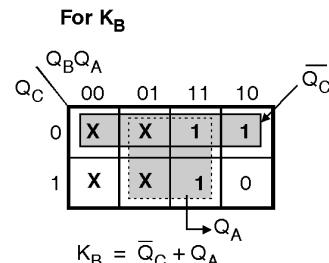
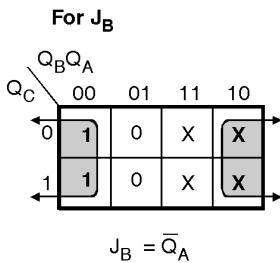
(C-8053)Table P. 7.11.1(a)

Present state			Next state			Flip flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Step 4 : K maps and simplification :

- Fig. P. 7.11.1(b) illustrates the K-maps for all the FF inputs and the corresponding simplified expressions.

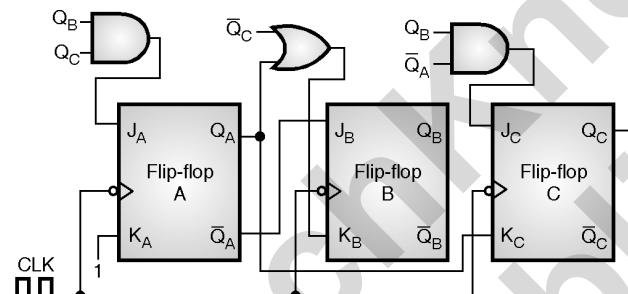
**Fig. P. 7.11.1(b) (cont...)**



(C-834) Fig. P. 7.11.1(b) : K-maps and simplification

Step 5 : Draw the logic diagram :

- The logic diagram for the required counter is shown in Fig. P. 7.11.1(c).

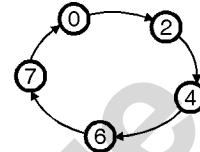


(C-835) Fig. P. 7.11.1(c) : Logic circuit of the required counter

7.12 Bush Diagram :

- In the previous section we have discussed the meaning of lock-out condition.
- The lock-out condition can be avoided by taking a precaution called "**bushing**". Bushing means adding branches in the state diagram.
- The invalid states of a counter are branched in the bush diagram in such a way that even if the counter enters into an invalid state, it will return to one of its valid states after one, two or three clock cycles.
- The concept of bushing can be well understood by referring to the following example.

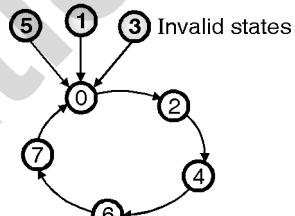
Ex. 7.12.1 : For the given state diagram in Fig. P. 7.12.1(a) of a counter draw the bush diagrams required to bring the counter from invalid to a valid state in one, two or three clock cycles.



(C-842) Fig. P. 7.12.1(a) : Given state diagram

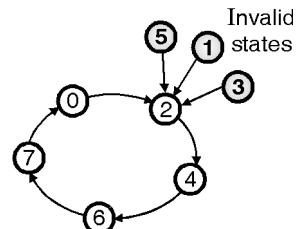
Soln. :

- Refer Figs. P. 7.12.1(b) to (d).
- Fig. P. 7.12.1(b) shows that the next state of each invalid state 1, 3 or 5 is the initial state 0. Hence even if the counter enters into an unused state, it will return to a valid state after only one clock cycle.



(C-842) Fig. P. 7.12.1(b) : Entry from invalid to valid state after only one clock pulse

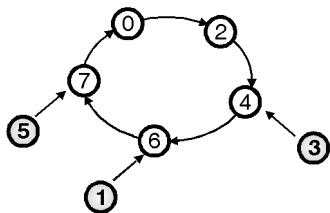
- Similarly for the state diagram shown in Fig. P. 7.12.1(c), it will take only one clock pulse to bring the counter into a used state if it enters into an unused one.



(C-842) Fig. P. 7.12.1(c) : Entry from invalid to valid state after only one clock pulse

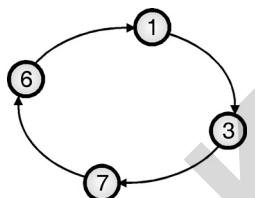
- Refer the state diagram of Fig. P. 7.12.1(d). If the counter enters into state "7" then it will return to "0" after one clock pulse.

- If it is in the state "5" then it will return to 0 state after two clock pulses and if it is in the state "1" then it will return to "0" state after three clock pulses.



(C-842) Fig. P. 7.12.1(d) : Entry from invalid to valid state after 1, 2 or 3 clock cycles

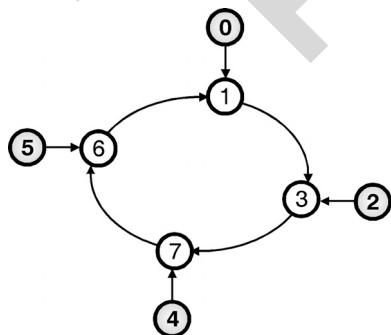
Ex. 7.12.2 : Design a synchronous counter from the state diagram shown in Fig. P. 7.12.2(a). Avoid lockout condition. Draw the bush diagram to avoid the lockout condition.



(C-843) Fig. P. 7.12.2(a) : Given state diagram

Soln. :

- So as to avoid the lockout condition, we have to ensure that the counter should return back to a valid state, as soon as it enters into an unused state. The bush diagram is shown in Fig. P. 7.12.2(b).



(C-844) Fig. P. 7.12.2(b) : Bush diagram

- 0, 2, 4 and 5 are the unused states. They are forcibly terminated into 1, 3, 7 and 6 respectively.

Step 1 : Number of FFs and the type of FF :

- Let us use T-type FF.
- Since the highest state in the state diagram is 7, we have to use three T flip-flops.

Step 2 : Write the circuit excitation table :

- Let us obtain the circuit excitation table from the bush diagram of Fig. P. 7.12.2(b).
- The excitation table for a T flip-flop is shown in Table P. 7.12.2(a) and the circuit excitation table is shown in Table P. 7.12.2(b).

(C-7842) Table P. 7.12.2(a) : Excitation table for a toggle FF

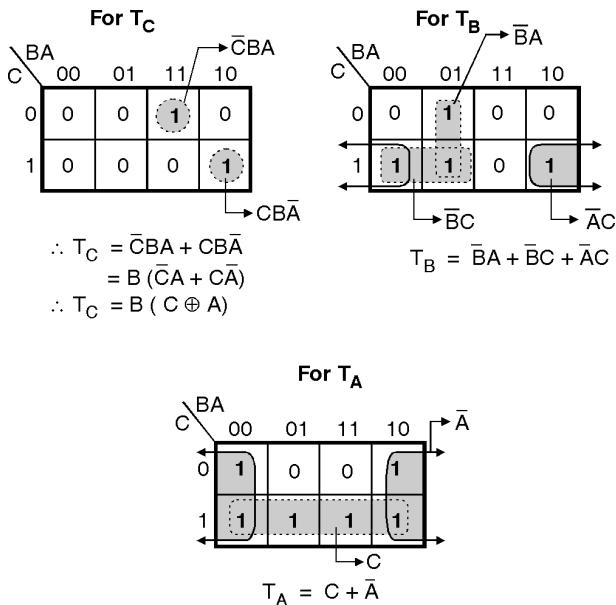
Present state Q_n	Next state Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

(C-6229) Table P. 7.12.2(b) : Circuit excitation table

→	Present state			Next state			FF inputs		
	C	B	A	C_{n+1}	B_{n+1}	A_{n+1}	T_C	T_B	T_A
→	0	0	0	0	0	1	0	0	1
→	0	0	1	0	1	1	0	1	0
→	0	1	0	0	1	1	0	0	1
→	0	1	1	1	1	1	1	0	0
→	1	0	0	1	1	1	0	1	1
→	1	0	1	1	1	0	0	1	1
→	1	1	0	0	0	1	1	1	1
→	1	1	1	1	1	0	0	0	1
→ Invalid states									

Step 3 : Draw the K-maps and obtain simplified expressions :

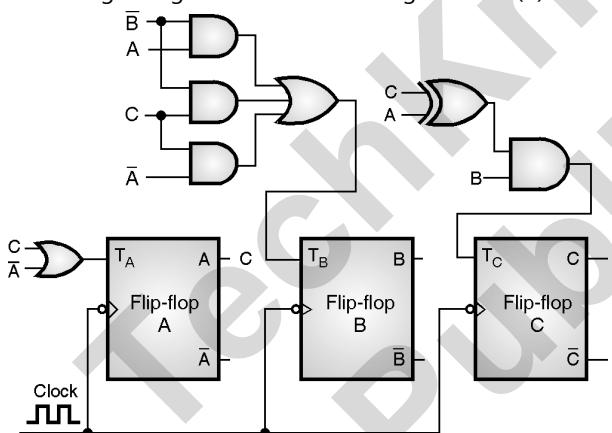
- Fig. P. 7.12.2(c) shows the K-maps for various T inputs. The simplified Boolean expressions also are given alongwith the K-maps.



(C-845) Fig. P. 7.12.2(c) : K-maps and simplifications

Step 4 : Logic diagram :

- The logic diagram is as shown in Fig. P. 7.12.2(d).



(C-846) Fig. P. 7.12.2(d) : Logic diagram of the required counter

7.13 Applications of Counters :

Some of the applications of counters are :

- In digital clock.
- In the frequency counters.
- In time measurement.
- In digital voltmeters.
- In the counter type A to D converter.

- In the digital triangular wave generator.

- In the frequency divider circuits.

Review Questions

- What is counter ? What are its types ?
- Explain the terms "synchronous" and "asynchronous".
- What is the difference between synchronous and asynchronous counter ?
- What are the merits and demerits of the synchronous counter over the asynchronous counter ?
- State the procedure for designing mod counters from N-bit ripple counter.
- What is the similarity between a decade counter and a binary counter ? Explain in brief.
- What are the applications of counter ?
- How is a counter different from a register ?
- Write the count sequence of three bit binary down counter.
- How does a counter works as frequency divider ?
- What is synchronous counter ?
- Define modulus of counter. State the states of MOD 5 counter.
- Explain the working of 4-bit ripple counter (asynchronous counter) using T flip-flop with suitable circuit diagram and timing diagram.
- Write a note on Mod-8 synchronous up-counter using T flip-flop.
- Write a note on Mod-8 synchronous down counter using T flip-flop.
- Draw the divide by 7 asynchronous up counter using T flip flop. Write truth table. Draw timing diagram.



- | | | | |
|-------|--|-------|---|
| Q. 17 | Draw the circuit for mod-12 counter. Explain the same with neat waveforms. | Q. 20 | Design a 3 bit synchronous counter using JK flip-flops. |
| Q. 18 | Compare synchronous and ripple counters. | Q. 21 | What is lock out ? How is it avoided ? |
| Q. 19 | Compare counters and registers. | Q. 22 | Explain decade counter. |

□ □ □

TechKnowledge
Publications

Unit 3

Chapter 8

Registers

Syllabus

Shift register types (SISO, SIPO, PISO & PIPO) & applications.

Chapter Contents

8.1 Introduction	8.7 Parallel In Serial Out Mode (PISO)
8.2 Data Formats	8.8 Parallel In Parallel Out (PIPO)
8.3 Classification of Registers	8.9 Bidirectional Shift Register
8.4 Buffer Registers	8.10 Applications of Shift Registers
8.5 Shift Registers	8.11 Sequence Detector
8.6 Serial In Parallel Out (SIPO)	



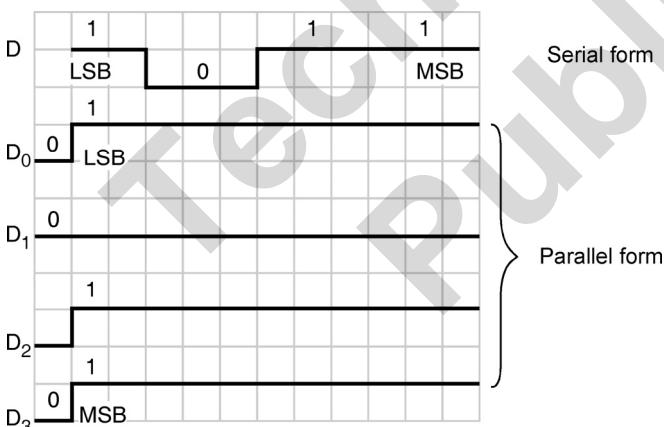
8.1 Introduction :

Definition :

- In chapter 6, we have stated various applications of flip-flops. One of them was "registers".
- Flip-flop is a 1 bit memory cell which can be used for storing the digital data.
- To increase the storage capacity in terms of number of bits, we have to use a group of flip-flops. Such a group of flip-flops is known as **a register**.
- Thus register is a group of flip-flops. The "n-bit" register will consist of "n" number of flip-flops and it is capable of storing an "n bit" word.

8.2 Data Formats :

- The data can be entered in **serial** (one bit at a time) manner or in the **parallel** form (all the bits at the same time) into a register.
- And the stored data can be retrieved in the serial or parallel form.
- A four bit digital data 1 0 1 1 is represented in the serial and parallel forms in Fig. 8.2.1.



(C-729(h)) Fig. 8.2.1 : Data representation in serial and parallel forms

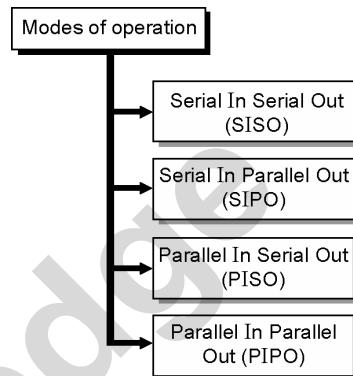
8.3 Classification of Registers :

SPPU : Dec. 04

University Questions

- Q. 1** Explain different modes of universal shift register with application of each. **(Dec. 04, 6 Marks)**

- Registers are classified on the basis of how the data bits are entered and taken out from a register. There are four possible modes as follows :



(C-729(i)) Fig. 8.3.1 : Modes of operation of registers

- We can design all these registers using discrete flip-flops such as SR, JK or D flip-flops.
- But the registers are also available in the IC form.
- Some of the registers available in 54/74 TTL series as listed in Table 8.3.1.

Table 8.3.1 : Shift registers available in 74/54 series

IC number	Description
7491, 7491A	8 bit serial in, serial out
7494	4 bit parallel in, serial out
7495	4 bit serial/parallel in, parallel out (shift right shift left)
7496	5 bit parallel in / parallel out, serial in / serial out.

8.4 Buffer Registers :

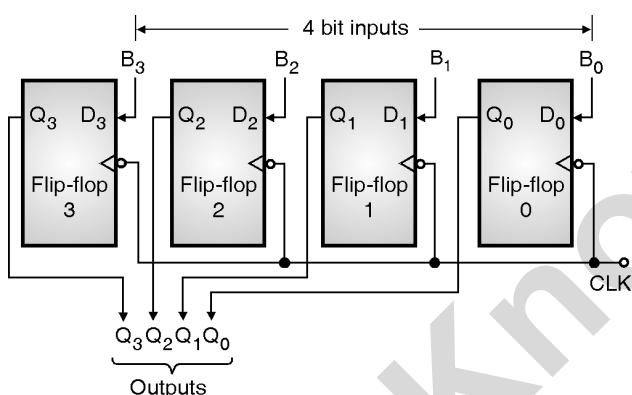
Logic diagram :

- The simplest type of register constructed using four D flip-flops is shown in Fig. 8.4.1.
- This is a 4 bit register, but we can construct an n-bit register by following the same principle.
- This register is also called as the **buffer register**.
- Each D-flip-flop is negative edge triggered and all the flip-flops are connected to a common clock signal. Hence all of them are triggered at the same instant of time, i.e. all the flip flops will change their state at the same instant of time.
- Buffer registers are used for temporary storage of digital words.

Operation :

- Let us assume that the word to be stored is $B_3 B_2 B_1 B_0 = 1010$.
- These bits are connected to the D inputs of the four D flip-flops as shown in Fig. 8.4.1.
- Then the clock pulse is applied.
- Corresponding to the first negative edge of the clock pulse, the outputs of all the D flip-flops will follow their respective inputs.

$$\therefore Q_3 Q_2 Q_1 Q_0 = B_3 B_2 B_1 B_0 = 1010$$

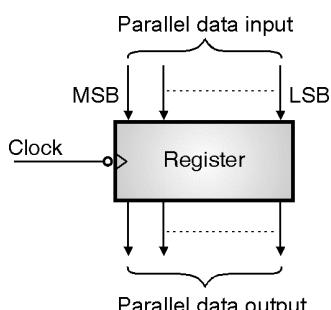


(C-729(j)) Fig. 8.4.1 : A four bit buffer register using D flip-flops

- Even if the inputs are now changed, the output remains latched to 1010 till the next negative edge of the clock arrives at the input.
- Thus the buffer register is capable of storing the digital data.

Schematic diagram :

- The schematic diagram of a buffer register is as shown in Fig. 8.4.2.



(C-730) Fig. 8.4.2 : Schematic diagram of buffer register

Conclusions :

- Some of the important conclusions from the discussion till now are as follows :
- There must be one-FF for each bit to be stored. Therefore to store a 4 bit number we need four flip-flops.
- Note that all the four input bits $B_3 B_2 B_1 B_0$ are loaded into the buffer register simultaneously, i.e. at the same instant of time.
- Therefore this way of applying the input and taking the output is called as Parallel Input Parallel Output (PIPO) and the mode of operation is called as **parallel shifting**.

8.5 Shift Registers :

Definition :

- The binary data in a register can be moved within the register from one flip-flop to the other or outside it with application of clock pulses.
- The registers that allow such data transfers are called as shift registers.
- Shift registers are used for data storage, data transfer and certain arithmetic and logic operations.

Modes of operation of a shift register :

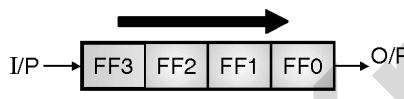
- The various modes in which a shift register can operate are as follows :
 - Serial Input Serial Output. (SISO).
 - Serial Input Parallel Output. (SIPO).
 - Parallel In Serial Out. (PISO).
 - Parallel In Parallel Out. (PIPO).
- These modes are explained in brief in Table 8.5.1.

Table 8.5.1 : Brief explanation of various modes of shift register

Sr. No.	Mode	Illustrative diagram	Comments
1.	Serial input serial output (serial shift right)	Refer Fig. A	Data bits shift from left to right by 1 position per clock cycle.



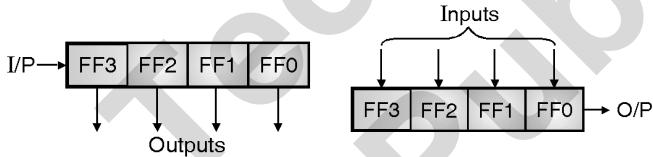
Sr. No.	Mode	Illustrative diagram	Comments
2.	Serial input serial output (serial shift left)	Refer Fig. B	Data bits shift from right to left by 1 position per clock.
3.	Serial input parallel output	Refer Fig. C	All output bits are made available simultaneously after 4-clock pulses.
4.	Parallel input serial output	Refer Fig. D	All inputs are loaded simultaneously but output bit by bit.
5.	Parallel input parallel output	Refer Fig. E	All inputs are loaded simultaneously and are available at the output simultaneously.



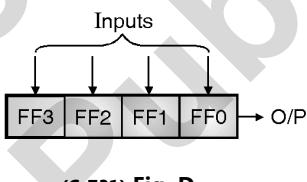
(C-731) Fig. A



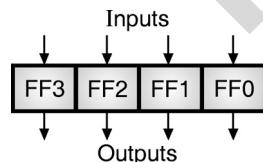
(C-731) Fig. B



(C-731) Fig. C



(C-731) Fig. D



(C-3167) Fig. E

8.5.1 Serial Input Serial Output (Shift Left Mode) :

SPPU : May 05, May 18, May 19

University Questions

Q. 1 Draw and explain the circuit diagram of 3-bit register with Serial Left Shift. (May 05, 4 Marks)

Q. 2 Draw and explain 4 bit SISO and SIPO shift register. Give applications of each.

(May 18, 6 Marks)

Q. 3 Draw and explain SISO and PIPO type of shift register. Give application of each.

(May 19, 6 Marks)

Principle :

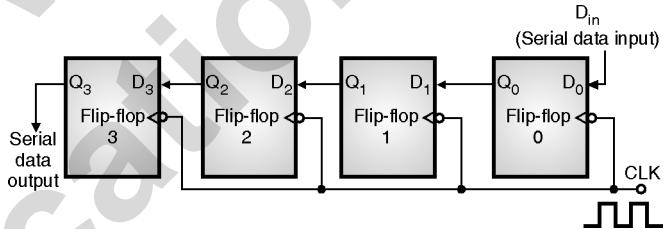
- Data bits shift from right to left by 1 position per clock cycle as shown in Fig. 8.5.1(a).



(C-731) Fig. 8.5.1(a) : Principle of Serial shift left register

Logic diagram :

- The serial input serial output type shift register with shift left mode is shown in Fig. 8.5.1(b).



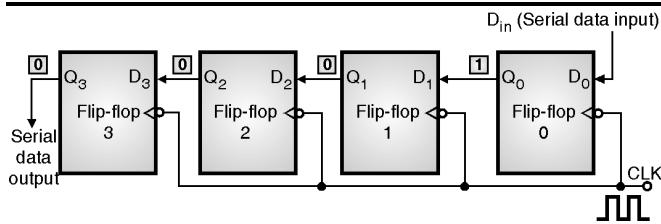
(C-732) Fig. 8.5.1(b) : Serial shift left register

- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1 1 1 1 into the register.
- When this is to be done, this number should be applied to "D_{in}" bit by bit with the MSB bit applied first.
- The D input of FF-0 i.e. D₀ is connected to serial data input (D_{in}). Output of FF-0 i.e. Q₀ is connected to the input of the next flip-flop i.e. D₁ and so on.

Operation :

- Before application of clock signal let $Q_3 Q_2 Q_1 Q_0 = 0 0 0 0$ and apply MSB bit of the number to be entered to D_{in}. So D_{in} = D₀ = 1.
- Apply the clock. On the first falling edge of clock, the FF-0 is set and the stored word in the register is

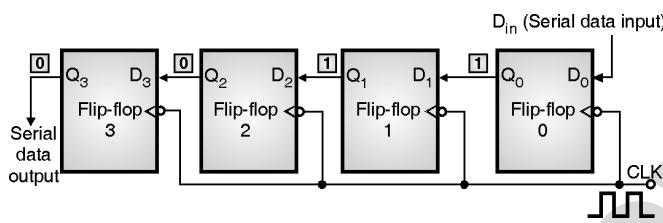
$$Q_3 Q_2 Q_1 Q_0 = 0 0 0 1$$



(C-733) Fig. 8.5.2 : Shift register status after first falling clock edge

- Apply the next bit to D_{in} . So $D_{in} = 1$.
- As soon as the next negative edge of the clock hits, FF-1 will set and the stored word changes to,

$$Q_3 \ Q_2 \ Q_1 \ Q_0 = 0 \ 0 \ 1 \ 1$$



(C-734) Fig. 8.5.3 : Shift register status after the second falling edge of clock

- Apply the next bit to be stored i.e. 1 to D_{in} .
- Apply the clock pulse. As soon as the third negative clock edge hits, FF-2 will be set and the output get modified to, $Q_3 \ Q_2 \ Q_1 \ Q_0 = 0 \ 1 \ 1 \ 1$.
- Similarly with $D_{in} = 1$, and with the fourth negative clock edge arriving, the stored word in the register is, $Q_3 \ Q_2 \ Q_1 \ Q_0 = 1 \ 1 \ 1 \ 1$.

(C-737) Table 8.5.2 : Summary of shift left operation

	CLK	Q_3	$Q_2 = D_3$	$Q_1 = D_2$	$Q_0 = D_1$	Serial input $D_{in} = D_0$
Initially		0	0	0	0	
1 st	↓	0	0	0	1	1
2 nd	↓	0	0	1	1	1
3 rd	↓	0	1	1	1	1
4 th	↓	1	1	1	1	1

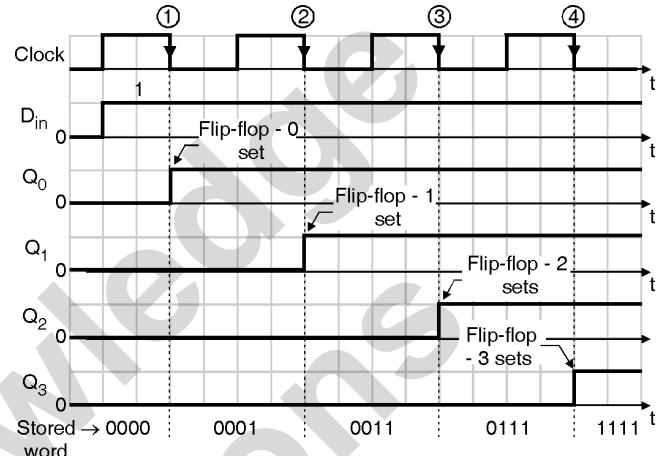
Direction of data travel ←

Waveforms for shift left operation :

- The waveforms for the shift left operation are shown in Fig. 8.5.4.

Important note :

Using the SISO mode, we needed 4-clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.



(C-738) Fig. 8.5.4 : Waveforms for the shift left operation

8.5.2 Serial In Serial Out (Shift Right Mode) :

SPPU : May 05

University Questions

- Q. 1 Draw and explain the circuit diagram of 3-bit register with Serial Right Shift. (May 05, 5 Marks)

Principle :

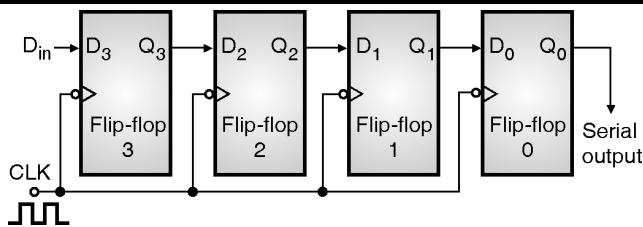
- Data bits shift from left to right by 1 position per clock cycle as shown in Fig. 8.5.5(a).



(C-731) Fig. 8.5.5(a) : Principle of Serial shift right register

Logic diagram :

- The serial input serial output type shift register with shift right mode is shown in Fig. 8.5.5(b).
- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1111 into the register.
- When this is to be done, this number should be applied to "D_{in}" bit by bit with the LSB bit applied first.

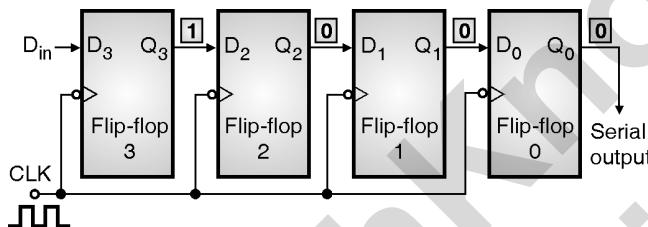


(C-739) Fig. 8.5.5(b) : Serial shift right register

- The D input of FF-3 i.e. D_3 is connected to serial data input (D_{in}). Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Operation :

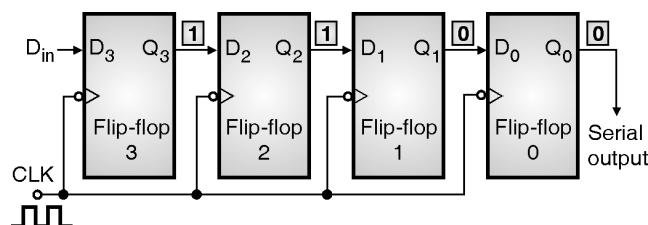
- Before application of clock signal let $Q_3 Q_2 Q_1 Q_0 = 0 0 0 0$ and apply LSB bit of the number to be entered to D_{in} . So $D_{in} = D_3 = 1$.
 - Apply the clock. On the first falling edge of clock, the FF-3 is set, and the stored word in the register is,
- $$Q_3 Q_2 Q_1 Q_0 = 1 0 0 0$$



(C-740) Fig. 8.5.6 : Shift register status after first falling clock edge

- The shift register after the application of the first clock pulse is as shown in Fig. 8.5.6.
 - Apply the next bit to D_{in} . So $D_{in} = 1$.
 - As soon as the next negative edge of the clock is applied, FF-2 will set and the stored word changes to,
- $$Q_3 Q_2 Q_1 Q_0 = 1 1 0 0$$

- This is as shown in Fig. 8.5.7.



(C-741) Fig. 8.5.7 : Shift register status after the second falling edge of clock

- Apply the next bit to be stored i.e. 1 to D_{in} .

– Apply the clock pulse. As soon as the third negative clock edge gets applied, FF-1 will be set and the output get modified to,

$$Q_3 Q_2 Q_1 Q_0 = 1 1 1 0$$

– Similarly with $D_{in} = 1$ and with the fourth negative clock edge arriving, the stored word in the register is

$$Q_3 Q_2 Q_1 Q_0 = 1 1 1 1$$

- Table 8.5.3 summarizes the shift right operation.

(C-744) Table 8.5.3 : Summary of shift right operation

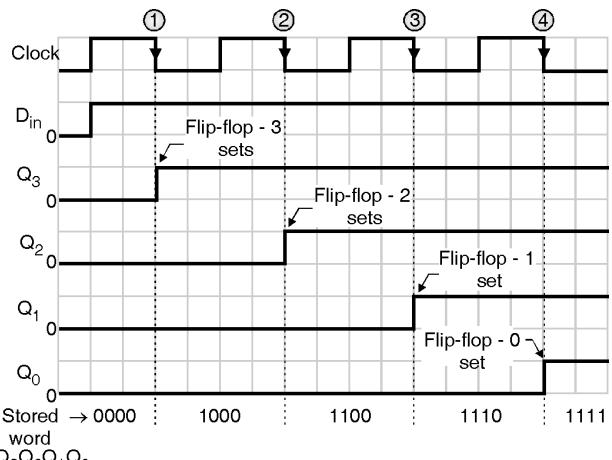
CLK	$D_{in} = D_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially		0	0	0	0
1 st	↓ 1	1 → 1	0	0	0
2 nd	↓ 1	1 → 1	1	0	0
3 rd	↓ 1	1 → 1	1	1	0
4 th	↓ 1	1 → 1	1	1	1

→ Direction of data travel

Waveforms for shift right operation :

- The waveforms for shift right operation are as shown in Fig. 8.5.8.

Important note : Using the SISO mode, we needed 4-clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.



(C-745) Fig. 8.5.8 : Waveforms for the shift right operation

8.5.3 Applications of Serial Operation :

SPPU : May 18, May 19

University Questions

- Q. 1** Draw and explain 4 bit SISO and SIPO shift register. Give applications of each.

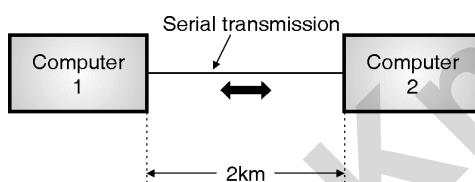
(May 18, 6 Marks)



Q. 2 Draw and explain SISO and PIPO type of shift register. Give application of each.

(May 19, 6 Marks)

- The transmission of data from one place to the other takes place in serial manner as shown in Fig. 8.5.9. The serial shifting of data transmits one bit at a time per clock cycle.
- It takes a longer time for serial transmission, because the time required to transmit one bit is equal to the time corresponding to one clock cycle.
- The parallel transmission is much faster as it can transmit 8 bits per clock cycle. But it needs more wires for connecting a transmitter to receiver.
- However for long distance communication where the distances are in kilometres, serial communication has an advantage that only one conductor is required to be used for the data transfer.



(C-746) Fig. 8.5.9 : Application of serial operation

8.6 Serial In Parallel Out (SIPO) :

SPPU : May 18

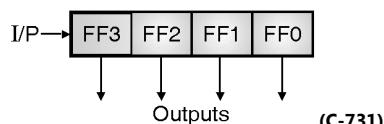
University Questions

Q. 1 Draw and explain 4 bit SISO and SIPO shift register. Give applications of each.

(May 18, 6 Marks)

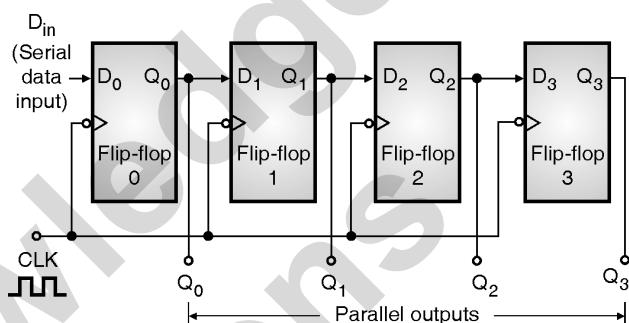
Principle :

- In this operation the data is entered serially and taken out in parallel as shown in the following figure.



- In this operation the data is entered serially and taken out in parallel.
- That means first the data is loaded bit by bit. The outputs are disabled as long as the loading is taking place.

- As soon as the loading is complete, and all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines simultaneously.
- Number of clock cycles required to load a four bit word is 4. Hence the speed of operation of SIPO mode is same as that of SISO mode.



(C-747) Fig. 8.6.1 : Serial input parallel output mode

8.7 Parallel In Serial Out Mode (PISO) :

SPPU : Dec. 08, May 10, Dec. 17

University Questions

Q. 1 Draw and explain the circuit diagram of 3 bit register with the following facility :

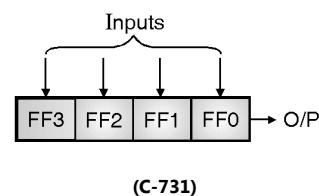
1. Parallel in serial out
2. Reset

(Dec. 08, 6 Marks)

Q. 2 Explain with a neat diagram working of parallel in serial out 4-bit shift register. Draw necessary timing diagram. (May 10, Dec. 17, 6 Marks)

Principle :

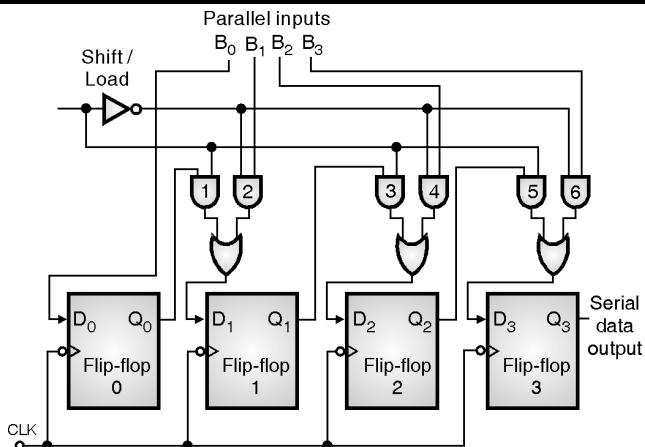
- In this mode, the bits are entered in parallel i.e. simultaneously into a shift register as shown below.



(C-731)

Logic diagram :

- The circuit shown in Fig. 8.7.1 is a four bit parallel input serial output register.



(C-748) Fig. 8.7.1 : Parallel in serial out shift register

- Output of previous FF is connected to the input of the next one via a combinational circuit.
- The binary input word B_0, B_1, B_2, B_3 is applied through the same combinational circuit.
- There are two modes in which this circuit can work namely shift mode or load mode.

Load mode :

- In order to load the word $B_3 B_2 B_1 B_0$ into the shift register we have to select the load mode by setting shift/ load input to 0.
- When the shift / load line is low (0), the AND gates 2, 4 and 6 become active.
- They will pass B_1, B_2 and B_3 bits to the inputs D_1, D_2 and D_3 of the corresponding flip-flops. D_0 is directly connected to B_0 .
- On the low going edge of clock, the binary inputs $B_0 B_1 B_2 B_3$ will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode :

- In order to operate the shift register in the shift mode we have to select the shift mode by applying a logic 1 to the shift/ load input.
- When the shift / load line is high (1), the AND gates 2, 4, 6 become inactive. Hence the parallel loading of the data becomes impossible.
- But the AND gates 1, 3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses becomes possible. D_0 acts as

the data input terminal and at Q_3 we get the serial data output.

- Thus the parallel in serial out operation takes place.

8.8 Parallel In Parallel Out (PIPO) :

SPPU : May 19

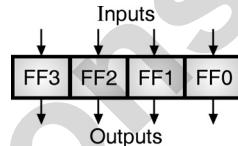
University Questions

- Q. 1** Draw and explain SISO and PIPO type of shift register. Give application of each.

(May 2019, 6 Marks)

Principle :

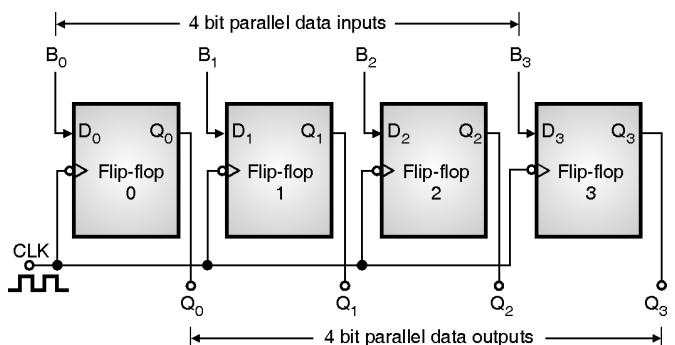
- All inputs are loaded simultaneously and are available at the output simultaneously as shown below.



(C-3167) Fig. 8.8.1(a) : Principle of PIPO register

Logic diagram :

- Fig. 8.8.1(b) demonstrates the parallel in parallel out mode of operation.
- The 4 bit binary input $B_0 B_1 B_2 B_3$ is applied to the data inputs D_0, D_1, D_2 and D_3 respectively of the four flip-flops.
- As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously.



(C-749) Fig. 8.8.1(b) : Parallel in parallel out shift register

- The loaded bits will appear simultaneously to the output side. Only one clock pulse is essential to load all the bits. Therefore PIPO mode is the fastest mode of operation.

**Applications of PIPO shift register :**

PIPO shift registers can be used as :

1. A temporary storage device.
2. As a delay element.

8.9 Bidirectional Shift Register :

SPPU : May 06, Dec. 11, Dec. 13

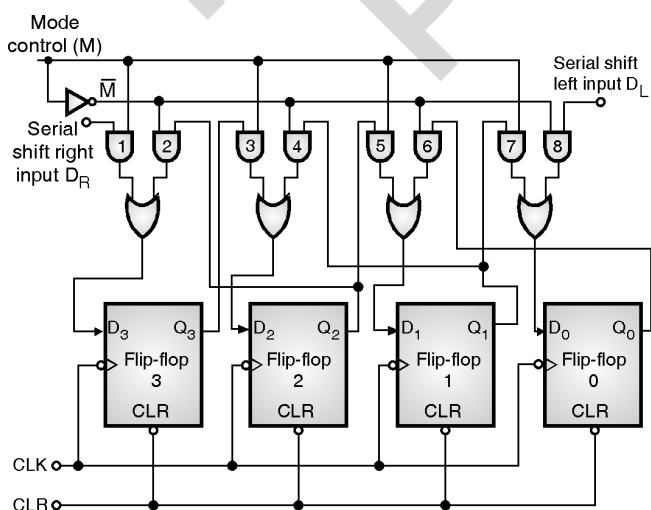
University Questions

Q. 1 Draw and explain 4-bit shift register having shift left and shift right facilities. Explain any one application of such register. **(May 06, 6 Marks)**

Q. 2 Draw and explain 4 bit bidirectional shift register. **(Dec. 11, 8 Marks)**

Q. 3 Explain with a neat diagram working of 3 bit bidirectional shift register. **(Dec. 13, 6 Marks)**

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2.
- On the other hand if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2. This is illustrated below.
- Let a four bit number $Q_3 Q_2 Q_1 Q_0 = 0010 = (2)_{10}$ is existing in a shift register.
- Now with a 0 applied at the input, if we shift these contents by one position to left then we get $Q_3 Q_2 Q_1 Q_0 = 0100 = (4)_{10}$.
- Thus the shift left is equivalent to multiplying by 2. Now shift it right by one position to get $Q_3 Q_2 Q_1 Q_0 = 0001 = (1)_{10}$. Thus shifting right is equivalent to dividing by 2.



(C-750) Fig. 8.9.1 : A 4-bit bi-directional shift register

- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction as and when we want.
- Such a register is called as a bi-directional register. A four bit bi-directional shift register is shown in Fig. 8.9.1.
- There are two serial inputs namely the serial right shift data input D_R and the serial left shift data input D_L alongwith a Mode control input (M).

Operation :**With M = 1 : Shift right operation**

- If $M = 1$, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled.
- Hence the data at D_R (shift right input) is shifted right bit by bit from FF-3 to FF-0 on the application of clock pulses.
- Thus with $M = 1$ we get the serial right shift operation.

With M = 0 : Shift left operation

- When the mode control M is connected to "0" then the AND gates 2, 4, 6 and 8 are enabled while 1, 3, 5 and 7 are disabled.
- Therefore the data at D_L (shift left input) is shifted left bit by bit from FF-0 to FF-3 on application of the clock pulses.
- Thus with $M = 0$ we get the serial left shift operation.
- Note that M should be changed only when $CLK = 0$, otherwise the data stored in the register may get changed in an undesirable manner.

8.9.1 A 3-bit Bidirectional Register using the JK Flip Flops :

SPPU : Dec. 05

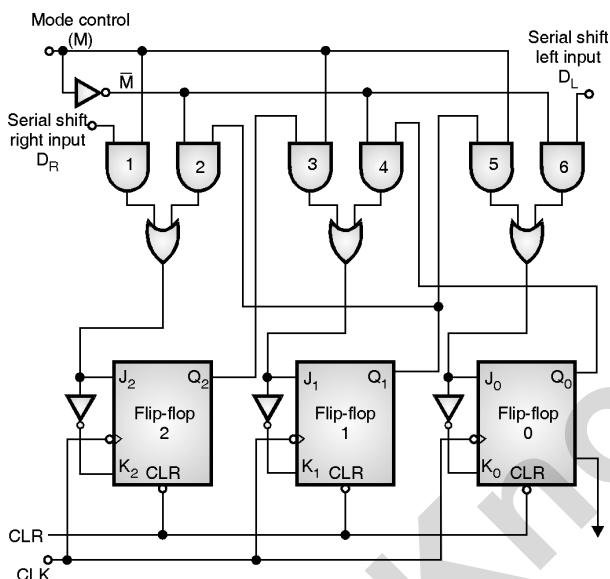
University Questions

Q. 1 Draw the circuit diagram of 3-bit bi-directional shift register. **(Dec. 05, 6 Marks)**

Logic diagram :

- A 3-bit bi-directional shift register using JK flip flops is shown in Fig. 8.9.2.

- This circuit is same as the 4 bit bi-directional shift register discussed in the previous section.
- The only modification is the inclusion of an inverter between the J and K inputs of each flip flops.
- With $M = 1$ the shift right operation will take place as discussed in the previous section and with $M = 0$ the shift left operation is going to take place.



(C-751) Fig. 8.9.2 : A 3 bit bi-directional shift register using JK flip flops

8.10 Applications of Shift Registers :

SPPU : May 06, May 10, May 18

University Questions

- Q. 1** Draw and explain 4-bit shift register having shift left and shift right facilities. Explain any one application of 4-bit shift register.
(May 06, 6 Marks)
- Q. 2** Give any four applications of shift registers.
(May 10, 4 Marks)
- Q. 3** Draw and explain 4 bit SISO and SIPO shift register. Give applications of each.
(May 18, 6 Marks)

- Some of the common applications of a shift register are :
 1. For temporary data storage.
 2. For multiplication and division.
 3. As a delay line.

- 4. Serial to parallel converter.
- 5. Parallel to serial converter.
- 6. Ring counter.
- 7. Twisted ring counter or Johnson counter.
- 8. Serial data transmission.
- We have already seen the use of shift register for temporary storage of data and for multiplication or division.

8.10.1 Serial to Parallel Converter :

SPPU : Dec. 08

University Questions

- Q. 1** Explain how shift registers are used as serial to parallel converters ?
(Dec. 08, 3 Marks)

- In many applications, the data in serial form needs to be converted in the parallel form.
- The Serial Input Parallel Output Mode (SIPO) of the shift register is used for such applications.

8.10.2 Parallel to Serial Converter :

SPPU : Dec. 08

University Questions

- Q. 1** Explain how shift registers are used as parallel to serial converter ?
(Dec. 08, 3 Marks)

- The data communication between two computers takes place in the serial transmission form.
- But internally the data processing takes place in the parallel form.
- Hence we need to use a parallel to serial converter to convert the internal parallel data into a serial data suitable for transmission.
- We can use the parallel to serial mode of the shift register for this operation.

8.10.3 Ring Counter :

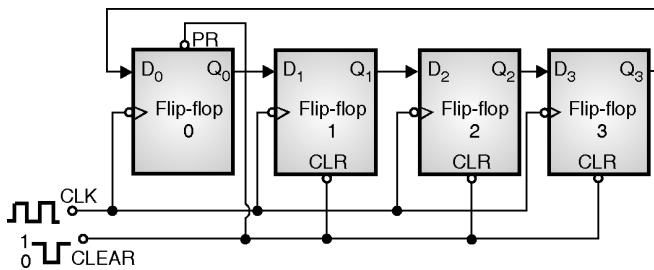
SPPU : Dec. 08, May 17

University Questions

- Q. 1** Write short note on ring counter.
(Dec. 08, 6 Marks)
- Q. 2** Explain with a neat diagram ring counter.
(May 17, 6 Marks)

Logic diagram :

- Fig. 8.10.1 shows a typical application of shift registers called Ring Counter.
- The connections reveal that they are similar to the connections for shift right operation, except for one change.



(C-871) Fig. 8.10.1 : A four bit ring counter

- Output of FF-3 is connected to data input D_0 of FF-0. Ring counter is a special type of shift register.

Operation :

- Initially a low clear (CLR) pulse is applied to all the flip-flops.
- Hence FF-3, FF-2 and FF-1 will reset but FF-0 will be preset. So the outputs of the shift register are :

$$Q_3 Q_2 Q_1 Q_0 = 0 0 0 1.$$

- Now the clear terminal is made inactive by applying a high level to it.
- The clock signal is then applied to all the flip-flops simultaneously. Note that all the flip-flops are negative edge triggered.

On the first negative going CLK edge :

- As soon as the first falling edge of the clock hits, only FF-1 will be set because $Q_0 = D_1 = 1$.
- The FF-0 will reset because $D_0 = Q_3 = 0$ and there is no change in the status of FF-2 and FF-3.
- Hence after the first clock pulse the outputs are

$$Q_3 Q_2 Q_1 Q_0 = 0 0 1 0.$$

On the second falling edge of clock :

- At the second falling edge of the clock, only FF-2 will be set because $D_2 = Q_1 = 1$.
- FF-1 will reset since $D_1 = Q_0 = 0$. There is no change in status of FF-3 and FF-0.
- So after the second clock pulse the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 0 1 0 0.$$

- Similarly after the third clock pulse the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 1 0 0 0.$$

- And after the fourth one the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 0 0 0 1.$$

- These are the outputs from where we started. Hence the operation repeats from this point onwards.

Number of output states :

- The number of output states for a ring counter will always be equal to the number of flip-flops. So for a 4-bit ring counter the number of states is equal to 4.
- The operation of a four bit ring counter is summarized in Table 8.10.1.

(C-871(a)) Table 8.10.1 : Summary of operation of a ring counter

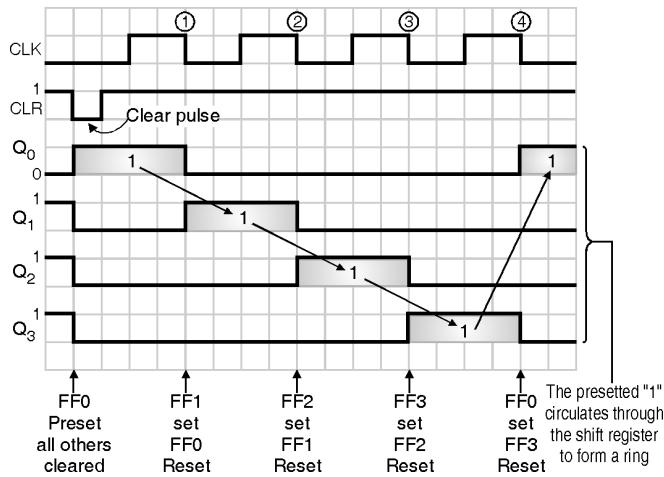
CLR	CLK	Q_0	Q_1	Q_2	Q_3	
1	X	1	0	0	0	
1	↓	0	1	0	0	
1	↓	0	0	1	0	
1	↓	0	0	0	1	
1	↓	1	0	0	0	

← FF-0 Preset, others cleared

The presetted 1 follows a circular path to form a ring

Waveforms for the ring counter :

- The waveforms for the 4-bit ring counter are as shown in Fig. 8.10.2.
- These waveforms clearly show that the presetted "1" shifts one bit per clock cycle and forms a ring. Hence the name ring counter.



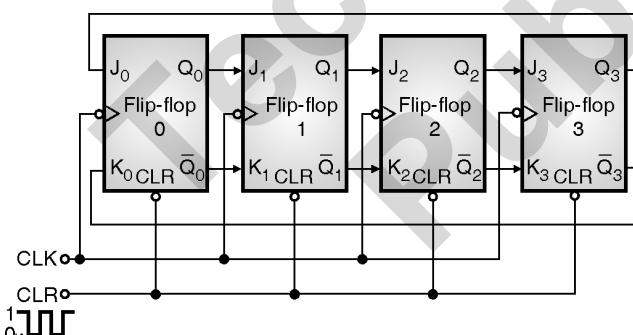
(C-1360) Fig. 8.10.2 : Waveforms of a four bit ring counter

Applications of ring counter :

- Ring counters are used in those applications in which several operations are to be controlled in a sequential manner.
- For example in resistance welding the operations called squeeze, hold, weld and off are to be performed sequentially.
- We can use a ring counter to initiate these operations.

Ring counter using JK Flip Flop :

- A ring counter can also be constructed using the JK flip flops, as shown in Fig. 8.10.3.



(C-1361) Fig. 8.10.3 : Ring counter using JK flip flops

8.10.4 Johnson's Counter (Twisted / Switch Tail Ring Counter) :

SPPU : Dec. 08, May 10, May 11, May 14, Dec. 16

University Questions

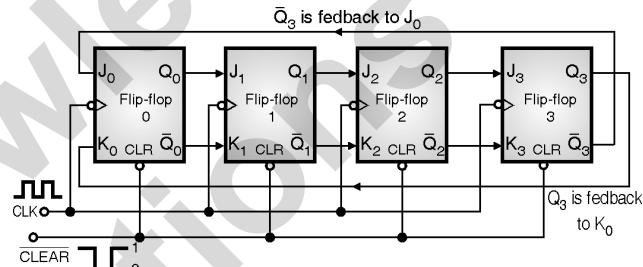
Q. 1 Write short notes on Johnson counter.
(Dec. 08, May 10, 6 Marks)

Q. 2 Explain twisted ring counter in brief.
(May 11, 4 Marks)

Q. 3 Draw and explain Johnson counter with initial state "1110", from initial state. Explain all possible states.
(May 14, Dec. 16, 6 Marks)

Logic diagram :

- In the ring counter the outputs of FF-3 were connected directly to the inputs of FF-0 i.e. Q₃ to J₀ and \bar{Q}_3 to K₀.
- Instead if the outputs are cross coupled to the inputs i.e. if Q₃ is connected to K₀ and \bar{Q}_3 is connected to J₀ then the circuit is called as twisted ring counter or Johnson's counter.
- The Johnson's counter is shown in Fig. 8.10.4.



(C-1362) Fig. 8.10.4 : Twisted ring counter or Johnson counter

- All the flip-flops are negative edge triggered, and clock pulses are applied to all of them simultaneously.
- The clear inputs of all the flip-flops are connected together and connected to an external clear signal. Note that all these clear inputs are active low inputs.

Operation :

- Initially a short negative going pulse is applied to the clear input of all the flip-flops. This will reset all the flip-flops. Hence initially the outputs are, Q₃ Q₂ Q₁ Q₀ = 0 0 0 0.
- But $\bar{Q}_3 = 1$ and since it is coupled to J₀ it is also equal to 1.

$$\therefore J_0 = 1 \text{ and } K_0 = 0 \dots \text{Initially}$$

On the first falling edge of clock pulse :

- As soon as the first negative edge of clock arrives, FF-0 will be set. Hence Q₀ will become 1.



- But there is no change in the status of any other flip-flop.
- Hence after the first negative going edge of the clock the flip-flop outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 0001$$

On the second negative going clock edge :

- Before the second negative going clock edge, $Q_3 = 0$ and $\bar{Q}_3 = 1$. Hence $J_0 = 1$ and $K_0 = 1$. Also $Q_0 = 1$. Hence $J_1 = 1$.
- Hence as soon as the second falling clock edge arrives, FF-0 continues to be in the set mode and FF-1 will now set. Hence Q_1 will become 1 and $\bar{Q}_1 = 0$.
- There is no change in the status of any other flip-flop.
- Hence after the second clock edge the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 0011.$$

- Similarly after the third clock pulse, the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 0111.$$

- And after the fourth clock pulse, the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 1111.$$

Note that now $\bar{Q}_3 = 0$ i.e. $J_0 = 0$ and $K_0 = 1$.

- Hence as soon as the fifth negative going clock pulse strikes, FF-0 will reset.
- But the outputs of the other flip-flops will remain unchanged. So after the fifth clock pulse, the outputs are,

$$Q_3 Q_2 Q_1 Q_0 = 1110 \dots \text{after the } 5^{\text{th}} \text{ clock pulse}$$

- This operation will continue till we reach the all zero output state. (i.e. $Q_3 Q_2 Q_1 Q_0 = 0000$).
- The operation of Johnson's counter is summarised in Table 8.10.2.

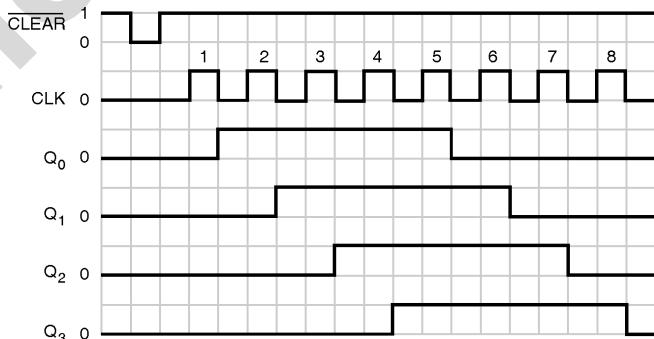
(C-6295) Table 8.10.2 : Summary of operation of Johnson's counter

CLEAR	CLK	Q_3	Q_2	Q_1	Q_0	State number	Decimal equivalent
	Initially	0	0	0	0	1	0
1	↓	0	0	0	1	2	1
1	↓	0	0	1	1	3	3
1	↓	0	1	1	1	4	7
1	↓	1	1	1	1	5	15
1	↓	1	1	1	0	6	14
1	↓	1	1	0	0	7	12
1	↓	1	0	0	0	8	8
1	↓	0	0	0	0	1	0

- Note that there are 8 distinct states of output.
- In general we can say that the number of states of a Johnson's counter is twice the number of flip-flops used.
- Therefore for a 4-flip-flop Johnson's counter, there are 8-distinct output states.

Waveforms for Johnson's counter :

- The waveforms for a 4-bit Johnson's counter are shown in Fig. 8.10.5.



(C-875) Fig. 8.10.5 : Waveforms of Johnson counter

- Ex. 8.10.1 :** Explain ring counter design having initial state 01011 from initial state explain all possible states in the ring. **Dec. 10, 10 Marks**

Soln. :

- Since there are 5 bits in the given initial state, we have to use 5 flip flops as shown in Fig. P. 8.10.1.
- When we apply a low going clear (CLR) pulse, then flip flops 0, 1, B and 3 will be preset to 1 output while flip flops 2 and 4 are reset to 0 output.



- $\therefore Q_4 Q_3 Q_2 Q_1 Q_0 = 01011 \dots$ Initially
 – The remaining states are as shown in Table P. 8.10.1.

(C-3563) Table P. 8.10.1 : Ring counter states

CLR	CLK	Q_0	Q_1	Q_2	Q_3	Q_4
X	X	1	1	0	1	0
1	↓	0	1	1	0	1
1	↓	1	0	1	1	0
1	↓	0	1	0	1	1
1	↓	1	0	1	0	1
1	↓	1	1	0	1	0

FFs 0, 1 and 3 are preset

Ex. 8.10.2 : Explain the ring counter design for the initial condition 10110. **May 11, 4 Marks**

Soln. :

Similar to the previous example.

Ex. 8.10.3 : Explain the Johnson's counter design for initial state 0110. From initial state explain and draw all possible states. **Dec. 09, 8 Marks**

Soln. :

- The required Johnson's counter is shown in Fig. P. 8.10.3.
- Counters 0 and 3 are reset to 0 while counters 1 and 2 and preset to 1 initially.
 $\therefore Q_3 Q_2 Q_1 Q_0 = 0110 \dots$ initially
- The other possible states of this Johnson's counter are listed in Table P. 8.10.3.

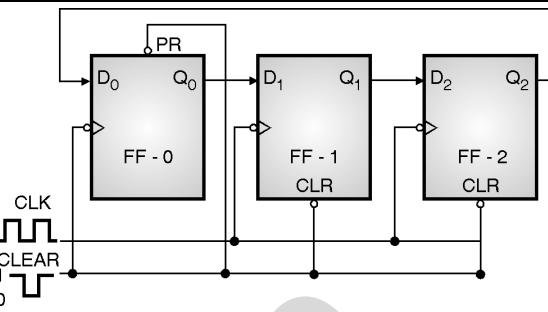
(C-3565) Table P. 8.10.3 : All possible states of a Johnson's counter

Clear	CLK	Q_3	Q_2	Q_1	Q_0
X	X	0	1	1	0
1	↓	1	1	0	1
1	↓	1	0	1	0
1	↓	0	1	0	0
1	↓	1	0	0	1

Ex. 8.10.4 : Draw and explain 3-bit ring counter. **Dec. 15, 6 Marks**

Soln. :

- The 3 bit ring counter is as shown in Fig. P. 8.10.4(a).



(C-3277) Fig. P. 8.10.4(a) : A three bit ring counter

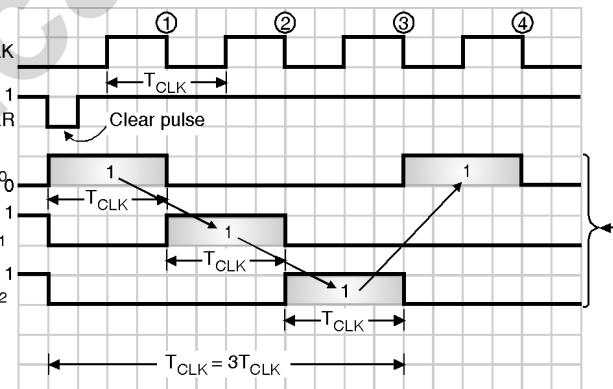
- The operation of a three bit ring counter is summarized in Table P. 8.10.4.

(C-3278) Table P. 8.10.4 : Summary of operation of a ring counter

CLR	CLK	Q_0	Q_1	Q_2
X	X	1	0	0
1	↓	0	1	0
1	↓	0	0	1
1	↓	1	0	0

Waveforms :

- The waveforms of a 3-bit ring counter are as shown in Fig. P. 8.10.4(b).



The presetted "1" circulates through the shift register to form a ring

(C-3279) Fig. P. 8.10.4(b) : Waveforms of a 3-bit ring counter

Compare the time periods of waveforms at Q_2 and CLK.

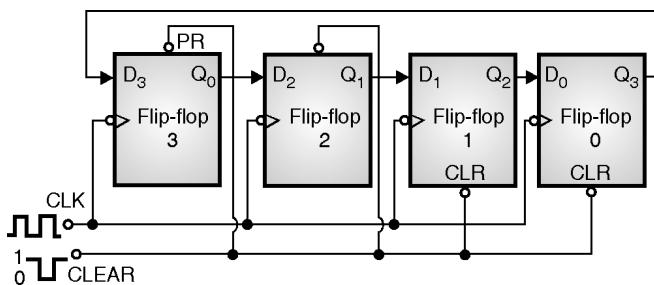
$$T_{\text{out}} = 3 T_{\text{CLK}} \quad \therefore \quad f_{\text{out}} = \frac{1}{3} f_{\text{CLK}} \quad \dots \text{Proved.}$$

Ex. 8.10.5 : Draw and explain 4 bit Ring counter. Write the Truth Table for same showing all possible states if initial state is 1100. **May 18, 6 Marks**

Soln. :

- Refer section 8.10.3 for 4-bit ring counter.

- Since there are 4 bits in the given initial state, we have to use 4 flip flops as shown in Fig. P. 8.10.5.
- When we apply a low going clear (CLR) pulse, then flip-flops 2 and 3 will be preset to 1 output while flip-flops 0 and 1 and are reset to 0 output.
 $\therefore Q_3 Q_2 Q_1 Q_0 = 1100$...initially
- The remaining states are as shown in Table P. 8.10.5.



(C-7148) Fig. P. 8.10.5 : A four bit ring counter

(C-7149) Table P. 8.10.5 : Ring counter states

CLR	CLK	Q_0	Q_1	Q_2	Q_3
1	X	0	0	1	1
1	↓	1	0	0	1
1	↓	1	1	0	0
1	↓	0	1	1	0
1	↓	0	0	1	1

Ex. 8.10.6 : Draw 3-bit Ring and Twisted ring counter. Draw state diagram for 3-bit Ring and Twisted ring counter, assuming initial state as 001.

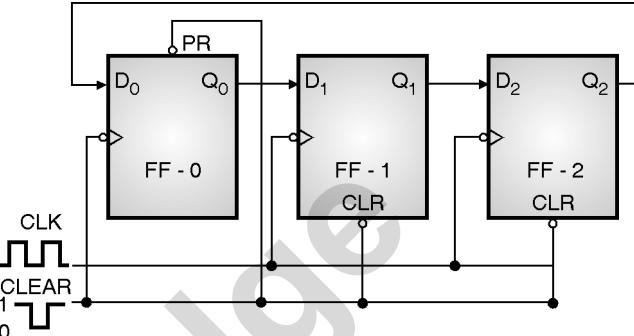
Dec. 19, 7 Marks

Soln. :

- Refer Sections 8.10.3 and 8.10.4 for Ring and Twisted ring counter.

3-bit Ring Counter:

The 3 bit ring counter is as shown in Fig. P. 8.10.6(a).



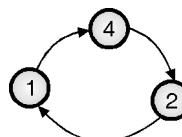
(C-3277) Fig. P. 8.10.6(a) : A three bit ring counter

- The operation of a three bit ring counter is summarized in Table P. 8.10.6(a).

(C-3278) Table P. 8.10.6(a) : Summary of operation of a ring counter

CLR	CLK	Q_0	Q_1	Q_2
1	↓	1	0	0
1	↓	0	1	0
1	↓	0	0	1
1	↓	1	0	0

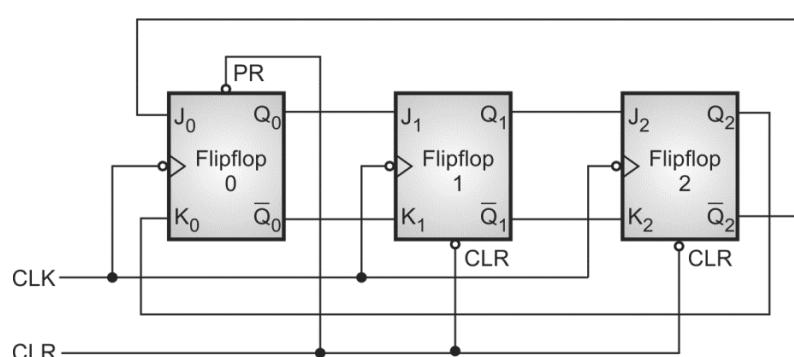
- Fig. P. 8.10.6(b) shows the state diagram of a 3 bit ring counter.



(C-1885) Fig. P. 8.10.6(b)

3-bit Twisted Ring Counter :

- The required Johnson's / Twisted Ring counter is shown in Fig. P. 8.10.6(c).



(C-8328) Fig. P. 8.10.6(c) : Required Johnson's counter

- Counters 1 and 2 are reset to 0 while counter 0 preset to 1 initially.
 $\therefore Q_2 Q_1 Q_0 = 001$...initially
- The other possible states of this Johnson's counter are listed in Table P. 8.10.6(b).

Table P. 8.10.6(b) : All possible states of a Johnson's counter

Clear	CLK	Q_0	Q_1	Q_2
1	X	1	0	0
1	↓	0	0	0
1	↓	0	0	1
1	↓	0	1	1
1	↓	1	1	1
1	↓	1	1	0
1	↓	1	0	0

8.10.5 Sequence Generator :

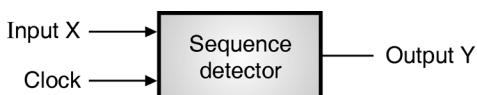
Definition :

- A sequence generator is a sequential circuit which generates a desired sequence at its output. The output sequence is in synchronization with the clock input.
- It is possible to design a sequence generator using counters or using the shift registers.

8.11 Sequence Detector :

Definition :

- **A sequence detector** is a synchronous FSM which is designed to detect a specific sequence of bits arriving at its input.
- The detector will produce a logic "1" at its output (Y) as soon as it detects the specified sequence of bits. This concept is illustrated in Fig. 8.11.1(b).
- As shown in Fig. 8.11.1(b), in the long data string at the input (X), the desired input sequence may sometimes get overlapped.
- Fig. 8.11.1(a) shows the block diagram of a sequence detector.



(C-691) **Fig. 8.11.1(a) : Block schematic of a sequence detector**

8.11.1 Pseudo Random Binary Sequence (PRBS) Generator :

- Another important application of a shift register is the pseudo random generator. It is used for generating the random sequences.

- The PRBS generator consists of a number of flip-flops and a combinational circuit which provides a suitable feedback.

Applications of PRBS generator :

- Since the sequence produced is random, the PRBS generator is also called as a Pseudo Noise (PN) generator and the generated signal is called noise.
- This noise can be used to test the noise immunity of the system under test.
- PRBS generator is an important part of the data encryption system. Such a system is required to protect the data from data hackers.

Review Questions

- Q. 1 What is the function of a shift register ? Give its applications.
- Q. 2 State the types of shift registers.
- Q. 3 With a neat diagram explain the operation of 4-bit left shift register. Give its truth table and timing diagram.
- Q. 4 With a neat diagram explain the operation of 4-bit SISO (Serial-In-Serial-Out) register. Draw the timing diagram and give its truth table.
- Q. 5 With a neat diagram explain the operation of 4-bit Serial-In-Parallel Out (SIPO) register. Give the truth table and timing diagram.
- Q. 6 With a neat diagram explain the operation of 4-bit Parallel-In-Serial-Out (PISO) register. Give the truth table and timing waveforms.
- Q. 7 What is meant by "Universal shift register" ?
- Q. 8 What do you understand by a bi-directional shift register ? Explain its operation.
- Q. 9 List any one shift register IC. Sketch its schematic diagram and pin configuration. Give its specifications.
- Q. 10 Draw circuit diagram of 3 bit SIPO shift register. Use D flip flops. Explain its working.
- Q. 11 Define bi-directional shift register. Draw 3 bit bi-directional shift register using D flip flop.

□□□

Unit 4

Chapter 9

Computer Organization & Processor

Syllabus

Computer organization and computer architecture, Organization, Functions and types of computer units - CPU (Typical organization, Functions, Types), Memory (Types and their uses in computer), IO (Types and functions) and system bus (Address, data and control, Typical control lines, Multiple-Bus Hierarchies); Von Neumann and Harvard architecture; Instruction cycle.

Processor : Single bus organization of CPU; ALU (ALU signals, functions and types); Register (Types and functions of user visible, control and status registers such as general purpose, address registers, data registers, flags, PC, MAR, MBR, IR) & control unit (Control signals and typical organization of hard wired and microprogrammed CU). Micro Operations (Fetch, Indirect, Execute, Interrupt) and control signals for these micro operations.

Case Study : 8086 processor, PCI bus.

Chapter Contents

9.1 Introduction	9.5 CPU Architecture and Register Organization
9.2 Basic Organization of Computer and Block Level Description of Functional Units	9.6 Instruction, Micro-instructions and Micro-operations : Interpretation and Sequencing
9.3 Von Neumann and Harvard Architecture	9.7 Control Unit : Hardwired Control Unit Design Methods
9.4 Basic Instruction Cycle	9.8 Control Unit : Soft Wired (Micro programmed) Control Unit Design Methods

9.1 Introduction :

- There are various people involved in making of a computer.
- The chip designer who designs and manufactures the chip, the system designer who designs the system using the manufactured chip or microprocessor and the programmer who makes software using the system.
- Those attributes of a computer that are necessary to be known to a system designer or a programmer are called as the architectural features of the computer.
- Hence the people manufacturing the chip have to reveal certain things about the processor to the system designer and the programmer using the datasheets for their processor chips.
- Those attribute of a computer or moreover the processor, that are just used for the designing purposes of the processor and are not revealed are called as the organizational features of the processor.

Sr. No.	Computer architecture	Computer organization
1.	It refers to those attributes of a system visible to the programmer.	It refers to the implementation of these features and is mostly not known to the user.
2.	Instruction set, number of bits used for data representation, addressing techniques etc. form the part of computer architecture.	Control signals, interfaces, memory technology etc. form the part of the computer organization.
3.	For example, is there a multiply instruction?	For example, is there a dedicated hardware multiply unit or it is done by repeated addition?
4.	All INTEL 80x86 microprocessors share the same basic architecture.	All INTEL 80x86 microprocessors differ in their organization.

9.2 Basic Organization of Computer and Block Level Description of Functional Units :

9.2.1 Structural Components of a Computer :

- It is the way in which components related to each other. Fig. 9.2.1 shows the structure of a digital computer.
- It is made up of three main components namely the Central Processing Unit (CPU) or the processor, the memory to store the programs and data, and the Input / Output (I/O) devices. The functions of these are explained below :

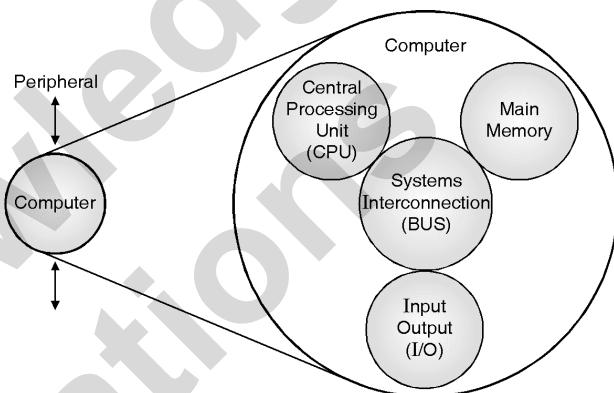


Fig. 9.2.1 : Structure of a computer

1. The components of a computer are central processing unit, Input and Output (I/O) devices and memory.
2. The three units are connected with the help of system interconnection i.e. buses.
3. Memory is used to store code (programs) and data. It can be various kinds of like semiconductor memory using ICs, magnetic memory or optical memory etc.
4. I/O devices are used to accept a input or give an output by the CPU. There are various input devices like keyboard, mouse, scanner; and various output devices like CRT, printer etc.
5. The CPU is further divided into three units as shown in the Fig. 9.2.2.

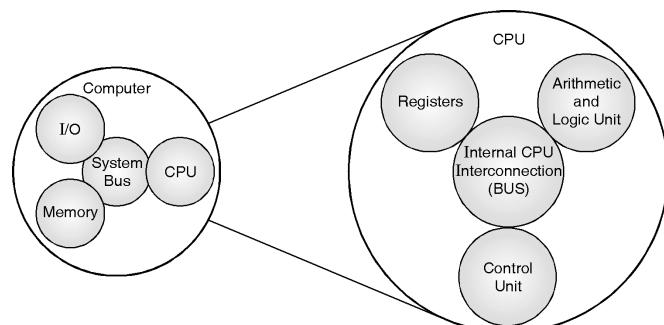


Fig. 9.2.2 : Structure of a CPU

6. The components of the Central Processing Unit (CPU) are Arithmetic and Logic Unit (ALU), Control Unit(CU) and CPU Registers, which are also connected with the internal buses.
7. ALU is used to perform arithmetic operations like addition, subtraction etc. and logical operations such as AND, OR etc.
8. CPU Registers are used to store the data temporarily in the CPU to save memory access time.
9. The CU is further divided in three parts as shown in Fig. 9.2.3.

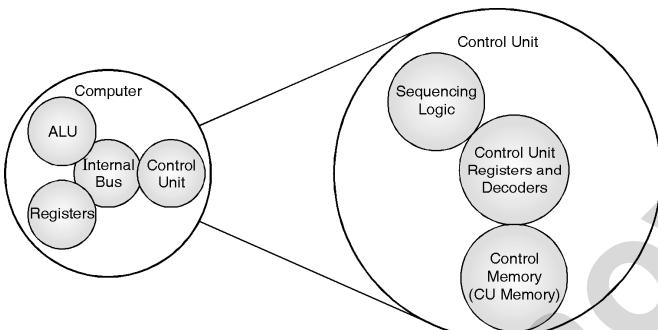


Fig. 9.2.3 : Structure of Control Unit

10. Control Unit comprises of control memory, control unit register and sequencing logic.
11. The control memory stores the microinstructions loads it into the control unit register and the sequencing logic gives these signals in a proper sequence to execute a instruction.

9.2.2 Functional View of a Computer :

1. It is the operation of the individual component as the part of the structure.
2. The functions of a computer are data processing, data storage, data movement and control.
3. There can be various paths followed by the data as shown in the Fig. 9.2.4. They can be: the data may be taken into the processor from the input device, processed and then the result may be given at the output device; the data may be taken into the processor from the input device, processed and the result stored in memory; the data may be taken from memory, processed and the result given to output device; the data taken from an input device and given to an output device etc.

4. In Fig. 9.2.4, we can see that the computer is divided into three main components namely data storage facility, data movement facility and the data processing facility. These are the different functions can perform a computer.

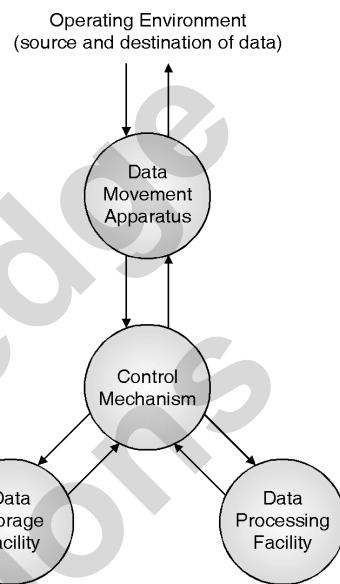


Fig. 9.2.4 : Functional view of a Computer

9.3 Von Neumann and Harvard Architecture :

- There are two ways of memory interfacing architectures for a processor depending on the processor design.
- The first one is called Von Neumann architecture and later Harvard architecture.

9.3.1 Von Neumann Architecture :

- Fig. 9.3.1 shows the connection for Von Neumann architecture of computer.

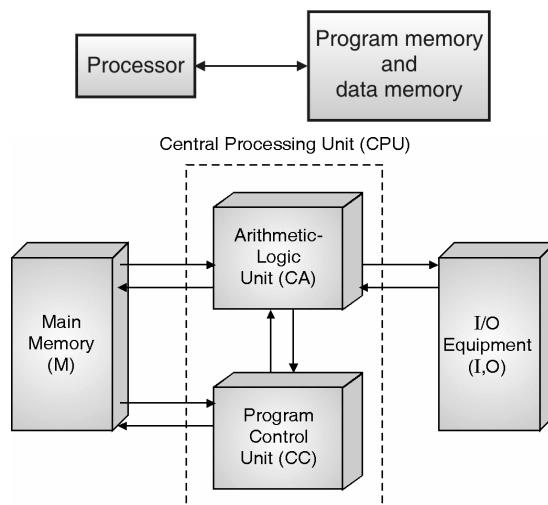


Fig. 9.3.1 : Von Neumann Architecture of a computer



- The name is derived from the mathematician and early computer scientist John Von Neumann.
- The computer has a common memory for data as well as code to be executed.
- The processor needs two clock cycles to complete an instruction, first to get an instruction and second to get the data.
- This system has three units CPU, Memory and I/O devices. The CPU has two units Arithmetic Unit and Control unit. Let us discuss these units in detail.

1. Input unit :

- A computer accepts inputs from the user through these devices i.e. input devices.
- The commonly used input devices are keyboard and mouse. Besides that, there are devices like joystick, camera, scanner etc. which are also input devices.
- The devices input the data accepted from the user in a proper coded form understood by the computer.

2. Output unit :

- The result is given back by the computer to the user through an output device. Input devices and output devices are also called as human interface devices, because they are used to interface the human to the computer.
- The mainly used output devices are monitor and printer. But there are many other output devices like plotter, speaker etc.

3. Arithmetic and Logic Unit (ALU) :

- Arithmetic or logic operations like multiplication, addition, division, AND, OR, EXOR etc. are performed by ALU.
- Operands are brought into the ALU, where the necessary operation is performed.

4. Control unit :

- The control unit as we know is the main unit that controls all the operations of the system, inside and outside the processor.
- The memory or I/O devices have to be controlled by the computer to perform the operation according to the instruction given to it.

5. Memory unit :

- Memory is used to store the programs and data for the computer.

- The instructions from the programs are taken by the processor, decoded and executed accordingly.
- The data is also stored in the memory.
- The data is taken from memory and the operation is performed on that data, as well as the results are stored in the memory.
- In some cases the input to an operation and the result may also be from input and output devices.
- Memory in the Von Neumann system has a special organization wherein the data and instructions are stored in the same memory.
- We will see about this in the subsequent section.

Key features of a Von Neumann machine :

- The Von Neumann machine uses stored program concept.
- The program and data are stored in the same memory unit.
- Each location of the memory has a unique address i.e. no two locations have the same memory address.
- Execution of instruction in Von Neumann machine is carried out in a sequential manner (unless explicitly altered by the program itself) from one instruction to the next.

Detailed structure of the CPU :

- The block diagram of the computer proposed by Von Neumann have a minimal number of registers along with the above blocks.
- This computer has a small set of instruction and an instruction was allowed to contain only one operand address.
- Fig. 9.3.2 gives the detailed structure of IAS CPU.
- The structure shown in Fig. 9.3.2 consists of the following registers.

Accumulator (AC) :

- It normally provides one of the operand to ALU and stores the result.

Data Register (DR) :

- It acts as buffer storage between the CPU and the main memory or I/O devices.

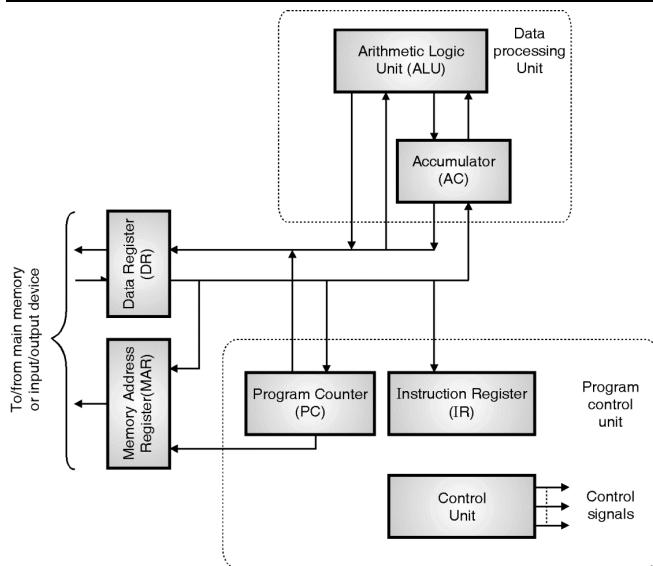


Fig. 9.3.2 : Structure of the CPU

Program Counter (PC) :

- It always contains the address of the next instruction to be executed.

Instruction Register (IR) :

- It holds the current instruction i.e. the opcode and operand of the instruction to be executed.

Memory Address Register (MAR) :

- The address from which the data or instruction is to be fetched is provided by the processor through MAR.
- It also is used to forward the address of memory location where data is to be stored.

Execution of a program by Von Neumann machine :

- The program to be executed is stored in memory.
- A register, PC (Program counter) always points to the first instruction of the program when the computer starts.
- CPU fetches the instruction pointed by PC. PC contents are automatically incremented to point to the next instruction.
- If the initial value of PC = 0000 (in binary), first instruction be fetched for execution. After fetching "Instruction number 1", PC will be incremented by one. It is assumed that the size of each instruction is 1 byte.

$$PC = PC + 1 = 0 + 1 = 1$$

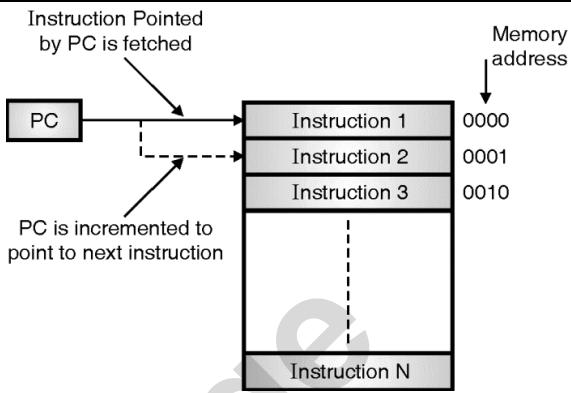


Fig. 9.3.3 : Instruction pointed by PC is fetched by the CPU for execution. Subsequently PC is made to point to the next instruction

Fetching an instruction :

- CPU interacts with memory through two special registers :
 - 1. MAR (Memory Address Register)** : It provides address of memory location from where data or instruction is to be retrieved or to which data is to be stored.
 - 2. DR (Data Register)** : It acts as buffer storage between the main memory and the CPU.
- The function and operation of these registers will be understood by the example below.
- The instruction to be executed is brought from the memory to the CPU, through the following steps :
 1. The address of the instruction is transferred from PC to MAR.

$$MAR \leftarrow PC$$

2. MAR puts this address on the address bus for selection of the required location of the memory.
3. Control Unit generates the RD (read control signal) signal to perform read operation on memory. Required instruction is given on data bus by the memory. Instruction on data bus is accepted in DR (Data Register).

Execution of instruction :

- The fetched instruction is in the form of binary code and is loaded into instruction register (IR) from DR (Data Register).

- The instruction specifies what action the CPU has to take.
- The CPU interprets the instruction and performs the required action. The action could be :
 1. Data transfer between CPU and memory.
 2. Data transfer between CPU and I/O.
 3. The CPU may perform an arithmetic or logic operation on data.

9.3.2 Harvard Architecture :

- Fig. 9.3.4 shows the connection for Harvard computer architecture.



Fig. 9.3.4

- The name is originated from Harvard's, "Harvard Mark I" a relay based old computer.
- In this case there are two separate memories for storing data and program.
- In this case the processor can simultaneously access instruction as well as the data and hence can complete an instruction execution in one cycle.

9.4 Basic Instruction Cycle :

- The instruction cycle is a representation of the states that the computer or the microprocessor performs when executing an instruction.
- The instruction cycle comprises of two main steps to be followed to execute the instruction, namely the fetch operation in the fetch cycle and the execution operation during the execute cycle.

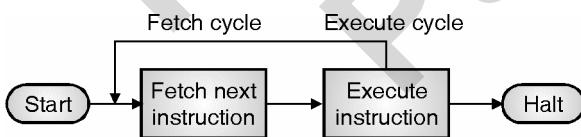


Fig. 9.4.1 : Basic instruction cycle

- Fig. 9.4.1 shows the basic instruction cycle. It comprises of the fetch and executes cycle in a loop to execute huge number of instructions, until it reaches the halt instruction.
- The fetch cycle comprises of the following operations :
 1. Program Counter (PC) holds address of next instruction to fetch; hence the CPU (Processor) fetches instruction from memory location pointed to by PC. This is done by providing the value of the PC to the MAR and giving the Read control signal to the memory. On this the memory

- provides the value in the given address (which is the instruction) to MBR.
- 2. The PC value has to be incremented to point to the next instruction (Sometimes the value of PC may have been completely changed in case of some special instructions called as branching instructions).
- 3. The instruction is loaded into Instruction Register (IR) from the MBR.
- 4. Finally the processor interprets or decodes the instruction. The processor performs required operations in the execute cycle.
- In the execute cycle the operation asked to be performed by the instruction is done. It may comprise of one or more of the following operations :
 1. Transfer of data between processor and memory or between processor and IO module.
 2. Processing of data like some arithmetic or logical operations on data.
 3. Change of the sequence of operation i.e. branching instructions.

9.4.1 Interrupt Cycle :

- Fetch and execute are not the only two states in the instruction cycle.
- There is one more state i.e. Interrupt cycle.
- In this subsection we will see the concept of interrupt in short and the interrupt cycle.
- Interrupt is a mechanism by which I/O modules can interrupt normal sequence of processing. Interrupt can be because of some request from an I/O device to service that particular device.
- This service may take or give data or some control operation.
- It may also be because of some unexpected operation in the program execution by the CPU itself.
- Interrupt cycle as discussed earlier is added to instruction cycle.
- During this cycle the processor checks for interrupt, and if present and enabled services the same.
- If no interrupt is present then it fetches the next instruction else if interrupt pending then it performs the following operations :
 1. Suspend the execution of current program.
 2. Save the context of the current program under execution.

3. Set the PC value to start address of interrupt handler routine also called as interrupt service routine. Interrupt service routine is a small program which when executed, services the interrupting source.
 4. Process the interrupt service routine (ISR) and then.
 5. Restore the context and continue execution of the interrupted program.
- Thus the complete basic instruction cycle with Interrupts can be as shown in the Fig. 9.4.2.

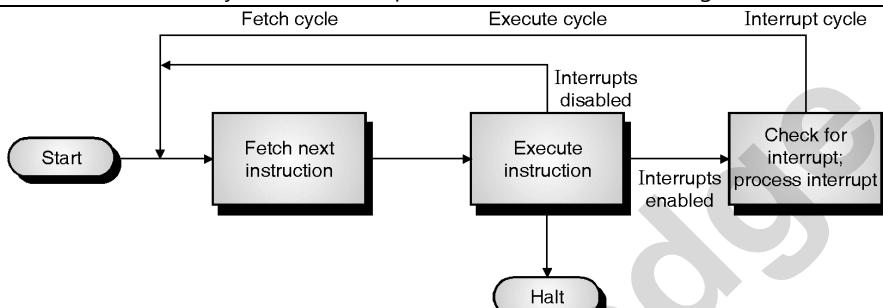


Fig. 9.4.2 : Complete basic instruction cycle

- You will notice in Fig. 9.4.2, the interrupts are checked for, after the execute cycle and processed if enabled and exist; else, it fetches the next instruction.
- The detailed instruction cycle is shown in Fig. 9.4.3.

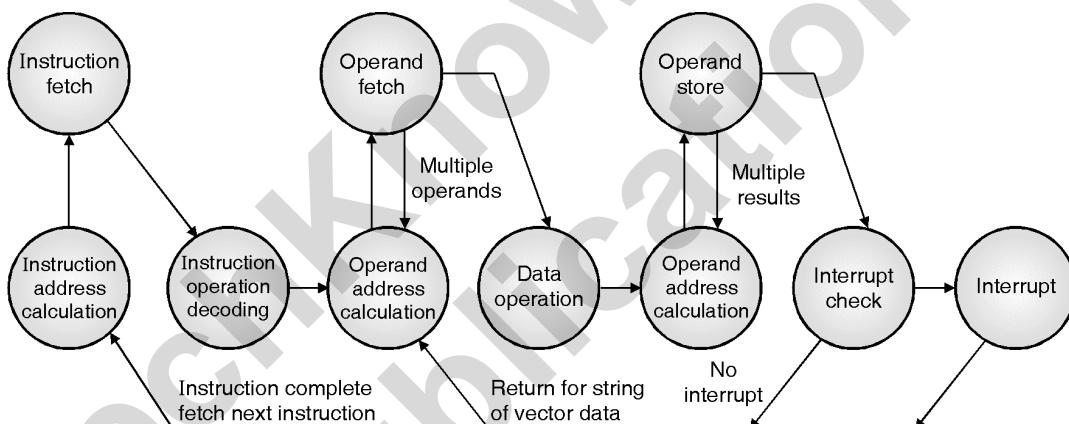


Fig. 9.4.3 : Detailed instruction cycle

- In Fig. 9.4.3, there are some states drawn on the upper side, while some on the lower side.
- The ones on the upper side are the operations carried out on the buses or are external operations, while the ones at the lower level are the operations carried out inside the CPU or are internal operations.
- The instruction cycle begins from the “Instruction address calculation” state, wherein the address of the next instruction is calculated or the value of the PC is updated.
- Then the instruction is fetched, which requires the operation on the buses.
- The instruction fetched is then decoded. Until this state, it is the fetch cycle.
- In the execute cycle, the operand address is calculated and the operands are fetched from the calculated address.
- Again to fetch the operands, we require the buses. After fetching the operand, if more operands are required for multiple operand instructions, then the next state is again calculate the operand address i.e. the address of the next operand.
- Once all the operands are fetched, the data operation is carried out as per the operation indicated in the instruction.
- Now for the result storage again the address of operand is calculated and the result is stored in the specified location of the memory.
- In case of multiple operands again the calculation and storage process for the operand continues until all the operands are stored.
- Now begins the interrupt cycle, wherein the first step is to check the presence of an enabled interrupt.

- If there is none, then the next state as seen in the Fig. 9.4.3 is the calculation of next instruction address i.e. executes the next sequential instruction.
- But in case the interrupt is present and enabled then the servicing of the same is done as discussed earlier in this section.
- In the Fig. 9.4.3, you will also notice that there are two paths from the end of the previous instruction.
- The one that goes to the state "Instruction address calculation" for the next instruction; and the one that goes to the "Operand address calculation" for vector instructions.
- Vector instructions are those instructions wherein the operation is same but the data on which the operation is to be performed in a huge block of data or an array of data.
- Hence in the second case, the instruction is already fetched and decoded i.e. the operation is already known, and the operation is to be performed on a block of data.
- After completing the operation on one set of operands, the CPU returns to the next operand address calculation state, wherein it calculates and fetches the next operand.
- Then it performs the operation, stores the result and again for the next set of operand, until all the operands in the array are completed.

9.5 CPU Architecture and Register Organization :

- Fig 9.5.1 shows the architecture of microprocessor. This architecture is divided in different groups as follows :
 1. Registers
 2. Arithmetic and logic unit
 3. Interrupt control
 4. Timing and control circuitry.

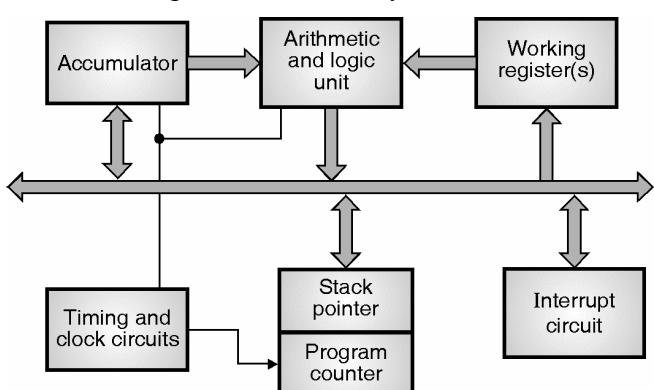


Fig. 9.5.1 : General architecture of a microprocessor

- It consists of PIPO (Parallel in parallel out) register as shown in Fig. 9.5.2.
- This section is also called as scratch pad memory. It stores data and address of memory.
- The register organization affects the length of program, the execution time of program and simplification of the program.
- To achieve better performance, the number of registers should be large.
- The architecture of microcomputer depends upon the number and type of the registers used in microprocessor.
- It consists 8-bit registers or 16 bit registers.
- The register section varies from microprocessor to microcomputer.
- The registers are used to store the data and address.
- These registers are classified as :
 - o Temporary registers
 - o General purpose registers
 - o Special purpose registers.

1. Register section :

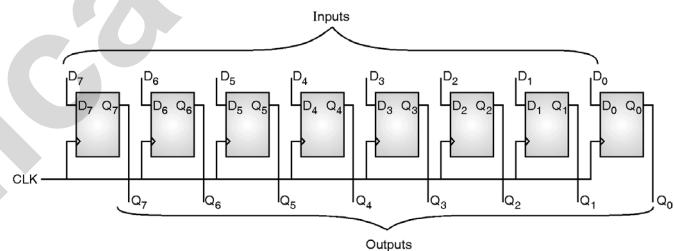


Fig. 9.5.2 : 8 bit register

2. Arithmetic and logical unit :

- This section processes data i.e. it performs arithmetic and logical operations.
- It performs arithmetic operations like addition, subtraction and logical operations like ANDing, ORing, EX-ORing, etc.
- The ALU is not available to the user. Its word length depends upon the width of an internal data bus.
- The ALU is controlled by timing and control circuits.
- It accepts operands from memory or register. It stores result of arithmetic and logic operations in register or memory.
- It provides status of result to the flag register. Flag register shows status of result.
- ALU looks after the branching decisions.

3. Interrupt control :

- This block accepts different interrupt request inputs. When a valid interrupt request is present it informs control logic to take action in response to each signal.

4. Timing and control unit :

- This is a control section of microprocessor made up of synchronous sequential logic circuit.
- It controls all internal and external circuits.
- It operates with reference to clock signal.
- This accepts information from instruction decoder and generates micro steps to perform it.
- In addition to this, the block accepts clock inputs, performs sequencing and synchronising operations.
- The synchronization is required for communication between microprocessor and peripheral devices.
- To implement this it uses different status and control signals.
- The basic operation of a microprocessor is regulated by this unit.
- It synchronizes all the data transfers.
- This unit takes appropriate actions in response to external control signals.

9.5.1 Interrupt Control :

- This block accepts different interrupt request inputs. When a valid interrupt request is present it informs control logic to take action in response to each signal.

9.5.2 Timing and Control Unit :

- This is a control section of microprocessor made up of synchronous sequential logic circuit.
- It controls all internal and external circuits.
- It operates with reference to clock signal.
- This accepts information from instruction decoder and generates microsteps to perform it.
- In addition to this, the block accepts clock inputs, performs sequencing and synchronising operations.
- The synchronization is required for communication between microprocessor and peripheral devices.
- To implement this it uses different status and control signals.
- The basic operation of a microprocessor is regulated by this unit.
- It synchronizes all the data transfers.
- This unit takes appropriate actions in response to external control signals.

9.6 Instruction, Micro-instructions and Micro-operations : Interpretation and Sequencing :

- The structure of the CPU seen in section 9.5 is shown in details in Fig. 9.6.1.
- This structure has a speciality that all the control signals are shown in it.
- Programs are executed as a sequence of instructions. As seen in the previous sections of this chapter, each instruction consists of a series of steps that make up the instruction cycle i.e. fetch, decode, etc. Each of these steps is, in turn, made up of a smaller series of steps called micro-operations or micro-instructions.
- Control signals are issued to perform these micro-operations and micro-instructions are these control signals.
- Fig. 9.6.1 shows the structure of the CPU with these micro-instructions or the control signals.
- It also shows those registers as already seen in section 9.2 like PC, MAR, MBR, etc.
- There are some registers like the register 'Y' to provide one of the operand to the ALU as shown in the Fig. 9.6.1.

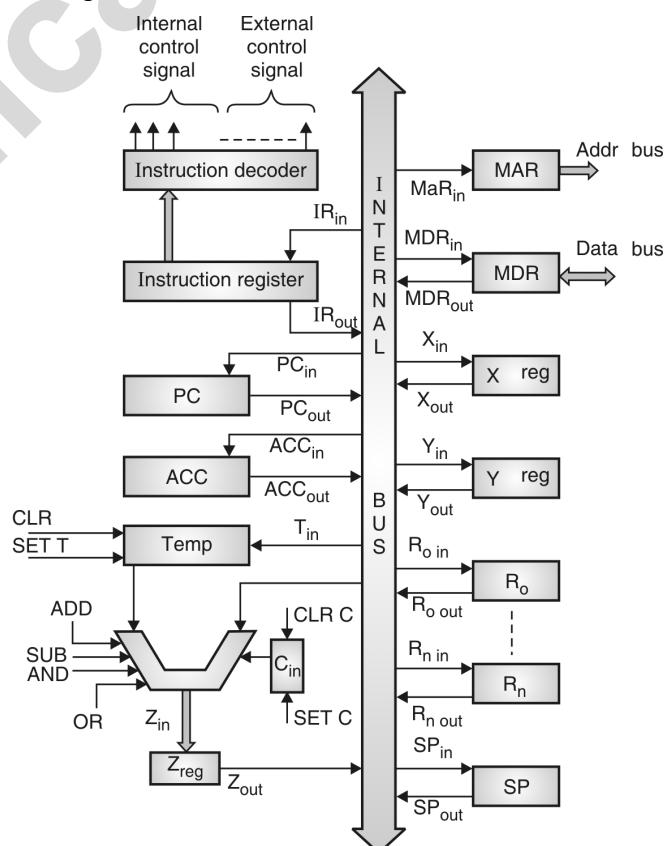


Fig. 9.6.1 : Data path structure with control signals



- Another register is the 'Z' register, which is used to store the result given by the ALU.
- A "temp" register or the temporary register to store some temporary data.
- The set of registers R_0 to R_n (the value of 'n' depends on the registers in the CPU) for general purpose operations.
- There is also an instruction decoder for decoding the instructions stored in the instruction register and in turn provides the micro-instructions or the control signals for the resources inside and outside the CPU.
- The ALU also gets the control signals from this decoder indicating the operation to be performed like Add, Sub, and AND etc.
- The ALU also has an extra input called as C_{in} i.e. the carry input as required for adder.
- To execute any instruction as seen earlier it is to be divided into three cycles viz. fetch, execute and interrupt cycles.
- The execute cycle will differ based on the operation to be carried out in the instruction, but the fetch and interrupt cycle will be common for all the cycles.
- Let us see the micro-instructions to be given for each of these cycles.

9.6.1 Fetch Cycle :

- Fetch cycle is concerned to fetch (i.e. read from memory) the instruction.
- It involves following operations in different t-states (t-state is a time state and is equal to one clock pulse) and hence the mentioned microinstructions in Table 9.6.1.

Table 9.6.1 : Microinstructions for the fetch cycle

	Operation	Microinstructions
T1	$PC \rightarrow MAR$	$PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}$
T2	$M \rightarrow MBR$ $PC \leftarrow PC + 1$	$Z_{out}, PC_{in}, Wait for memory fetch cycle$
T3	$MBR \rightarrow IR$	MBR_{out}, IR_{in}

- As seen in the table, three clock pulses or t-states are required for the fetch cycle.
- Note, the control unit is an organizational part of the CPU; hence the design can vary from processor to processor.
- In the first t-state, the address of the instruction to be executed is given to the MAR register from the PC register.

- To perform this operation the control signals given are PC_{out} and MAR_{in} .
- This will make the PC register give out its data and the MAR register accept this data.
- Also the memory is indicated to perform a read operation from memory hence the signal "Read".
- To increment the value of PC, the various operations are performed on ALU signals i.e. Clear Y, Set C_{in} , Add, Z_{in} .
- The 'Y' register is cleared and the carry flag is set. Now when the ALU is said to perform the "ADD" operation it will add the contents of the 'Y' register, carry flag and the contents of the internal data bus.
- The contents of the internal data bus are nothing but the value given out by the PC register.
- Hence the PC is added with '1' i.e. the carry flag and hence incremented value of PC is given to the 'Z' register.
- In the second clock pulse the CPU has to wait for the memory operation, but in the same time it can transfer the result in 'Z' register to the PC register with the control signals namely Z_{out} and PC_{in} .
- This could not be done in the previous t-state, as two data cannot be given simultaneously on the data bus, else it will get mixed up.
- Only one data can be given on the data bus in any clock pulse, but as many as required can accept the data.
- In the final t-state, the contents received from the memory i.e. the instruction is transferred to its correct place i.e. the instruction register.
- This is done by the control signals namely MBR_{out} and IR_{in} . This also completes the entire fetch operation of the instruction.

9.6.2 Execute Cycle :

- Execute cycle as discussed can be of various types based on the operation to be performed in the instruction and the location of the operand. We will see some examples in this subsection.
- The first example we will take for the execution of a direct addressed operand.
- In this case the address of the operand is directly given in the instruction.
- It involves different operations in various t-states as shown in Table 9.6.2 assuming the instruction ADD R1, [X].

**Table 9.6.2 : Microinstructions for the execute cycle of direct addressed mode of operand access**

	Operation	Microinstructions
T1	IR → MAR	IR _{out} (address), MAR _{in} , Read, Clear C _{in}
T2	M → MBR	R1 _{out} , Y _{in} , Wait for memory read cycle
T3		MBR _{out} , Add, Z _{in}
T4	MBR + R1 → R1	Z _{out} , R1 _{in}

- In this case of direct addressing mode, the address of the memory operand is in the instruction itself.
- The instruction as we have seen in the fetch cycle reaches the IR register.
- Hence the IR register is given a signal to give out the address part and the MAR register to accept this address input value by giving the control signals IR_{out}(address) and MAR_{in}.
- At the same time, since the memory is to be read from the control signal is given to the memory i.e. "Read".
- Also the carry flag is cleared to get ready for the addition operation.
- Since the instruction expects addition of the register 'R1' and the data at memory location with address 'X', the contents of register 'R1' are transferred to the 'Y' register, which is one of the operands for any ALU operation.
- To perform this transfer operation the control signals given are R1_{out} and Y_{in}.
- Also by the end of the second t-state, the data operand required from the memory will be available in the MBR register.
- In the third t-state the contents of the MBR, which is the content of memory location with the address 'X', is placed on the internal data bus and the ALU is indicated to perform the addition operation.
- It adds the contents of the 'Y' register and the contents of the internal data bus, and the result is given to the 'Z' register.
- An extra t-state is required to send the data from the 'Z' register to the register R1, as seen earlier two data

cannot be given simultaneously on the data bus in the same t-state.

- And the contents of memory location with the address 'X' are already put on the data bus in the third t-state.
- The fourth t-state is thus required to transfer the data from register 'Z' to register R1 using the signals Z_{out}, R1_{in}.
- Another execute cycle we will be studying in this sub-section is for the indirect addressed operand. In this case, the address given in the instruction is the memory location that contains the address of the operand.
- Table 9.6.3 shows the micro-operations required for such an execute cycle for an example instruction ADD R1, [X].
- Table 9.6.3 shows the control signals to be given exactly similar to that of the Table 9.6.2, with a minor difference i.e. the value received in the MBR on first memory read is the operand address and hence is to be given back to the memory to fetch the actual operand.

Table 9.6.3 : Microinstructions of the execute cycle of an indirect addressed operand instruction

	Operation	Microinstructions
T1	IR → MAR	IR _{out} (address), MAR _{in} , Read, Clear C _{in}
T2	M → MBR	R1 _{out} , Y _{in} , Wait for memory read cycle
T3	MBR → MAR	MBR _{out} (address), MAR _{in} , Read
T4	M → MBR	Wait for memory read cycle
T5		MBR _{out} , Add, Z _{in}
T6	MBR + R1 → R1	Z _{out} , R1 _{in}

9.6.3 Interrupt Cycle :

- It is concerned to perform the test for any pending interrupts at the end of every instruction execution and if an interrupt occurs.
- It involves the different micro-operations for various t-states as shown in Table 9.6.4 points to the top of the stack.
- This stack is used to store the return address of the interrupted program.

**Table 9.6.4 : Microinstructions for the interrupt cycle**

	Operation	Microinstructions
T1	SP \leftarrow SP - 1	SP _{out} (address), Decrement, Z _{in}
T2		Z _{out} , MAR _{in} , SP _{in}
T3	PC \rightarrow MBR	PC _{out} (return address), MBR _{in} , Write
T4	ISR address \rightarrow PC	ISR address out, PC _{in} (new address), Wait for memory write cycle

- The control signals are to be generated using the control unit. The design of this control unit can be done in two ways namely: Hardwired Control Unit and Microprogrammed Control Unit. We will see these two methods in the subsequent sections.

9.6.4 Examples of Microprograms :

1. Write a microprograms for the instruction : MOV R₃, R₄

T-state	Operation	Microinstructions
T1	PC \rightarrow MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M \rightarrow MBR PC \leftarrow PC + 1	Z _{out} , PC _{in} , Wait for memory fetch cycle
T3	MBR \rightarrow IR	MBR _{out} , IR _{in}
T4	R ₃ \leftarrow R ₄	R ₄ _{out} , R ₃ _{in}
T5	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z _{in} ,
T6	SP \leftarrow SP - 1	Z _{out} , SP _{in} , MAR _{in}
T7	PC \rightarrow MDR	PC _{out} , MDR _{in} , WRITE
T8	MDR \rightarrow [SP]	Wait for mem access
T9	PC \leftarrow ISR addr	PC _{in} ISR addr out

2. Write a microprogram for the instruction : ADD R₃, R₄

T-state	Operation	Microinstructions
T1	PC \rightarrow MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M \rightarrow MBR PC \leftarrow PC + 1	Z _{out} , PC _{in} , Wait for memory fetch cycle

T-state	Operation	Microinstructions
T3	MBR \rightarrow IR	MBR _{out} , IR _{in}
T4	R ₃ \rightarrow ALU	R ₃ _{out} , X _{in} , CLRC
T5	R ₄ \rightarrow ALU	R ₄ _{out} , ADD, Z _{in}
T6	Z \rightarrow R ₃	Z _{out} , R ₃ _{in}
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z _{in} ,
T8	SP \leftarrow SP - 1	Z _{out} , SP _{in} , MAR _{in}
T9	PC \rightarrow MDR	PC _{out} , MDR _{in} , WRITE
T10	MDR \rightarrow [SP]	Wait for mem access
T11	PC \leftarrow ISR addr	PC _{in} ISR addr out

3. Write a microprogram for the instruction:
MOV R₃ , [R₄] OR LOAD R₃ , [R₄]

T-state	Operation	Microinstructions
T1	PC \rightarrow MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M \rightarrow MBR PC \leftarrow PC + 1	Z _{out} , PC _{in} , Wait for memory fetch cycle
T3	MBR \rightarrow IR	MBR _{out} , IR _{in}
T4	R ₄ \rightarrow MAR	R ₄ _{out} , MAR _{in} , READ
T5	Mem \rightarrow MDR	Wait for mem access
T6	MDR \rightarrow R ₃	MDR _{out} , R ₃ _{in}
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z _{in} ,
T8	SP \leftarrow SP - 1	Z _{out} , SP _{in} , MAR _{in}
T9	PC \rightarrow MDR	PC _{out} , MDR _{in} , WRITE
T10	MDR \rightarrow [SP]	Wait for mem access
T11	PC \leftarrow ISR addr	PC _{in} ISR addr out

4. Write a microprogram for the instruction : ADD R₃, [R₄]

T-state	Operation	Microinstructions
T1	PC \rightarrow MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M \rightarrow MBR PC \leftarrow PC + 1	Z _{out} , PC _{in} , Wait for memory fetch cycle
T3	MBR \rightarrow IR	MBR _{out} , IR _{in}
T4	R ₄ \rightarrow MAR	R ₄ _{out} , MAR _{in} , READ, CLRC
T5	Mem \rightarrow MDR	Wait for men access



T-state	Operation	Microinstructions
T5	MDR → ALU	MD R_{out} , Z_{in} ADD
T6	$R_3 \rightarrow X_1$	R_{3out}, X_{in}
T7	$Z \rightarrow R_3$	Z_{out}, R_{3in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z_{in} ,
T9	$SP \leftarrow SP - 1$	$Z_{out}, SP_{in}, MAR_{in}$
T10	PC → MDR	PC _{out} , MDR _{in} , WRITE
T11	MDR → [SP]	Wait for mem access
T12	PC ← ISR addr	PC _{in} ISR addr out

5. Write a microprogram for the instruction :

ADD $R_3, [R_4]$]

T-state	Operation	Microinstructions
T1	PC → MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in} , Wait for memory fetch cycle
T3	MBR → IR	MBR _{out} , IR _{in}
T4	mem → MDR	Wait for mem access
T5	MDR → ALU	MDR _{out} , Z_{in} , ADD
T6	$Z \rightarrow R_3$	Z_{out}, R_{3in}
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z_{in} ,
T8	$SP \leftarrow SP - 1$	$Z_{out}, SP_{in}, MAR_{in}$
T9	PC → MDR	PC _{out} , MDR _{in} , WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← ISR addr	PC _{in} ISR addr out

6. Write a microprogram for the instruction :

ADD $R_3, 45H$

T-state	Operation	Microinstructions
T1	PC → MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in} , Wait for memory fetch cycle
T3	MBR → IR	MBR _{out} , IR _{in}

T-state	Operation	Microinstructions
T4	$R_3 \rightarrow X$	$R_{out}, X_{in}, CLRC$
T5	IRdata → ALU	IR _{out} , ADD, Z_{in}
T6	$Z \rightarrow R_3$	Z_{out}, R_{3in}
T7	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z_{in} ,
T8	$SP \leftarrow SP - 1$	$Z_{out}, SP_{in}, MAR_{in}$
T9	PC → MDR	PC _{out} , MDR _{in} , WRITE
T10	MDR → [SP]	Wait for mem access
T11	PC ← ISR addr	PC _{in} ISR addr out

7. Write a microprogram for the instruction : ADD $R_3, [45H]$

T-state	Operation	Microinstructions
T1	PC → MAR	PC _{out} , MAR _{in} , Read, Clear y, Set C _{in} , Add, Z _{in}
T2	M → MBR PC ← PC + 1	Z_{out}, PC_{in} , Wait for memory fetch cycle
T3	MBR → IR	MBR _{out} , IR _{in}
T4	IR addr → MAR	IR _{out} , MAR _{in} , D, CLRC
T5	$R_3 \rightarrow X$	R_{3out}, X_{in}
T5	mem → MDR	Wait for mem access
T6	MDR → ALU [$R_3 + 45$]	MDR _{out} , Z_{in} , ADD
T7	$Z \rightarrow R_3$	Z_{out}, R_{3in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SP _{out} , SUB, Z_{in} ,
T9	$SP \leftarrow SP - 1$	$Z_{out}, SP_{in}, MAR_{in}$
T10	PC → MDR	PC _{out} , MDR _{in} , WRITE
T11	MDR → [SP]	Wait for mem access
T12	PC ← ISR addr	PC _{in} ISR addr out

8. Write a microprogram for the instruction ADDX, [Y]

T-state	Operation	Microinstruction
T1	PC → MAR	PC _{out} , MAR _{in} , READ, CLRT, STC, ADD, Z

T-state	Operation	Microinstruction
T2	mem → MDR PC ← PC + 1	Wait for mem access Z_{out} , PC_{in}
T3	MDR → IR	MDR_{out} , IR_{in}
T4	Y → MAR	y_{out} , MAR_{in} , READ, CLRC
T5	mem → MDR X → Temp	Wait for mem access X_{out} , T_{in}
T6	MDR → ALU	MDR_{out} , Z_{in} , ADD
T7	Z → X	Z_{out} , X_{in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SPout, SUB, Zin,
T9	SP ← SP – 1	Z_{out} , SP_{in} , MAR_{in}
T10	PC → MDR	PC_{out} , MDR_{in} , WRITE
T11	MDR → [SP]	Wait for mem access
T12	PC ← ISR addr	PC_{in} ISR addr out

9. Write a microprogram for the instruction :

ADD X, [[400]]

T-state	Symbolic operations	Microinstruction
T1	PC → MAR	PC_{out} , MAR_{in} , READ, CLRT, SETC, ADD, Z
T2	mem → MDR PC ← PC + 1	Wait for mem access Z_{out} , PC_{in}
T3	MDR → IR	MDR_{out} , IR_{in}
T4	IRaddr → MAR	IR_{out} , MAR_{in} , READ, CLRC
T5	mem → MDR X → Temp	Wait for mem access X_{out} , T_{in}
T6	MDR → MAR	MDR_{out} , MAR_{in} , READ
T7	mem → MDR	Wait for mem access
T8	MDR → ALU	MDR_{out} , ADD, Z_{in}
T9	Z → X	Z_{out} , X_{in}
T8	Check for intr	Assumption enabled intr pending CLRX, SETC, SPout, SUB, Zin,
T9	SP ← Sp – 1	Z_{out} , SP_{in} , MAR_{in}

T-state	Symbolic operations	Microinstruction
T10	PC → MDR	PC_{out} , MDR_{in} , WRITE
T11	MDR → [SP]	Wait for mem access
T12	PC ← ISR addr	PC_{in} ISR addr out

9.6.5 Applications of Microprogramming :

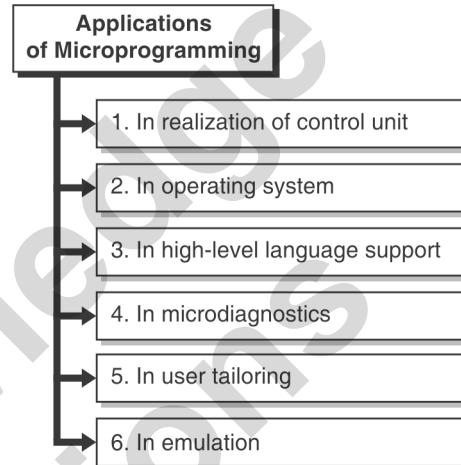


Fig. 9.6.2 : Applications of Microprogramming

The applications of microprogramming are :

- In Realization of control unit :** Microprogramming is used widely now for implementing the control unit of computers.
- In Operating system :** Microprograms can be used to implement some of the primitives of operating system. This simplifies operation system implementation and also improves the performance of the operating system.
- In High-Level Language support :** In High-Level language various sub functions and data types can be implemented using microprogramming. This makes compilation into an efficient machine language from possible.
- In Micro diagnostics :** Microprogramming can be used for detection isolation monitoring and repair of system errors. This known as micro diagnostics and they significant enhance system maintenance.
- In User Tailoring :** By using RAM for implementing control memory (CM), it is possible to tailor the machine to different applications.
- In Emulation :** Emulation refers to the use of a microprograms on one machine to execute programs originally written for another machine. This is used widely as an aid for users in migrating from one computer to another.

9.7 Control Unit : Hardwired Control Unit Design Methods :

- The hardwired Control unit is viewed as a sequential and combinational logic circuit.
- It is used to generate a set of fixed sequences of control signals. It is implemented using any of a variety of "standard" digital logic circuits.
- The major advantages of hardwired control units are higher speed of operation and smaller space required for implementation on silicon wafer i.e. the IC (Integrated Circuit), since the components required are lesser.
- The only disadvantage is that modifications to the design are slightly difficult.
- The use of hardwired control unit is majorly found in the RISC designs.
- There are different methods to implement hardwired control unit :
 - State table method.
 - Delay-Element method.
 - Sequence counter method.
 - PLA method.

1. State table method :

- In this method state transition for each instruction is made and hence a state table is obtained.
- This state table is then combined to form an instruction set state table, where all the instructions (OPCODE) are considered as inputs and according to this the next state is being determined. Each state with a set of microinstructions to be issued to various components of the processor as well as external control signals.
- This state table is then implemented using flip-flops and combinational circuit to generate different control signals.
- An example state table implementation is shown in Fig. 9.7.1.

Inputs				
State	I	I ₂	I _m
S ₁	S _{1, 1} , O _{1, 2}	S _{1, 2} , O _{1, 2}	S _{1, m} , O _{1, m}
S ₂	S _{2, 1} , O _{2, 1}	S _{2, 2} , O _{2, 2}	S _{2, m} , O _{2, m}
:			
S _n	S _{n, 1} , O _{n, 1}	S _{n, 2} , O _{n, 2}	S _{n, m} , O _{n, m}

Fig. 9.7.1(a) : Mealy

Inputs					
State	I	I ₂	I _m	Outputs
S ₁	S _{1, 1}	S _{1, 2}	S _{1, m}	O ₁
S ₂	S _{2, 1}	S _{2, 2}	S _{2, m}	O ₂
:				
S _n	S _{n, 1}	S _{n, 2}	S _{n, m}	O _n

(b) Moore Type

Fig. 9.7.1 : State tables for a finite-state machine

2. Delay element method :

- This method is implemented using delay elements i.e. D-flipflops.
- A flipflop is made to give output logic '1' after the specific event or in a t-state in sequence and the outputs of these flipflops are used to generate control signals or the micro-instructions i.e. two operations that require a delay of 1 t-state between them are separated by a D flipflop between them. Fig. 9.7.2 shows this implementation.

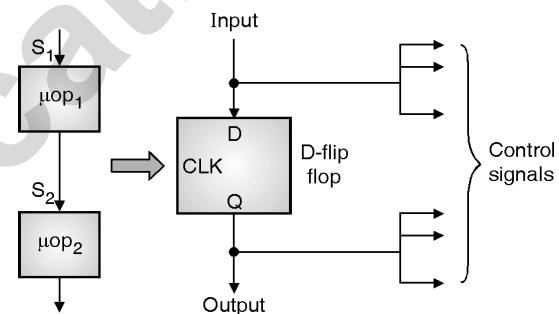


Fig. 9.7.2 : Use of D flip flop as a delay element between two sets of control signals

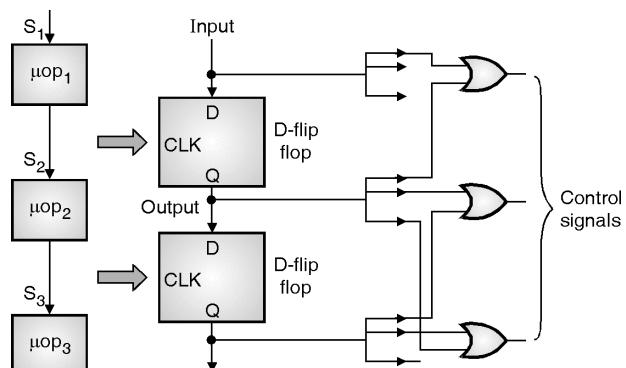


Fig. 9.7.3 : Use of OR gate in delay element method of Hardwired control unit

- The signals that activate the same control signal are ORed together i.e. if a signal has to be activated from

the outputs of multiple flipflops then an OR gate is used as shown in Fig. 9.7.3.

- In case if a decision is to be made then it is implemented using a If-Then-Else circuit i.e. two AND gates coupled to a OR gate. This is shown in Fig. 9.7.4.

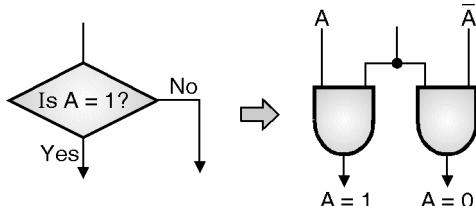


Fig. 9.7.4 : Implementation of If-Then-Else in delay element method of Hardwired control unit

3. Sequence counter method :

- In this method, multiple clock signals are derived from the master clock using a standard counter-decoder approach as shown in the Fig. 9.7.5. These signals are applied to the combinational portion of the circuit.

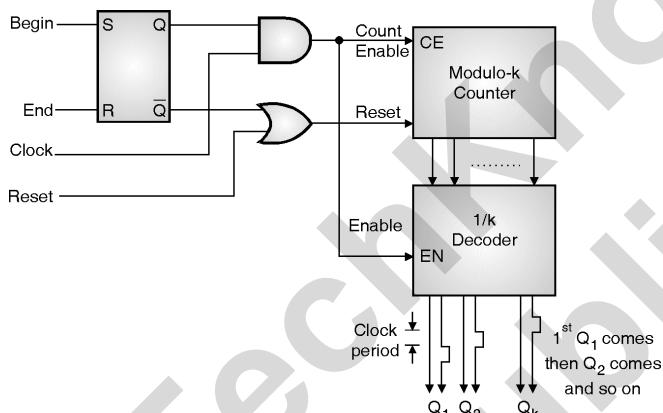


Fig. 9.7.5 : Sequential counter method of Hardwired control unit implementation

- As shown in Fig. 9.7.5, the counter keeps on incrementing and generating different counts.
- The counts are decoded using a decoder and the decoder outputs are given to various components as control signals in the CPU.

4. PLA method :

- In this method a PLA (Programmable Logic Array) is used to generate the control signals.
- PLA is an array of AND gates at input and the OR gates at output.
- The inputs are to be given to the AND gates, which can be connected to the specific OR gates as required.
- The OR gates outputs are the outputs of the overall PLA and are used as control signals in the system i.e. the

inputs to the AND array is from various control signals generated and the output of the OR array is given as control signals to various components of the processor as well as the external control signals required.

- Fig. 9.7.6 shows the implementation of the PLA method of implementation of control unit.

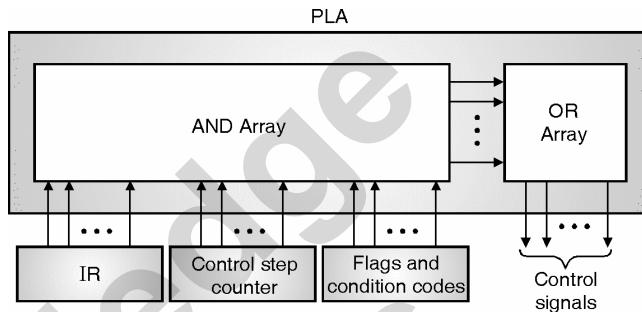


Fig. 9.7.6 : PLA Technique

9.8 Control Unit : Soft Wired (Micro programmed) Control Unit Design Methods :

- Micro programmed control unit generates control signals based on the microinstructions stored in a special memory called as the control memory.
- Each instruction points to a corresponding location in the control memory that loads the control signals in the control register.
- The control register is then read by a sequencing logic that issues the control signals in a proper sequence.
- The implementation of the micro programmed is shown in the Fig. 9.8.1.

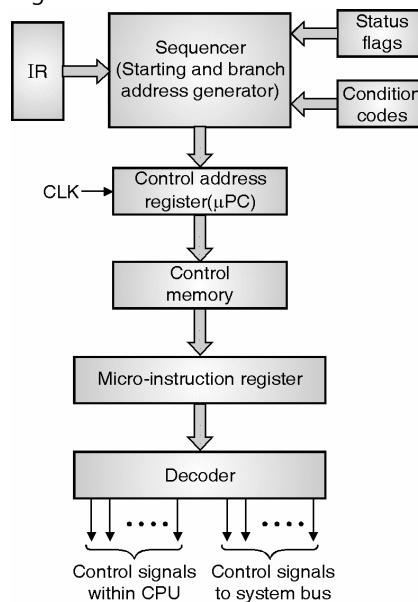


Fig. 9.8.1 : Micro programmed control unit

- The Instruction Register (IR), Status flag and condition codes are read by the sequencer that generates the address of the control memory location for the corresponding instruction in the IR.
- This address is stored in the Control address register that selects one of the locations in the control memory having the corresponding control signals.
- These control signals are given to the microinstruction register, decoded and then given to the individual components of the processor and the external devices.

9.8.1 Wilkie's Microprogrammed Control Unit :

- First working model of a micro-programmed control unit was proposed by Wilkie's in 1952.
- In the above design, a microinstruction has two major components :
 - Control field
 - Address field

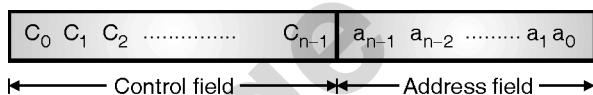


Fig. 9.8.2 : A typical microinstruction

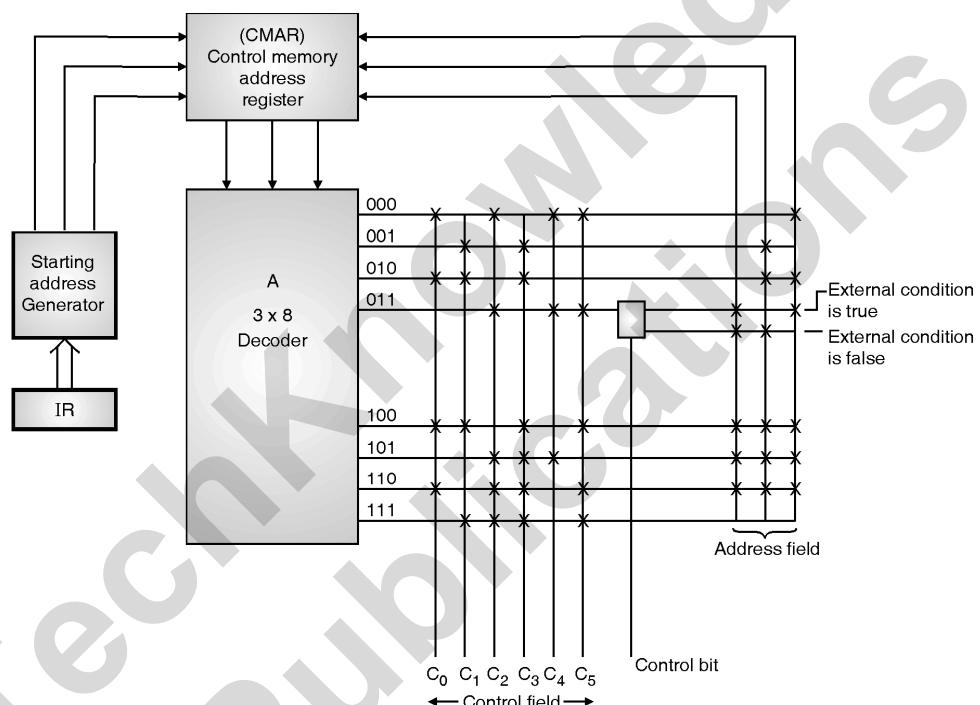


Fig. 9.8.3 : Wilkie's control

- If a microinstruction is encoded as given below :

C_0	C_1	C_2	C_3	C_4	C_5	C_6	A_2	A_1	A_0
0	1	0	0	1	1	0	0	1	0

- Then the control information 0100110 indicates that on execution of above microinstruction, control signals C_1 , C_4 and C_5 will be activated. Address field contains the address of the next microinstruction.
- Thus, after execution of the above instruction, the next instruction to be executed is one which is at the address 010.
- The control field tells the control signals which are to be activated and the address field provide the address of the next microinstruction to be executed.
- In Wilkie's control, control memory is organized as a program logic array.
- The Control Memory Access Register (CMAR) can be loaded from an external source (instruction register) as well as from the address field of a microinstruction.



- A machine instruction typically provides the starting address of a micro-program in control memory.
- On the basis of starting address from instruction register, decoder activates one of the eight output lines.
- This activated line, in turn, generates control signals and the address of the next microinstruction to be executed.
- This address is once again fed to the CMAR resulting in activation of another control line and address field.
- This cycle is repeated till the execution of the instruction is achieved.
- For example, as shown below, if the machine instruction under execution causes the decoder to have an entry address for a machine instruction in control memory at line 000. The decoder activates the lines in the sequence given below :

Line activated	Control signal generated	Address of next microinstruction
000	C ₀ , C ₂ , C ₄ , C ₅	001
001	C ₁ , C ₃	010
010	C ₀ , C ₁ , C ₃	011
011	C ₂ , C ₄ , C ₅	2

- On execution of microinstruction at address 011, address of the next microinstruction depends on the external condition.
- If the condition is true then the address 101 will be selected else the address 110 will be selected.

9.8.2 Comparison between Hardwired and Micro-programmed Control :

Attribute	Hardwired Control	Micro-programmed Control
Speed	Fast	Slow
Cost of implementation	More	Cheaper
Implementation approach	Sequential circuit	Programming

Attribute	Hardwired Control	Micro-programmed Control
Flexibility	Not flexible, difficult to modify for new instruction.	Flexible, new machine instructions can easily be added.
Ability to handle Complex instructions	Difficult	Easier
Design process	Complicated	Systematic
Decoding and sequencing logic	Complex	Easy
Applications	RISC μ p	CISC μ p
Instruction set size	Small	Large
Control memory	Absent	Present
Chip area required	Less	More

Review Questions

- Q. 1 Explain structural components of a computer.
- Q. 2 With neat block diagram explain von neumann architecture of a computer.
- Q. 3 With neat block diagram explain harvard architecture.
- Q. 4 With neat block diagram explain general architecture of a microprocessor.
- Q. 5 Explain detailed instruction cycle.
- Q. 6 List and explain applications of microprogramming.
- Q. 7 What are the different methods to implement hardwired control unit explain state table method?
- Q. 8 What are the different methods to implement softwired control unit?
- Q. 9 Differentiate between hardwired and micro-programmed control unit.

Unit 5

Chapter 10

Processor Instructions & Processor Enhancements

Syllabus

Instruction : Elements of machine instruction; Instruction representation (Opcode and mnemonics, Assembly language elements) ; Instruction format and 0-1-2-3 address formats, Types of operands, Addressing modes; Instruction types based on operations (Functions and examples of each); Key characteristics of RISC and CISC; Interrupt : Its purpose, Types, Classes and interrupt handling (ISR, Multiple interrupts), Exceptions; Instruction pipelining (Operation and speed up)

Multiprocessor systems : Taxonomy of Parallel Processor Architectures, Two types of MIMD clusters and SMP (Organization and benefits) and multicore processor (Various alternatives and advantages of multicores), Typical features of multicore intel core i7.

Case Study : 8086 Assembly language programming.

Chapter Contents

10.1 Instruction Encoding Format	10.7 Pipeline Processing
10.2 Instruction Format and 0-1-2-3 Address Formats	10.8 Instruction Pipelining and Pipelining Stages
10.3 Addressing Modes	10.9 Pipeline Hazards
10.4 Instruction Set of 8085	10.10 Multiprocessor Systems and Multicore Processor (Intel Core i7 Processor)
10.5 Reduced Instruction Set Computer Principles	10.11 Flynn's Classifications
10.6 Polling and Interrupts	



10.1 Instruction Encoding Format :

- In 8085 we have variety in instructions; therefore all instructions will not be of same size.
- The instructions vary from 1 to 6 bytes in length.
- The obvious question is, **what these bytes will contain and on what parameter the length of the instruction bytes is decided ?** The length of instruction bytes is dependent upon addressing mode used by programmer. i.e. immediate, register, register relative, based indexed, relative based indexed and so on.

Basically instruction bytes will contain information of :

- (1) OPCODE
- (2) Addressing mode designations :
 - (a) 2 bits Effective Address.
 - (b) 1 or 2 bits displacement.
 - (c) 1 or 2 bits immediate operand.

- To understand more clearly, let's observe Fig. 10.1.1 carefully.
- 1. Normally first bits is OPCODING byte.
- 2. 2nd bits normally specifies addressing mode. (Remember MOD and R/M). Sometime it may also contain OPCODING part.
- 3. After OPCODING and addressing mode bytes, we have following different cases :
 - (a) No additional bytes (Figs. 10.1.1(a), (b), (c) and (d)).
 - (b) A 2 bits EA (for direct addressing mode (Fig. 10.1.1(e)).
 - (c) A 1 or 2 bits immediate operand (Fig. 10.1.1(f)).
 - (d) A 1 or 2 bits displacement followed by 1 or 2 bits immediate operand (Fig. 10.1.1(g)).
- 4. If a displacement or immediate operand is 2 bytes long, the low order bits always appears first, this is Intel standard (same was followed by 8085 also).

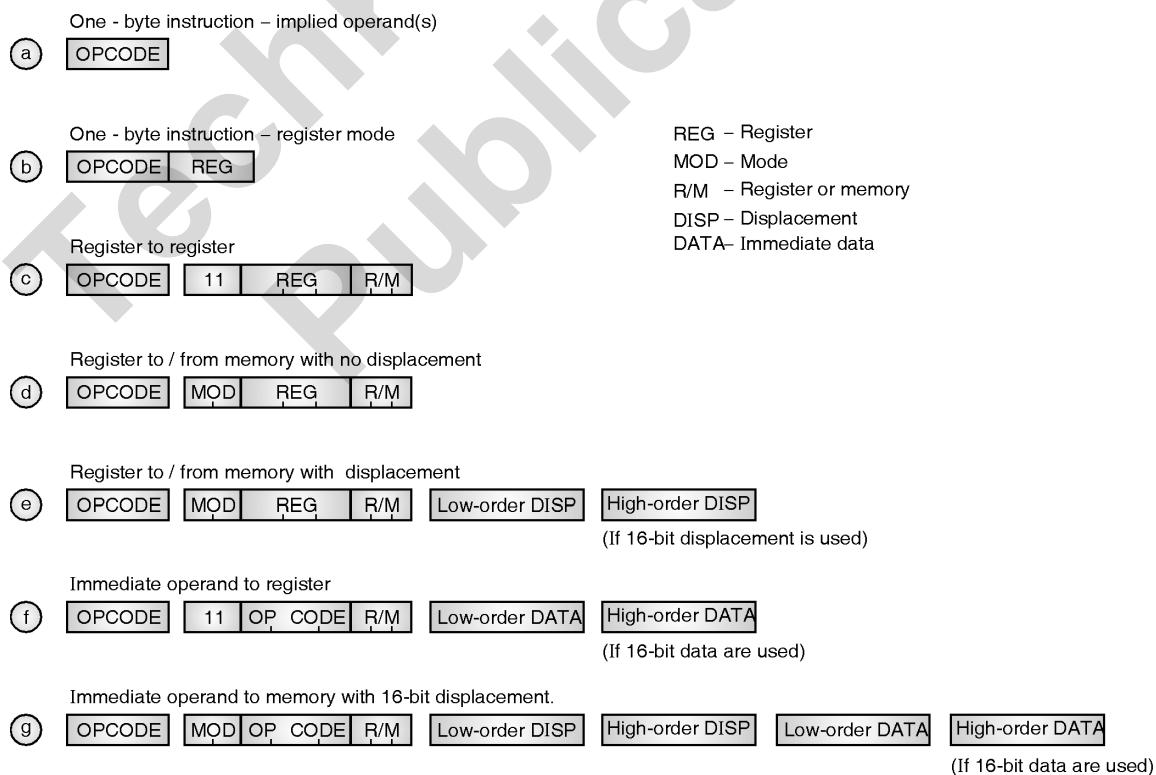


Fig. 10.1.1 : Summary of 8085 instruction format

- To remember these formats, I will give you only a single format, from that we get these different formats refer Fig. 10.1.2.
- As shown in Fig. 10.1.2, the first six bits of a multibits instruction generally contains an opcoding that identifies the basic instruction type i.e. ADD, XOR etc.

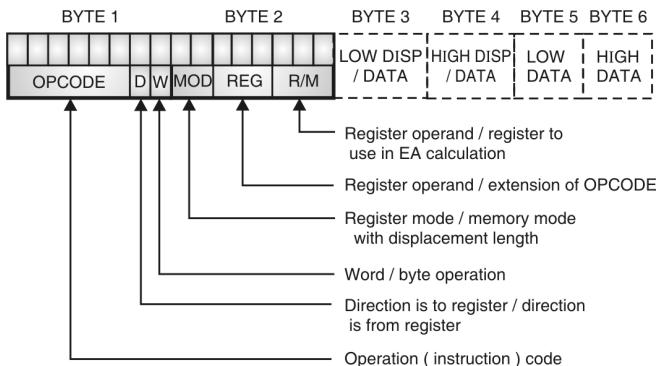


Fig. 10.1.2 : Instruction format

- The following bit, called the D field, generally specifies the **direction** of the operation.

D = 1 means instruction source is specified in REG field.

D = 0 means instruction destination is specified in REG field.

The next following bit is **W**. This bit identifies between bits and word operation.

W = 0 Instruction operates on bits data.

= 1 Instruction operates on word data.

- Refer Fig. 10.1.2, if you observe in some case, in 2nd bits we have MOD, OPCODING and R/M, for some of the cases we have MOD, REG and R/M. First we will concentrate on OPCODING bits in 2nd bits of instruction format. This field is 3 bit wide. Under that we have three single bit fields, S, V and Z.

S bit :

- An 8 bit, 2's complement number can be extended to a 12 bit 2's complement number by letting all of the bits in high order bits equal the MSB in low order byte. This is referred to as **sign extension**.
- S bit is used in conjunction with W to indicate sign extension of **Immediate fields** in arithmetic instructions.

S = 0 No sign extension

= 1 Sign extended 8 bit immediate data to 12 bits if W = 1.

Therefore for 8 bit operation : S = W = 0

12 bit operation with a 12 bit immediate operand : S = 0, W = 1

12 bit operation with a sign extended 8 bit immediate operand : S = W = 1

V bit :

- Used by shift and rotate, to determine single and variable - bit shifts and rotate.

V = 0 shift/rotate count is one

= 1 shift/rotate count is specified in CL register.

Z bit :

- This bit is used as a compare bit with zero flag in conditional repeat (REP) and loop instructions.

Z = 0 Repeat/loop while zero flag is clear.

= 1 Repeat/loop while zero flag is set.

Refer Table 10.1.1, it summarizes all 5 bits used in OPCODING field.

Table 10.1.1 : 5 bits used in OPCODING field

Field	Value	Function
S	0	No sign extension
	1	Sign extend 8-bit immediate data to 12 bits if W = 1
W	0	Instruction operates on bits data
	1	Instruction operates on word data
Field	Value	Function
V	0	Shift/rotate count is one
	1	Shift/rotate count is specified in CL register
Z	0	Repeat/loop while zero flag is clear
	1	Repeat/loop while zero flag is set

- Now concentrate on MOD, R/M and REG field in 2nd bits of instruction format. The second bits of the instruction usually identifies the instruction's operands.

MOD :

- The mode (MOD) field indicates whether one of the operands is in memory or whether both operands are register. Table 10.1.2 shows MOD field encoding, this field is of size 2 bits.

Table 10.1.2 : MOD field ENCODING

CODE	EXPLANATION
0 0	Memory mode, no displacement follows *
0 1	Memory mode, 8 bit displacement follows



CODE	EXPLANATION	
1 0	Memory mode, 12 bit displacement follows	
1 1	Register mode (No displacement)	

- * Except when R/M = 110, then 12 bit displacement follows. As seen MOD is basically concerned with displacement i.e. 8 bit or 12 bit or no displacement.

REG :

- The Register (REG) field identifies a register that is one of the instruction operands. REG field depends upon W bit. Table 10.1.3 shows the selection of register(s) depending upon W bit.

Table 10.1.3 : REG (Register) field encoding

REG	W = 0	W = 1
0 0 0 1 0 0 1	AL	AX
0 0 1 0 0 1	CL	CX
0 1 0 0 1	DL	DX
0 1 1 1	BL	BX
1 0 0 1	AH	SP
1 0 0 1	CH	BP
1 1 0	DH	SI
1 1 1	BH	DI

- When W = 0, all 8 bit registers are selected, whereas for W = 1 all 12 bit registers are selected.
- Thus in a number of instructions and mainly in immediate to memory variety, REG is used as an extension of the OPCODING to identify the type of operation i.e. 8 bit or 12 bit.

R/M : Register or memory : This field is of 4 bits. The meaning of R/M bits changes depending upon mode (MOD) field.

- At this stage we have general clear idea about these three fields, now, we will take some cases.

Case I : Register to register transfer

- In this operation, data movement is within the register either 8 bit or 12 bit. As mentioned in this operation REG field identifies ONE of the instruction operands. **What about another instruction operand ?** It is specified by R/M, W and MOD bits. Refer Table 10.1.4.

Table 10.1.4 : R/M field encoding when MOD = 11 (binary)

MOD = 11 (binary)		
R/M	W = 0	W = 1
0 0 0 1 0 0 1	AL	AX
0 0 1 0 0 1	CL	CX

MOD = 11 (binary)		
R/M	W = 0	W = 1
0 1 0 0 1	DL	DX
0 1 1 1	BL	BX
1 0 0 1	AH	SP
1 0 0 1	CH	BP
1 1 0	DH	SI
1 1 1	BH	DI

- You will find that Table 10.1.3 matches with Table 10.1.4. Secondly when W = 0, you can select ONLY 8 bit source and destination operand. When W = 1, you can select ONLY 12 bit source and destination operand.

Following are the example instructions those are not correct or valid :

MOV AH, CX ; Moving 12 bit to 8 bit (right)

MOV DX, BL ; Moving 8 bit to 12 bit (right)

- Thus any such combination with other register(s) is INVALID.

Case II : Memory MODE (8 bit/12 bit or No displacement)

- When MOD selects memory mode (MOD = 00 or 01 or 10), then data transfer is register to/from memory.
- In that case R/M field indicates how the effective address of the memory operand is to be calculated. Now refer Table 10.1.5, it depicts EA calculation.

Table 10.1.5 : R/M field encoding when MOD = 00/01/10

MOD = 11		EFFECTIVE ADDRESS CALCULATION				
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

D8 = 8 bit displacement D16 = 12 bit displacement



- REG field in this case, as usual identify the register that is one of the instruction operand.

10.2 Instruction Format and 0-1-2-3 Address Formats :

10.2.1 Instruction Formats :

- The Control Unit and the ALU (Arithmetic and Logic Unit) along with some registers constitute the Central Processing Unit.
- Fig. 10.2.1 shows the basic components of the computer and their interconnection. Also the internal components of the CPU are shown in the Fig. 10.2.1.
- The computer consists of three basic components namely the CPU, memory and I/O devices connected with each other via the buses.
- Input devices are required to give the instructions and data to the system. The output devices are used to give the output devices.
- The instructions and the data given by the input device are to be stored, and for storage we require memory.

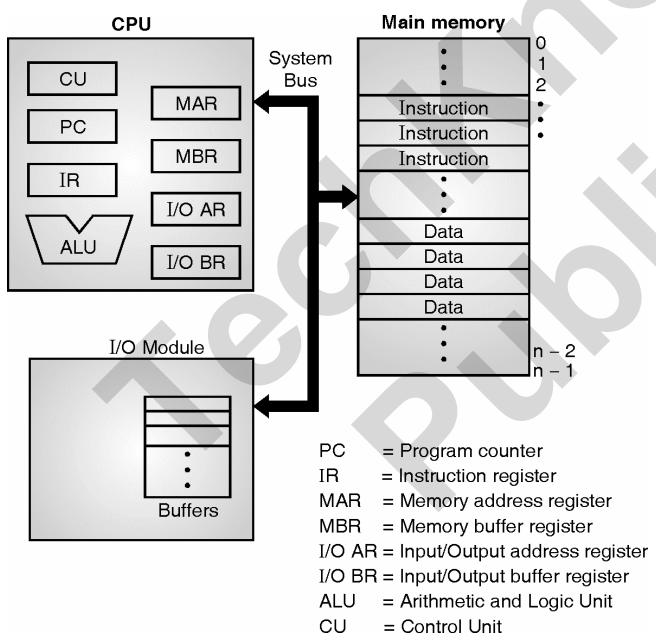


Fig. 10.2.1 : Basic Components of the computer and the CPU

- The memory module receives address and control signals (like read, write, timing etc) and also receives and sends data.
- The memory is assumed to be of ' n ' locations with the addresses from 0 to $n - 1$.
- CPU module reads instruction and data, writes the data and also sends the control signals. It also receives interrupts from the I/O devices and acts accordingly.

- The CPU has the control unit that provides control signals to all the resources inside and outside the CPU. The CPU also has the ALU that performs the arithmetic and logical operations.
- The CPU also has some registers as seen in the Fig. 10.2.1. The register PC (Program Counter) is a register that always points to the next instruction to be executed. Hence the word "Program" in the name.
- It has to increment always to point to the next instruction, which is automatically done. Hence the word "Counter" in the name.
- The MAR (Memory Address Register) is used to store the address to be provided to the memory.
- And MBR (Memory Buffer Register) is used to store the data to be given or taken from the memory. Similarly for I/O devices we have IOAR and IOBR.
- Another register named as IR (Instruction Register) is used to store the instruction to be executed by the CPU.
- The use of these registers will be further seen in the next section named as Instruction Cycle.

10.2.2 Instruction Word Format - Number of Addresses :

Elements of an Instruction :

- Operation code (Opcode) is that part of the instruction which gives the code for the operation to be performed.
 - Source Operand reference or address 1, gives the reference of the data on which the operation is to be performed. This address could be a register, memory or an input device.
 - Source Operand reference or address 2, gives the reference of the second data on which the operation is to be performed. This address could again be a register, memory or an input device.
 - Result Operand reference gives the reference where the result after performing operation is to be stored. The result could be stored in the register, memory or given to an output device.
 - An instruction may have only one address with the other two fixed, or may have two addresses with one of the source operand address as the result operand address. Hence the instruction can have one, two or three addresses.
- Fig. 10.2.2 shows an example of a simple instruction format with one and two addresses.



Opcode	Operand Address 1	
(a) Single address instruction format		
Opcode	Operand Address 1	Operand Address 2
(b) Two address instruction format		

Fig. 10.2.2 : Instruction Word Formats

- Three address, One address and Zero address instructions.

Zero address instructions :

PUSH A ; ToS <- A
 PUSH B ; ToS <- B
 ADD ; ToS <- A + B
 PUSH C ; ToS <- C
 PUSH D ; ToS <- D
 ADD ; ToS <- C + D
 MUL ; ToS <- (A + B)*(C + D)
 PUSH E ; ToS <- E
 PUSH F ; ToS <- F
 SUB ; ToS <- (E - F)
 DIV ; ToS <- (A + B)*(C + D)/(E - F)
 POP X ; M[X] <- ToS

One address instructions :

LOAD A ; AC <- M[A]
 ADD B ; AC <- AC + M[B]
 STORE P ; M[P] <- AC
 LOAD C ; AC <- M[C]
 ADD D ; AC <- AC + M[D]
 MUL P ; AC <- AC * M[P]
 i.e. AC <- (A + B) * (B + C)
 STORE P ; M[P] <- AC
 LOAD E ; AC <- M[E]
 SUB F ; AC <- AC - M[F]
 STORE Q ; M[Q] <- AC
 LOAD P ; AC <- M[P]
 DIV Q ; AC <- AC / M[Q]
 i.e. AC <- (A + B)*(C + D)/(E - F)
 STORE X ; M[X] <- X
 i.e. X <- (A + B)*(C + D)/(E - F)

Accumulator type one address format :

LOAD A ; AC <- M[A]

MUL B ; AC <- AC * M[B]
 STORE P ; M[P] <- AC
 LOAD C ; AC <- M[C]
 MUL D ; AC <- AC * M[D]
 SUB E ; AC <- AC - M[E]
 ADD P ; AC <- AC + M[P]
 i.e. AC <- (A * B) + (C * D - E)
 STORE P ; M[P] <- AC
 LOAD A ; AC <- M[A]
 ADD B ; AC <- AC + M[B]
 STORE Q ; M[Q] <- AC
 LOAD P ; AC <- M[P]
 DIV Q ; AC <- AC / M[Q]
 i.e. AC <- ((A * B) + (C * D - E)) / (A + B)
 STORE X ; M[X] <- X I.E. X <- ((A * B)
 + (C * D - E)) / (A + B)

Three address instructions :

((A * B) + (C * D - E)) / (A + B)
 ADD R1,A,B ; R1 <- M[A] + M[B]
 MUL R2,C,D ; R2 <- M[C] * M[D]
 SUB R2,R2,E ; R2 <- R2 - M[E]
 ADD R1,R1,R2 ; R1 <- R1 + R2 i.e. R1 < ((A * B)
 + (C * D - E))
 ADD R2,A,B ; R2 <- A + B
 DIV X,R1,R2 ; M[X] <- R1 / R2 i.e. X < ((A * B)
 + (C * D - E)) / (A + B)

10.3 Addressing Modes :

- We have seen in the previous section, that an instruction has various components among the source operand reference.
- For the source operand reference we have various methods to give the same.
- Based on these different methods we have different addressing modes. In other words, addressing modes are the different methods (modes) to select (address) the operand for an instruction.
- The different addressing modes are :
 1. Immediate
 2. Direct
 3. Indirect
 4. Register
 5. Register Indirect
 6. Displacement (Indexed)
 7. Relative
 8. Stack

1. Immediate addressing mode :

- In this case the operand is part of instruction. The operand is in the immediate next location of the opcode, hence the name.
- For example ADD AX, 0005H. This instruction adds the contents of register AX with the value 5. In this case 5 is an operand and is given in the instruction itself. AX register is also called as the accumulator register.
- The advantage of this addressing mode is that it is fast.
- The disadvantage is that it has a limited range.
- Fig. 10.3.1 shows the structure of an instruction and operand access technique for the Immediate addressing mode.



Fig. 10.3.1 : Immediate addressing mode

2. Direct addressing mode :

- In this case the address field contains memory address of the operand.
- For example ADD AX,[0005H]. This instruction adds the contents of memory location 0005H to accumulator. The operand is taken from the memory location specified in the instruction.
- In this case there is only a single memory reference to access data.
- The advantage is that there are no calculations to work out effective address.
- The disadvantage is that this addressing mode can be used for a limited address space.
- Fig. 10.3.2 shows the structure of the instruction and operand access technique for the direct addressing mode.

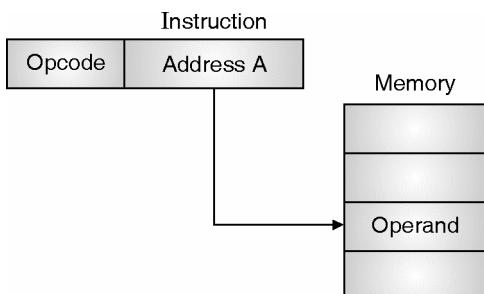


Fig. 10.3.2 : Direct Addressing mode

3. Indirect addressing mode :

- In this case a memory location has the address of the operand in another memory location i.e. a memory

operand is pointed to by address field contains the address of (pointer to) the operand.

- For example ADD AX, [[1000]]. This instruction adds the contents of memory location pointed to by contents of memory location 1000, with the contents of accumulator and stores the result in accumulator.
- The disadvantage is that this method is slower as multiple memory locations are to be accessed to get the operand.
- Fig. 10.3.3 shows the structure of an instruction and operand access technique for the indirect addressing mode.

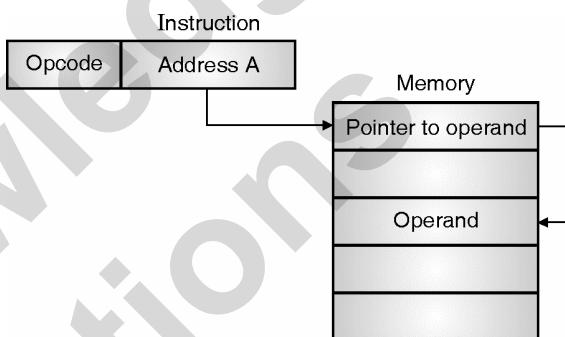


Fig. 10.3.3

4. Register addressing mode :

- In this case the operand is held in the register named in operand address field.
- There are limited numbers of registers; hence very small address field is required. Hence shorter instructions and faster instruction fetch.
- Also another advantage is that there is no memory access. We can say that this is the best addressing mode in terms of time required to access the operand.
- The only disadvantage of this method is the limited number of registers available in most of the processors.
- For example, ADD AX, BX. This instruction adds the contents of the registers AX with the contents of registers BX and the result is stored in the register AX.
- Fig. 10.3.4 shows the instruction structure and the method of access for the register addressing mode.
- As already discussed the major advantage of this addressing mode is that it has very fast execution but limited address space.
- Thus the processors that have multiple registers helps in improving the performance of the processor.

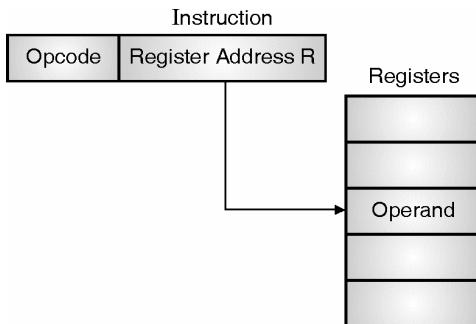


Fig. 10.3.4 : Register addressing mode

5. Register indirect addressing :

- In this case the operand memory address is pointed to by contents of register R.
- It requires one less memory access than indirect addressing mode as seen in above point number 3.
- Fig. 10.3.5 shows the structure of the instruction and the way to access the operand for the register indirect addressing mode.

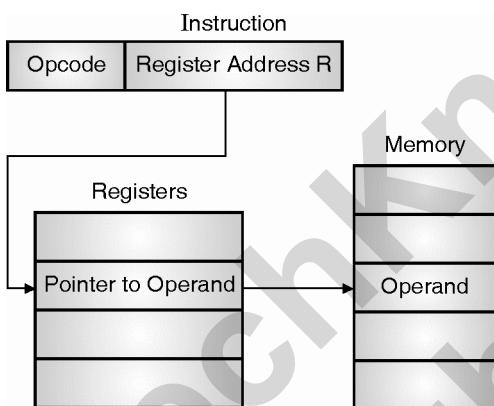


Fig. 10.3.5 : Register Indirect addressing mode

- As seen in Fig. 10.3.5, the instruction contains a register address field that selects the register that has the memory address of the operand to be accessed.

6. Displacement addressing mode :

- In this type of addressing mode, there are two address fields that hold the base address and the displacement. The base address is held by the register address given in the instruction.
- The register address field in the instruction select one of the register that has the address. This address is to be added with the displacement and hence giving the address of the memory location that has the operand.
- The major advantage of this type of this addressing mode is that the pointer need not be continuously updated; instead the displacement can be given with the instruction itself.

- The disadvantage is that there is extra computation required for the address calculation.
- The structure of the instruction and the access method for the displacement addressing mode is shown in Fig. 10.3.6.

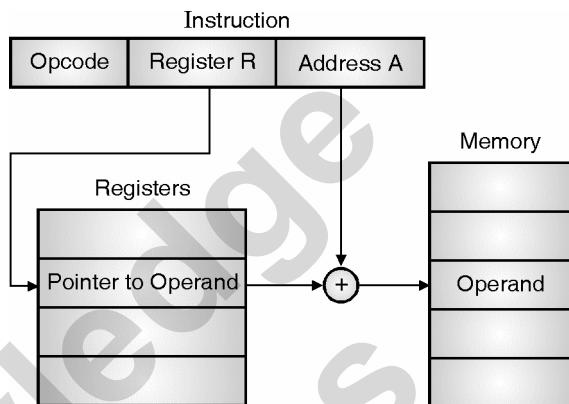


Fig. 10.3.6 : Displacement addressing mode

7. Relative addressing mode :

- It is a version of displacement addressing where the register is the program counter, PC. Program counter is a register that always points to the next instruction to be executed by the processor and hence tells the processor about which next instruction it has to execute.
- It is used for branching or transfer of control instructions. In this case the current value of the program counter is updated with the relative address specified in the instruction. This relative address is added to the current value of program counter and hence the name as relative addressing mode.

8. Stack addressing mode :

- This addressing mode is used to access the data from the top of the stack.
- It uses PUSH and POP instructions to access the stack. In this case the operand is implicitly on top of stack.
- For example, POP AX. ; Pop top two items bytes from the top of stack.

10.3.1 Examples on Addressing Modes :

Ex. 10.3.1 : An instruction is stored at location 300 with its address field at location 301, the address field has the value 400. A processor register R1 contains value 200. Evaluate effective address if addressing mode of instruction is :

- Direct
- Immediate
- Relative
- Register indirect
- Index with R1 as index register

Soln. :

(a) Direct :

In this case the instruction has the address field as 400, hence the effective address is 400.

(b) Immediate :

In this case the instruction has the address field as 400, hence the operand itself is 400. This operand is stored in the immediate next location of the instruction. Since the instruction is stored at location 300, the operand is at location 301.

(c) Relative :

In this case the instruction has the address field as 400, which will be added with the register R1's value i.e. 200 and hence the effective address will be 600.

(d) Register indirect :

In this case the register provides the address of the operand. Since the register R1 has a value 200, the effective address in this case will be also 200.

(e) Index with R1 as index register :

In this case the address field i.e. 400 will have an address that will be added with the value of the register R1 which has 200. Hence the effective address in this case will be the address at location 400 + the value of register R1 i.e. 200.

Ex. 10.3.2 : A two address instruction is stored in memory at an address designated with the symbol W. The address field of the instruction (stored at $W+1$) is designated by Y. The operand used during execution of instruction is stored at address symbolized Z. An index register contains value X. State how Z is calculated from other address if addressing mode of instruction is :

- (a) Direct (b) Indirect (c) Relative
- (d) Indexed

Soln. :

(a) Direct :

In this case the instruction has the address field as Y, hence the effective address is Y. Thus $Z = Y$.

(b) Indirect :

In this case the instruction has the address field as Y, hence the operand is at the address which is stored at location Y, i.e. the address field at $W + 1$, that has the

value Y, is actually the address of the address of operand. Thus $Z = [Y]$.

(c) Relative :

In this case the instruction has the address field as Y, which will be added with the value of program counter. Thus $Z = PC + Y$.

(d) Indexed :

In this case the address field i.e. Y will have an address that will be added with the value of the register R1 which has X. Hence the effective address in this case will be the address at location $Y + \text{the value of register R1}$ i.e. X. Thus $Z = [Y] + X$

10.4 Instruction Set of 8085 :

The instruction set of 8085/8088 is divided into number of groups, of functionally related instructions.

Different groups are :

1. Data transfer group.
 2. Arithmetic group.
 3. Bit manipulation group.
 4. String instruction group.
 5. Program transfer instruction group.
 6. Process control instruction group.
- Graphical presentation of different groups is as shown in Fig. 10.4.1.

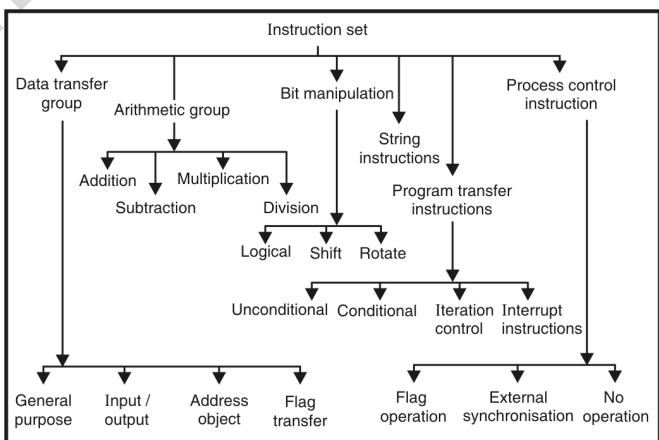


Fig. 10.4.1

Now we will start with instruction set. The information presented is from the point of view of utility to the assembly language programmer. The information given is :

1. Mnemonic (Syntax of the instruction)
2. Algorithm
3. Operation of the instruction
4. Examples.



- While giving you above information some typical symbols/labels are used. I feel that you should know the significance and meaning of those labels.
- Refer Table 10.4.1, which provides, for instruction coding format, different IDENTIFIER where it is used and explanation of the same.

Table 10.4.1 : Key to instruction coding formats

Identifier	Used In	Explanation
Destination	data transfer, bit manipulation	A register or memory location that may contain data operated on by the instruction, and which receives (is replaced by) the result of the operation.
Source	data transfer, arithmetic, bit manipulation	A register, memory location or immediate value that is used in the operation, but is not altered by the instruction.
source-table	XLAT	Name of memory translation table addressed by register BX.
Target	JMP, CALL	A label to which control is to be transferred directly, or a register or memory location whose content is the address of the location to which control is to be transferred indirectly.
short-label	Conditional transfer, iteration control	A label to which control is to be conditionally transferred; must lie within - 128 to + 127 bytes of the first bits of the next instruction.
Accumulator	IN, OUT	Register AX for word transfers, AL for bytes.
Port	IN, OUT	An I/P port number; specified as an immediate value of 0-255, or register DX (which contains port number in range 0-64k).

Identifier	Used In	Explanation
source-string	String operations	Name of a string in memory that is addressed by register SI; used only to identify string as bits or word and specify segment override, if any. This string is used in the operation, but is not altered.
Dest-string	String operations	Name of string in memory that is addressed by register DI; used only to identify string as bits or word. This string receives (is replaced by) the result of the operation.
Count	Shifts, rotates	Specifies number of bits to shift or rotate; written as immediate value 1 or register CL (which contains the count in the range 0-255).
interrupt-type	INT	Immediate value of 0-255 identifying interrupt pointer number.
optional-pop-value	RET	Number of bytes (0-64k, ordinarily an even number) to discard from stack.
external-OPCODE	ESC	Immediate value (0-63) that is encoded in the instruction for use by an external processor.

We also have some labels, used in operand types. Refer Table 10.4.2 to understand meaning of the same.

Table 10.4.2 : Key to operand types

Identifier	Explanation
(no operands)	No operands are written
Register	An 8-or 32-bit general register



Identifier	Explanation
Reg 16	A 32-bit general register
Seg-reg	A segment register
Accumulator	Register AX or AL
Immediate	A constant in the range 0-FFFFH
immed8	A constant in the range 0-FFH
Memory	An 8- or 32-bit memory location ⁽¹⁾
Mem8	An 8-bit memory location ⁽¹⁾
Mem16	A 32-bit memory location ⁽¹⁾
source-table	Name of 156-bits translate table
source-string	Name of string addressed by register SI
dest-string	Name of string addressed by register DI
DX	Register DX
short-label	A label within – 128 to + 127 bytes of the end of the instruction
Near-label	A label in current coding segment
Far-label	A label in another coding segment
Near-proc	A procedure in current coding segment
far-proc	A procedure in another coding segment
Memptr16	A word containing the offset of the location in the current coding segment to which control is to be transferred ⁽¹⁾
Memptr32	A double word containing the offset and the segment base address of the location in another coding segment to which control is to be transferred ⁽¹⁾
regptr16	A 32-bit general register containing the offset of the location in the current coding segment to which control is to be transferred
Repeat	A string instruction repeat prefix.

(1) Any addressing mode-direct, register indirect, based relative, based indexed or relative based indexed-may be used.

10.5 Reduced Instruction Set Computer Principles :

- Sun SPARC is a RISC processor; while all the processors studied till now in this book were CISC processors. Hence before studying the details of Sun SPARC processor, we will see how RISC processors are different than CISC and also the special features of RISC processor.

10.5.1 RISC Versus CISC :

Sr. No.	Properties	RISC	CISC
1.	Number of Instructions	Less	More
2.	Addressing Modes	Less	More
3.	Instruction Formats	Less	More
4.	Instruction Size	Fixed	Variable
5.	Control Unit	Hardwired	Micro-programmed
6.	Number of Bus Cycles to execute an instruction	Single CPU cycle (for 80% Instructions)	Multiple CPU cycles
7.	Control Logic And Decoding Subsystem	Simple	Complex
8.	Pipelining	Huge no. of stages of Pipelining	Difficulty in efficient implementation
9.	Design time and Probability of Design Errors	Smaller time and less probable	Long time and Significant probability
10.	Complexity of Compiler	Simpler	More complex and the results of "optimization" may not be most efficient and the fastest machine language code
11.	HLL instructions	Supported	Not Supported

10.5.2 RISC Properties :

A RISC system must satisfy the following properties :

1. Single-cycle execution of all (or at least 80 percent) instructions.
2. Single-word standard length of all instructions.
3. Small number of instructions (<= 128).
4. Small number of instruction formats (<= 4).
5. Small number of addressing modes (<= 4).
6. Memory access possible by load and store instructions only.

7. All operations, except load and store, are register to register i.e. within the CPU.
8. It must have a hardwired control unit.
9. It must also have a relatively large (at least 32) general-purpose CPU register file.

10.5.3 Register Window :

1. Since there are a huge number of registers in a RISC processor, it can be useful in saving the latency period during procedure call. Parameter passing in is possible by the **register window**. This policy also allows reasonable HLL support in RISC designs.
2. The register file is subdivided into groups of registers, called **register windows**.
3. A certain group of 'i' registers, suppose R_0 to $R_{(i-1)}$, are designated as **global registers**. The global registers are accessible to all procedures running on the system at all the times.
4. On the other hand, each procedure is assigned a separate window within the register file, which is not accessible to other procedures.
5. The window base (first register within the window) is pointed to by a field called **current window pointer (CWP)** located in the CPU's **status register (SR)**.

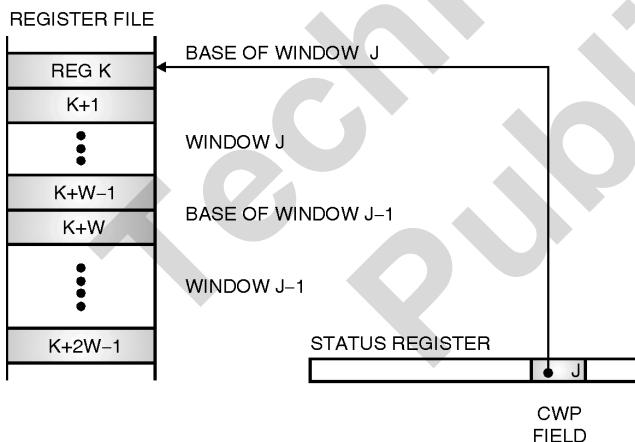


Fig. 10.5.1 : Simple Non-overlapping Register Window

6. If the currently running procedure is assigned the register window J, hence taking up registers K, K+1, ..., K+W-1 (where W is the number of registers per window), the CWP contains the value J, hence pointing to the base of window J. If the next procedure to execute takes up window J+1, the value in the CWP field will be incremented accordingly to point to J+1.
7. Register windowing can work more efficiently for parameter passing between calling and called

procedures by partial overlapping of the windows. The last N registers of window J will be the first N registers of window J+1.

8. If the procedure taking up window J calls a procedure, which will be assigned the next window J+1, it can pass N parameters to the called procedure by placing their values into registers (K+W-N) to (K+W-1). The same registers will be automatically be available to the called procedure without any further movement of data.

10.5.4 Miscellaneous Features or Advantages of RISC Systems :

1. **HLL support :**
 - (a) The support for High Level Language (HLL) features is mandatory in the design of any computing system.
 - (b) The procedure call-return and parameters passing is the most time-consuming operation in typical HLL programs.
 - (c) HLL support is provided in RISC machines by supporting efficiently the handling of local variables, constants, and procedure calls, while leaving less frequent HLL operations to instructions sequences and subroutines.
 - (d) One of the mechanisms supporting the handling of procedures, and their parameter passing in particular, is the feature of the **register window**, discussed in section 10.5.3.

2. Implementation of register windows :

- (a) A CISC processor's control unit takes up a large percentage of the chip area, leaving very little space for other subsystems and basically not permitting a large register file, needed for an efficient implementation of windowing.
- (b) A RISC processor's control unit makes up a much smaller percentage of the chip area, yielding the necessary space for a large register file.

- (c) And hence implementation of register windowing (that requires huge number of registers) is possible in RISC processors

3. Pipelining :

- (a) Pipelining was used on various CISC systems even before the RISC approach became popular.
- (b) But, a streamlined RISC can handle pipelines more efficiently.

**4. Delayed branch :**

- (a) The problem occurs in a system where instructions are prefetched, right after a branch.
- (b) If the branch is conditional, and the condition is not satisfied, then the next instruction, which was prefetched, is executed, and since no branch is to be performed, no time is lost.
- (c) But, if the branch condition is satisfied, or the branch is unconditional, the next prefetched instruction is to be flushed and other instruction pointed to by the branch address is to be fetched in its place. The time required to prefetch the flushed instruction is wasted.
- (d) Such waste of time is solved by using the **delayed branch approach**.
- (e) In this approach, the instructions are reshuffled such that the operation does not change the result.
- (f) A successful branch is assumed and the execution of the branch is delayed until the already prefetched instructions are executed. Hence, no time is lost and there is no change in the intended program operation.
- (g) But the compiler has to take care that the instructions followed by the branch instructions are to be executed irrespective to the branch to be taken or not.

5. Scoreboarding :

- (a) Another problem in instruction pipelines is that of data dependency.
- (b) The data in some register put by instruction 1 may be required by instruction 2; and before the value in the register is available, the instruction 2 may be ready for execution yielding a possibly incorrect result.
- (c) A method used to solve this problem is called as **scoreboarding**.
- (d) A special CPU control register i.e. the **scoreboard register**, is required for this purpose.
- (e) If there are 32 registers, a scoreboard register 32-bit long will be required; each of its bit represents one of the 32 CPU registers.
- (f) If register 'i' is involved as a destination in the execution of instruction 1, bit 'i' in the scoreboard register is set; and as long as bit 'i' is set, any subsequent instruction in the pipeline will be prevented from using Ri in any way until bit 'i' is cleared.
- (g) This instruction will now be executed as soon as the execution of the instruction, which caused bit 'i' to be set, is completed.

6. Dual split cache :

Another feature, implemented in not only a CISC but also RISC systems, is separated data and code caches, or split cache.

7. Instruction Level Parallelism (ILP) :

Superscalar and superpipelined designs are also mostly implemented in a RISC design.

8. VLSI realization :

- (a) The chip area, dedicated to the realization of the control unit, is considerably less. Therefore, on a RISC VLSI chip, there is more area available for other features (cache, FPU, part of the main memory, memory management unit, I/O ports, etc).
- (b) As a result of the considerable reduction of the control area, a large number of CPU registers can fit on-chip.
- (c) By reducing the control area on the VLSI chip and filling the area by numerous identical registers, the **regularization factor** (which is defined as, ratio of chip area utilized by other features to the chip space required by control unit) of the chip increases. The higher the regularization factor, the lower is the VLSI design cost.

9. The computing speed :

- (a) A simpler and smaller control unit in RISC requires fewer gates. This results in shorter propagation paths for control unit signals, decreasing the delay time for control signals and hence yielding a faster operation.
- (b) A significantly reduced number of instructions, formats, and addressing modes results in a simpler and smaller decoding system, resulting in faster decoding operation.
- (c) A hardwire-controlled system can hence be implemented, with a reduced control unit that will in general be faster than a microprogrammed control unit.
- (d) A relatively large CPU register file also reduces CPU-memory traffic to fetch and store data operands.
- (e) A large register set can also be used to store parameters to be passed from a calling to a called procedure, to store the information of a process that was interrupted by another.

10. Design cost and reliability considerations :

- (a) It takes a shorter time to complete the design of a RISC control unit, because of the smaller instruction set, fixed



instruction length and less instruction formats. Thus contributing to the reduction in the overall design cost.

- (b) A simpler and smaller control unit will obviously have a reduced number of design errors and, therefore, a higher reliability.

10.5.5 RISC Shortcomings :

1. Since a RISC has a small number of instructions, a number of functions, performed on CISC's by a single instruction, will need more instructions on RISC. Hence, RISC's code will be longer.
2. More memory will have to be allocated for RISC programs, and the instruction accesses between the memory and the CPU will be increased.

10.5.6 ON-Chip Register File Versus Cache Evaluation :

Modern CISC have on-chip cache to compensate the registers of RISC processors. But, let us evaluate the advantages of register file over on-chip cache.

Sr. No.	CACHE	CPU Register file
1.	Addressed as locations in memory-long addresses.	Separate register addressing-short addresses.
2.	Has to be tens of Kbytes to be effective.	About 128 registers (of 4-bytes each, i.e. 512 bytes) will have significant effect on performance.
3.	Information loaded in units of lines (blocks).	Information can be loaded individually to each register.
4.	Slower access (Effective address calculation, virtual to physical address translation).	Faster access.
5.	Information loaded based on prefetch and replacement policies.	The user can load any information at any time.
6.	Inaccessible by the user.	Fully accessible by the user.
7.	Possibility of a miss.	No miss is possible.

10.6 Polling and Interrupts :

- Whenever more than one I/O devices are connected to a microprocessor based system, any one of the I/O devices may ask service at any time.
- There are two methods in which the microprocessor can service these I/O devices. One method is to use the **polling routine**, while the other method employs **interrupt**.
- In the polling routine the microprocessor checks whether any of the I/O devices is requesting for service.
- The polling routine is a simple program that keeps a check for the occurrences of interrupt.
- For e.g. : Let us assume that our polling routine is servicing I/O ports 1, 2, 3,.....8. The polling routine will check the status of the I/O ports in a proper sequence.
- The polling routine will first transfer the status of the I/O port 1 to the accumulator. It then checks the contents of accumulator to determine if the service request bit is set.
- If the bit is set then I/O port 1 service routine is called, otherwise the polling routine will move forward to check if port 2 is requesting service.
- On completion of the service to port 1, the polling routine will test port 2. The process is repeated till all the 8 ports are tested and all the I/O ports those are demanding service are processed.
- On completion of the polling routine, the microprocessor will resume with the execution of the program. Fig. 10.6.1 shows the sequence for polling routine.
- The polling routine has priorities assigned to the different I/O devices. Once the routine begins port 1 will always be checked first, then port 2 and so on.
- Another way that allows the microprocessor stop with the execution of the program and give service to the I/O devices is **interrupt**.
- It is an external asynchronous input that informs the microprocessor to complete the instruction that it is currently executing and fetch a new routine in order to offer service to the I/O device. Once the I/O device is serviced, the microprocessor will continue with the execution of its normal program.

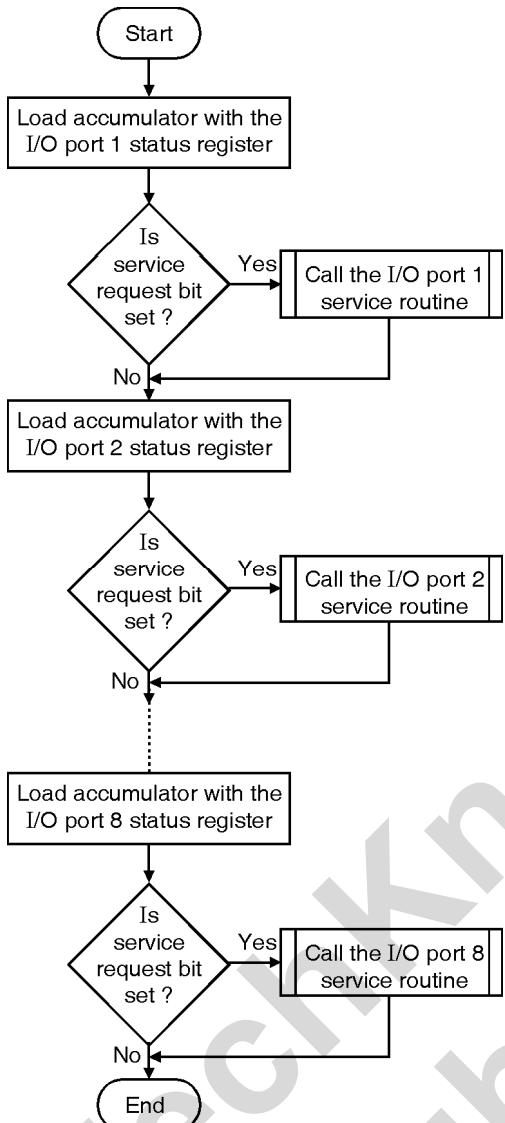


Fig. 10.6.1 : Polling sequence

10.7 Pipeline Processing :

- A processor has many resources like the ALU, buses, registers, etc. An attempt to utilize all these attributes to their fullest or continuously can be achieved by pipelining.
- In a pipelined system the instructions flow through the processor as if the processor was a pipe.
- The instructions move from one stage to another to accomplish the assigned operation. Hence at most of the times each unit of the processor is busy handling one or the other instruction, making the attribute of the processor being used continuously.
- This chapter deals with advanced pipelining and superscalar design in the processor development. We

will go through the concepts and design issues of the linear and non-linear pipelining. We will also discuss collision-free scheduling techniques for performing dynamic functions.

- Techniques to design instruction pipelines, arithmetic pipelines are also discussed.

10.7.1 Non-Pipelined System Versus Two Stage Pipelining :

- In a non-pipelined system, the processor fetches an instruction from memory, decodes it to determine what the instruction was, read the instructions inputs from the register file, performs the computation required by the instruction and writes the result back into the register file. This approach is also called as un pipelined approach.
- The problem with this approach is that, the hardware needed to perform each of these steps (Instruction fetch, instruction decode, register read, instruction execution and register write-back) is different and most of the hardware is idle at any given moment. Waiting for the other parts of the processor to complete their part of executing an instruction.
- Pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.
- Two stage pipelining includes two stages i.e. Fetch instruction and Execute instruction.
- These two operations are performed for one instruction and the next instruction overlapping i.e. when the first instruction is being executed the next is fetched and when this instruction is executed the next is fetched and so on.
- This method of execution the instructions in pipeline speeds up the processor operation.
- This also makes sure that all the units of the processor are busy operating and none of them is starving.
- Thus with the help of pipelining the operation speed of the processor increases i.e. more the number of pipeline stages, faster becomes the processor, but complex in design.
- Two stage instruction pipeline stage is as shown in Fig. 10.7.1(a).

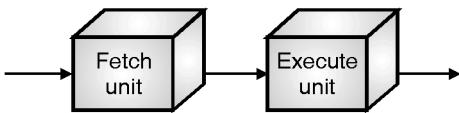
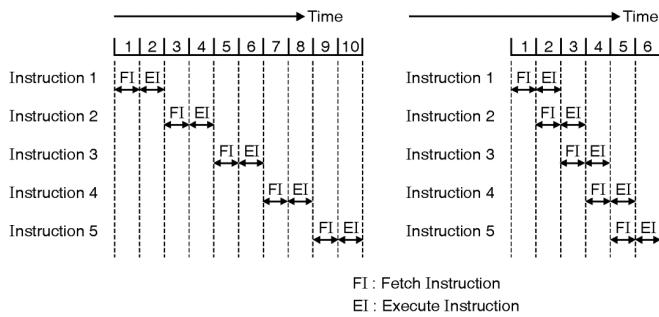


Fig. 10.7.1(a) : Two stage pipeline architecture



(b) Timing diagram of execution of instructions in non-pipelined system.

(c) Timing diagram of execution of instructions in pipelined systems.

Fig. 10.7.1

- In case of a system without pipelining the time required for executing a set of instructions is much more than the time required executing the same set of instructions in a pipelined system.
- The comparison of the execution of five instructions in a system with and without pipeline is shown in Figs. 10.7.1(b) and 10.7.1(c).
- You will notice that the time required for executing five instructions on a non-pipelined system is 10 clock pulses while that on two stage pipelined processor is 6 clock pulses. Thus the number off clock pulses required in two stage processor will always be $x/2 + 1$, where 'x' is the number of clock pulses in non-pipelined instructions and '2' is the number of stages.
- If we increase the number of instructions, we can make the expression as $x/2$ (since '1' is negligible for huge number of instructions) clock pulses for a two stage pipelined processor, wherein 'x' clock pulses in case of non-pipelined processor.
- If we try to generalize this expression, we can write it as x/n , where x is number of clock pulses required for non-pipelined instructions and 'n' is the number of stages of a pipelined processor.
- Thus we can say that the speed-up achieved by a pipelined processor can be maximum 'n' times of the non-pipelined processor.

10.7.2 Basic Pipelined Datapath and Control for a Six Stage CPU Instruction Pipeline :

- The flowchart given in Fig. 10.7.2 gives an instruction flow through the six stage pipelined processor.

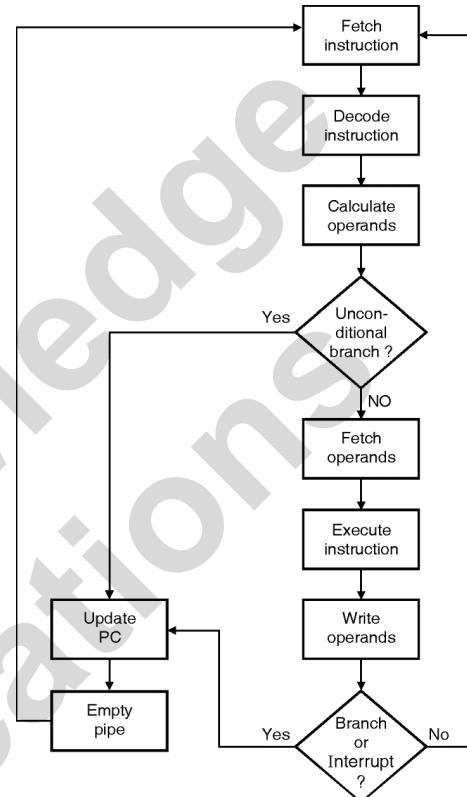
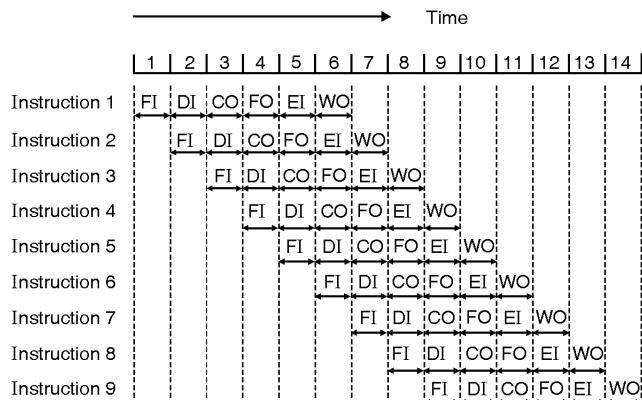


Fig. 10.7.2 : Six stage pipeline flowchart

This can be shown on a time scale as in Fig. 10.7.3.



Where,

FI : Fetch Instruction

DI : Decode instruction

CO : Calculate Operand address

FO : Fetch Operand

EI : Execute Instruction

WO : Write Operand result

Fig. 10.7.3 : Six stage pipelined processor timing diagram



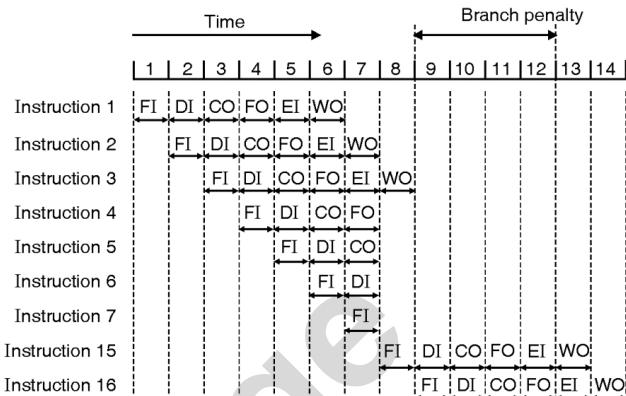
- Pipelining is termed as overlapped parallelism as in case of pipelining the instructions are overlapped.
- The execution of the instructions is overlapped in such a manner that there are several instructions under different process in the pipelined processor as shown in Fig. 10.7.3.
- As shown in the Fig. 10.7.3, for e.g. during the 6th clock pulse, there are six instructions in the processor.

Instruction :

1. has its result being written back, instruction
 2. is being executed, instruction
 3. has its operands being fetched, instruction
 4. has the address of operands being calculated, instruction
 5. is being decoded and instruction
 6. is being fetched.
- This shows how pipelining is a overlap parallelism of instructions.
 - This six stage pipeline system can be implemented with six units as shown in Fig. 10.7.4.


Fig. 10.7.4 : Six stage pipelined architecture

- In pipelining, when a branch instruction is executed the system causes a huge waste of time i.e. processor units starving. The instructions in the pipeline are the sequential instructions.
- If a branching instruction is given the next instruction to be executed is not the sequential one, instead it is the instruction at target instruction.
- Hence the sequential instructions in the pipeline are to be cleared and instructions from target are to be fetched. Clearing the sequential instructions from the pipeline is called as flushing of the pipeline.
- These problems are discussed in detail in section 10.8. Also the solutions to the same are discussed in that section. This is as shown in the timing diagram in Fig. 10.7.5.


Fig. 10.7.5 : Branch in a six-stage pipeline

Ex. 10.7.1 : Draw the space-time diagram for a six-segment pipeline showing the time it takes to process eight tasks.

Soln. :

Clock cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	
Segment :	1	T1	T2	T3	T4	T5	T6	T7	T8	-	-	-	-	
	2	-	T1	T2	T3	T4	T5	T6	T7	T8	-	-	-	
	3	-	-	T1	T2	T3	T4	T5	T6	T7	T8	-	-	
	4	-	-	-	T1	T2	T3	T4	T5	T6	T7	T8	-	
	5	-	-	-	-	T1	T2	T3	T4	T5	T6	T7	T8	
	6	-	-	-	-	-	T1	T2	T3	T4	T5	T6	T7	T8

It takes 13 clock cycles to process 8 tasks.

10.7.3 Linear Pipeline Processors :

- In a linear pipelined processor there are n stages connected linearly to perform different operations. These may perform different operations to execute an instruction, perform arithmetic operations or memory access operations.
- In a linear pipelined processor with suppose n stages, the partially processed instructions are passed from stage i to stage i + 1, where i varies from 1 to k - 1. The linear pipeline can either be a synchronous system or asynchronous system.

10.7.3.1 Asynchronous and Synchronous Linear Pipelining :

- In case of an asynchronous linear pipeline system, there is a set of handshaking signals between the two stages. Whenever a stage (say stage i) completes its operation, it places the result on the input lines of next stage (i.e. stage i + 1) and enables the ready (or strobe) signal.
- The next stage (i.e. stage i + 1) on completing its operation, accepts the data from its input lines and

indicates this to the previous stage (i.e. stage i) by giving an acknowledgement signal. On this, the stage which had placed the data (i.e. stage i), also checks its input if it has previous stage (i.e. stage $i - 1$) completed its operation and is ready with the result.

- It also repeats the same process as explained with stage i and $i + 1$. This can be explained as shown in the Fig. 10.7.6.

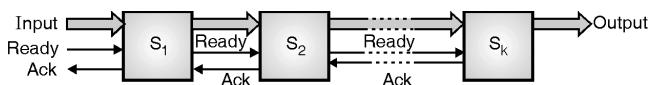


Fig. 10.7.6 : Asynchronous linear pipelining system

- Hence the asynchronous linear pipelined system will have variable throughput rate and will experience different amount of delay at each stage.
- In case of a synchronous linear pipelined system the stages are separated by latches. Whenever a stage completes its part of operation it stores the result in the latch.
- The clock signal is synchronously given to all the latches, such that on reception of the clock signal each stage takes the output of the latch connected to its input. This system is shown in Fig. 10.7.7.

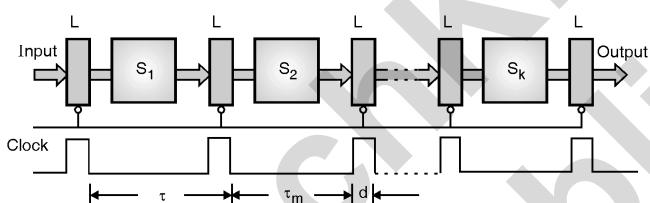


Fig. 10.7.7 : Synchronous linear pipelining system

- The latches are infact master-slave flipflops. The time required by each stage is expected to be equal; and it is this time that determines the clock period as well as the speed of the pipelined system.
- The utilization of the stages or the utilization pattern of stages in a synchronous pipeline can be represented by the reservation table.

		Time(clock cycles)								
		1	2	3	4					
Stages	S ₁	X								
	S ₂		X							
	S ₃			X						
	S ₄				X					
	Captions :									
S_i = stage i										
L = Latch										
τ = Clock period										
τ_m = Maximum stage delay										
d = Latch delay										
Ack = Acknowledge signal										

Fig. 10.7.8 : Reservation table of a synchronous linear pipeline

- The reservation table follows a diagonal line for synchronous linear pipeline as shown in Fig. 10.7.8.
- Reservation table is a space-time diagram showing streamline pattern. Hence as seen in Fig. 10.7.8, for an n-stage pipeline, n clock pulses are required to execute the instruction.
- Once the pipeline is filled up completely, the processor completes one instruction execution every clock pulse.

10.7.3.2 Clocking and Timing Control :

- The clock cycle, τ as shown in Fig. 10.7.7, can be calculated as discussed below. Let τ_i be the delay time for stage S_i . Hence the **clock cycle** time can be given as :
$$\tau = \max [\tau_i]_1^K + d = \tau_m + d$$
where τ_m the maximum stage delay and d is, as shown in the Fig. 10.7.7, the 'on' period of the clock pulse.
- The data from each stage is latched in the master flipflop of latch register during the rising edge and given to the slave flipflop during the falling edge. In fact, $\tau_m \gg d$; hence we can say that, $\tau \approx \tau_m$.
- Hence the pipeline frequency can be given as $= 1 / \tau$. This frequency f , is also termed as the throughput of the system as it gives the time required for one instruction to come out of the pipeline.
- The actual **throughput** of the pipeline may be less than the maximum throughput given by f , which may be because of more than one clock pulses, may be required for the initiation of successive instructions.
- The initiation of successive instructions may take more clock pulses because of their data or control dependency.
- The clock pulse at each stage is expected to arrive simultaneously. But, because of the time delay of the path, different stages get the pulse at different time offset s; this problem is referred to as **clock skewing**.
- Assuming the shortest logic path to get the clock at a delay of t_{min} and the longest logic path to get the clock pulse at delay of t_{max} ; to avoid this problem the $\tau_m \geq t_{max} + s$ and $d \leq t_{min} - s$. Thus the clock period with skew is :
$$d + t_{max} + s \leq \tau \leq \tau_m + t_{min} - s$$
- Hence in ideal case, $s = 0$, $t_{max} = \tau_m$ and $t_{min} = d$. Hence, even with the clock skewing $\tau = \tau_m + d$

10.7.3.3 Speedup, Efficiency and Throughput :

- A linear pipeline of k stages will take $k + (n - 1)$ clock cycles to execute n instructions; the first instruction will take k clock cycles and the remaining $n - 1$ instructions will take one clock cycle each (assuming there are no dependency of the instructions). Hence with the clock cycle width being τ , the total time required to execute these n instructions will be

$$T_k = \tau [k + (n - 1)] \quad \dots(10.7.1)$$

- An equivalent non-pipelined system the time required to execute n instructions will be

$$T_1 = nk\tau \quad \dots(10.7.2)$$

- Thus the speedup factor of a k -stage pipelined system can be given as :

$$\begin{aligned} S_k &= \frac{T_1}{T_k} = \frac{n k \tau}{k \tau + (n - 1) \tau} \\ &= \frac{nk}{k + (n - 1)} \quad \dots(10.7.3) \end{aligned}$$

- Hence as the number of instructions n increases, S_k tends to k . Thus, ' **k stage pipeline processor can be at most ' k ' times faster than the corresponding non-pipelined processor.**' The speed up factor as a function of the number of instructions is shown in the Fig. 10.7.9.

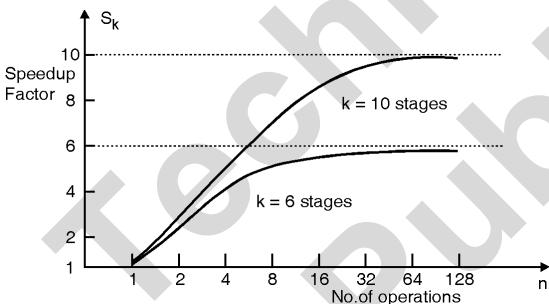


Fig. 10.7.9 : Relationship of speedup factor with the no. of operations

- Also there is a limit to the number of stages. As the number of stages increases the delay and skewing increases.
- Hence the finest pipelining or micro-pipelining that has the subdivision of the stages at almost gate level should consider this optimal number of stages.
- Most of the systems have the number of stages varying from 2 to 15. Very few systems are designed to have the number of stages beyond 10.
- This is because the increase in the number of stages should take care that the PCR (performance to cost ratio) has also increased. But beyond a particular

number of stages, termed k_0 (optimal no. of stages), the PCR starts reducing as shown in the Fig. 10.7.10.

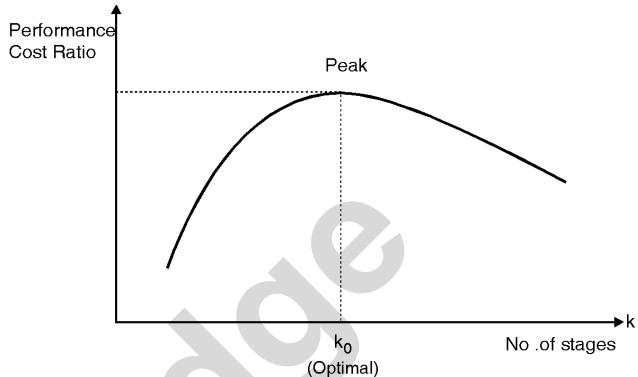


Fig. 10.7.10 : Graph of PCR vs. k

- To execute a program on sequential non-pipelined system, say the time required is ' t '. Thus to execute this program on a k -stage pipeline with equal flow-through the time required = $\frac{t}{k} + d$, where d is latch delay.

$$\text{Hence, } f = \frac{1}{\frac{t}{k} + d}$$

The performance / cost ratio (PCR) can be defined as :

$$\begin{aligned} \text{PCR} &= \frac{f}{(c + kh)} \\ &= \frac{1}{\left(\frac{t}{k} + d\right)(c + kh)} \quad \dots(10.7.4) \end{aligned}$$

where h is the cost of each latch, c is the cost of all logic gates of a stage, d is the latch delay, f is the pipeline frequency, k is the number of stages and t is the flow-through delay of each stage. The optimal number of stages can be given as

$$k_0 = \sqrt{\frac{t \cdot c}{d \cdot h}} \quad \dots(10.7.5)$$

The efficiency of such a system is defined as (using Equation (10.7.3))

$$E_k = \frac{S_k}{k} = \frac{n}{k + (n - 1)} \quad \dots(10.7.6)$$

The pipeline throughput H_k is defined as number of operations performed per unit time.

$$\begin{aligned} H_k &= \frac{\text{Efficiency}}{\text{One unit time } (\tau)} \\ &= \frac{n}{[k + (n - 1)] \tau} \\ &= \frac{nf}{k + (n - 1)} \quad \dots(10.7.7) \end{aligned}$$

Hence the maximum throughput as discussed earlier will be equal to f , when the number of instructions n tends to ∞ .

Ex. 10.7.2 : Consider the execution of a program of 15,000 instructions by a linear pipeline processor with a clock rate of 25 MHz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-sequence executions are ignored.

- Calculate the speedup factor in using this pipeline to execute the program as compared with the use of an equivalent non-pipelined processor with an equal amount of flow-through delay.
- What are the efficiency and throughput of this pipelined processor ?

Soln. :

Given : $n = 15000$, $f = 25 \text{ MHz}$, $k = 5$

$$\begin{aligned} \text{Speed up factor } (S_k) &= \frac{nk}{k + (n - 1)} \\ &= \frac{15000 \times 5}{5 + (15000 - 1)} = 4.9986 \end{aligned}$$

$$\text{Efficiency } (E_k) = \frac{S_k}{k} = 0.99973$$

$$\begin{aligned} \text{Throughput } (H_k) &= \frac{E_k}{k} = f E_k = 25 \text{ MHz} \times 0.99973 \\ &= 24.9933 \text{ MIPS} \end{aligned}$$

Ex. 10.7.3 : A non-pipeline system takes 50 ns to process a task. The same task can be processed in a six stage pipeline with a clock cycle of 10 ns. Determine the speedup and the efficiency of the pipeline for 100 tasks. What is the maximum speedup and efficiency that can be achieved ?

Soln. :

Given : For the non-pipeline system : $t_n = 50 \text{ ns}$

For the pipeline system : $k = 6$, $t_p = 10 \text{ ns}$

Number of tasks $n = 100$

$$\begin{aligned} \text{Speed up } (S_k) &= \frac{T_1}{T_k} = \frac{nk\tau_1}{k\tau + (n - 1)\tau} \\ &= \frac{100 \times 50}{6 \times 10 + (100 - 1) \times 10} = 4.7619 \end{aligned}$$

Maximum speedup will occur when the number of tasks (n) is very large ($n \gg k$). Hence neglecting the term $k - 1$, we have Max Speedup = $\frac{n\tau_1}{n\tau} = \frac{50}{10} = 5$

$$\text{Efficiency } (E_k) = \frac{S_k}{k} = \frac{4.7619}{6} = 0.7936$$

Considering the max speedup the max efficiency = $\frac{5}{6} = 0.8333$

10.7.4 Non Linear Pipeline Processors :

- A dynamic or multi-function pipeline is called as non-linear pipeline. In a linear pipeline the operations that are being performed are fixed; each stage as a fixed operation.
- But in a non-linear pipeline allows feed forward and feedback connections in addition to the streamline connection. It may also have more than one output i.e. the output need not be from the last stage. An example of three stage non-linear pipeline system is shown in Fig. 10.7.11.

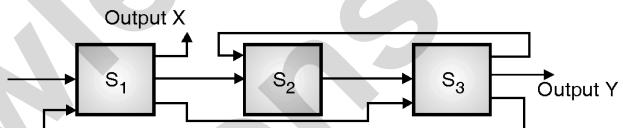


Fig. 10.7.11 : A 3-stage non linear pipeline

- In the Fig. 10.7.11 besides the three stages connected in streamline, there are also some feedback and feedforward connections.
- The feedforward connection is from S_1 to S_3 and feedback connections from S_3 to S_2 and S_3 to S_1 . The reservation table for such connections may be different for different operations.
- There are two examples of different operations say, X and Y , for which the reservation table is shown in Fig. 10.7.11.
- Fig. 10.7.12 shows the requirement of different stages at different times for doing the corresponding operation. For e.g. the function X , has first to be given to S_1 , then S_2 , then S_3 , then S_2 , then S_3 , then S_1 , then S_3 and finally to S_1 which will give the output.

	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

Fig. 10.7.12(a) : Reservation table for function X

	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

Fig. 10.7.12(b) : Reservation table for function Y

- Similarly the function Y is first given to S_1 , then S_3 , then S_2 , then S_3 , then S_1 and finally to S_3 which will give the output.
- The number of columns in a reservation table corresponds to the **evaluation time** of that function. Hence from Fig. 10.7.12, function X has an evaluation time of 8 clock cycles while function Y has an evaluation time of 6 clock cycles.
- A pipeline initiation table consists of different time when the next time the same function can be initiated. The

number of time units (clock cycle) required between the two initiations of a function is called as the **latency period** between them.

- Some valid latencies or latency sequences that do not cause any collision are shown in Fig. 10.7.13. The latencies that do not cause collision are called as **latency sequence** while latencies that cause collision are called as **forbidden latencies**. Examples of forbidden latencies are shown in Fig. 10.7.14.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S_1	X ₁	X ₂				X ₁	X ₂	X ₁	X ₂	X ₃	X ₄				X ₃	X ₄	X ₃	X ₄	X ₅	X ₆	
S_2		X ₁	X ₂	X ₁	X ₂					X ₃	X ₄	X ₃	X ₄							X ₅	...
S_3			X ₁	X ₂	X ₁	X ₂	X ₁	X ₂		X ₃	X ₄	X ₃	X ₄	X ₃	X ₄	X ₄					

(a) Latency cycle (1, 8) = 1, 8, 1, 8, 1, 8, ..., (With an average latency of 4.5)

Cycle repeats

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S_1	X ₁			X ₂		X ₁	X ₃	X ₁	X ₂	X ₄	X ₂	X ₃	X ₅	X ₃	X ₄	X ₆	X ₄	X ₅	X ₇	X ₅	
S_2		X ₁		X ₁	X ₂		X ₂	X ₃		X ₃	X ₄		X ₄	X ₅		X ₅	X ₆		X ₆	X ₇	
S_3			X ₁		X ₁	X ₂	X ₁	X ₂	X ₃	X ₂	X ₃	X ₄	X ₃	X ₄	X ₅	X ₄	X ₅	X ₆	X ₅	X ₆	

(b) Latency cycle (3) = 3, 3, 3, 3, ..., (With an average latency of 3)

Cycle repeats

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S_1	X ₁				X ₁	X ₂	X ₁				X ₂	X ₃	X ₂					X ₃	X ₄	X ₃	
S_2		X ₁	X ₁				X ₂		X ₂				X ₃		X ₃					X ₄	...
S_3			X ₁		X ₁		X ₁		X ₂		X ₂		X ₃		X ₃		X ₃		X ₃		

(c) Latency cycle (6) = 6, 6, 6, 6, ..., (With an average latency of 6)

Fig. 10.7.13 : Example latencies of function X that do not cause collision

	1	2	3	4	5	6	7	8	9	10	11								
Stages	S_1	X ₁		X ₂			X ₃		X ₁	X ₄		X ₁ , X ₂				X ₂ , X ₃			
Stages	S_2		X ₁		X ₁ , X ₂			X ₂ , X ₃				X ₃ , X ₄					X ₄		...
Stages	S_3			X ₁		X ₁ , X ₂			X ₁ , X ₂ , X ₃			X ₂ , X ₃ , X ₄							

(a) Collision with scheduling latency 2

	1	2	3	4	5	6	7	8	9	10	11
Stages	S_1	X ₁				X ₁ , X ₂		X ₁			
Stages	S_2		X ₁		X ₁			X ₂		X ₂	
Stages	S_3			X ₁		X ₁		X ₁ , X ₂		X ₂	

(b) Collision with scheduling latency 5

Fig. 10.7.14 : Example forbidden latencies i.e. latencies that cause collision of function X

- As shown in Fig. 10.7.13, forbidden latencies are 2 and 5. Besides, 4 and 7 are also forbidden latencies. To detect the forbidden latency, you need to check the distance between the two marks on the reservation table in a row.
- For e.g. in case of function X, as shown in Fig. 10.7.12(a), the distance between two marks in S_1 is 5 or 2. Average latency of a latency cycle is defined as the ratio of sum of all latencies to the number of latencies along the cycle. For e.g. (1,8) latency as shown in Fig. 10.7.13(a), has an average latency of $(1+8)/2 = 4.5$.
- The average latency of a constant cycle, i.e. latency cycle that contains only one latency value, is same as the constant value. For e.g. the average latency of the cycle 3 and 6, as shown in Figs. 10.7.13(b) and (c) are 3 and 6 respectively.

10.7.4.1 Collision Free Scheduling or Job Sequencing :

- As seen in non-linear pipelining, we cannot sequence the instructions (job) as in linear pipelining; else there would be collision.
- Hence we need to have optimal job sequencing or scheduling technique. In a non-linear pipelining while scheduling, the main aim is to obtain smallest average latency without any collision.
- This pipeline design theory requires the concepts of collision vectors, state diagrams, single cycles, greedy cycles and minimal average latency (MAL).

1. Collision vectors :

- As seen in the previous section, we can separate the permissible latencies and the forbidden latencies using the reservation table.
- For a reservation table with n columns, the maximum forbidden latency (m) should be $\leq n - 1$. The permissible latency (p), must be as small as possible. An ideal case would be $p = 1$. This smallest latency i.e. $p = 1$ is possible in linear pipelining, but in non-linear pipelining, it is difficult to achieve.
- A **collision vector** is an ' m ' bit binary vector ($C = C_m \ C_{m-1} \dots \ C_2 \ C_1$), that shows the set of permissible and forbidden latency. In a collision vector, a

bit C_i is '1' if the latency i causes a collision, else it is '0'. For e.g., the reservation tables seen in Fig. 10.7.12, will have the collision vectors as $C_x = (1011010)$ and $C_y = (1010)$. Thus for C_x , there is a permissible latency 1, 3 and 6 while forbidden latencies of 7, 5, 4 and 2.

2. State Diagrams :

- From the collision vector, we can make the state diagram for the pipeline.
- The collision vector C_x achieved above is called as the initial collision vector. When loaded in a register and shifted right, each bit at the output corresponds to an increase in latency.
- A '1' at the output indicates collision, while a '0' indicates no collision. A '0' is inserted from the left for every clock cycle. This can be implemented as said by a right shift register and OR gates, as shown in Fig. 10.7.15.

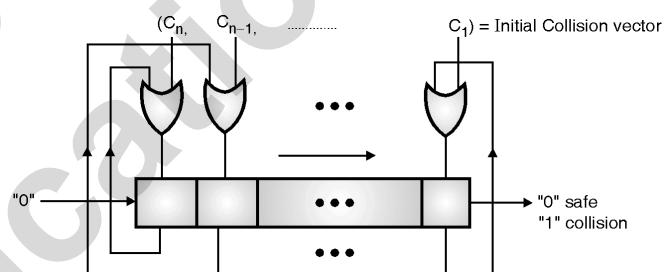
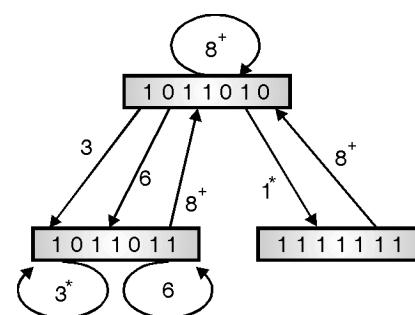


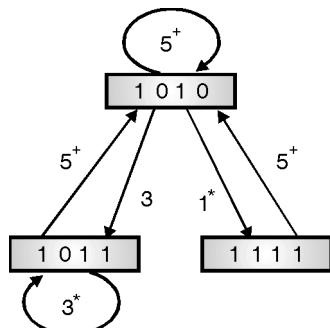
Fig. 10.7.15 : n-bit right shift register for state transition

- The state transition diagram can be constructed using this state register.
- The next state at time $t + p$, where p is the permissible latency and t is some no. of clock pulses, obtained by shifting the register for p times and ORing it with the initial collision vector.
- The state diagrams for the collision vectors C_x and C_y are as shown in Fig. 10.7.16.



(a) State diagram for collision vector C_y

Fig. 10.7.16(Contd...)



(b) State diagram for collision vector C_y
Fig. 10.7.16 : State diagrams for collision

- A transition example can be explained as below. For e.g. the three bit shifts with the initial vector of function X, will result in 0001011; this when ORed with the initial collision vector results in 1011011.
- The state diagram (Refer Fig. 10.7.16(a)) shows this transition for the function X. If the number of shifts is greater than m, the next state is same as the initial collision vector. For e.g. if the number of shifts is 8 or more in Fig. 10.7.16(a), it comes back to the initial collision state. This transition is denoted by 8^+ .

3. Single cycle and Greedy cycle :

- A single cycle is one in which any state appears not more than once. For example for the X function state diagram shown in Fig. 10.7.16(a), the different single cycles are (3), (6), (8), (1,8), (3,8) and (6,8).
- A cycle that travels for more than one time through the same state is a greedy cycle. some greedy cycles in the Fig. 10.7.16(a) are (1,8,3,8), (1,8,6,8), (3,6,3,8,6) etc.

4. Minimal Average Latency (MAL) :

We have already studied the minimal average latency in the previous sections. There are some bounds on this value of MAL. These bounds are listed below :

1. The lower bound of MAL is the maximum number of checkmarks in any row of the reservation table
2. The MAL should be lower than or equal to the latency of any greedy cycle in a reservation table.
3. The upper bound of MAL is equal to the number of 1s in the initial collision vector plus 1.

Ex. 10.7.4 : Consider following pipeline reservation table.

	1	2	3	4
S ₁	X			X
S ₂		X		
S ₃			X	

- What are the forbidden latencies ?
- Draw the state transition diagram.
- List all simple cycles and greedy cycles.
- Determine the optimal constant latency cycle and the minimal average latency.
- Let the pipeline clock period be $\tau = 20$ ns. Determine the throughput of this pipeline.

Soln. :

(a) Collision Vector = (100)

x ₁	x ₂	x ₃	x ₁ x ₄	x ₂ x ₅	x ₃	x ₄	x ₅
	x ₁	x ₂	x ₃	x ₄	x ₅
		x ₁	x ₂	x ₃	x ₄	x ₅	...

Collision with latency 1

x ₁		x ₁ x ₂			x ₂ x ₃			x ₃
	x ₁			x ₂			x ₃	...
		x ₁			x ₂			x ₃ ...

Collision with latency 3

Hence latencies (1) and (3) are forbidden latencies.

(b) State transition shift register

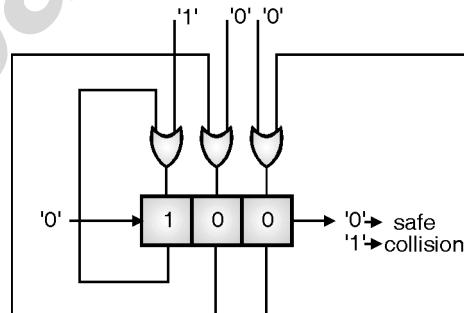


Fig. P. 10.7.4

Using the above shift register, we can generate the following state transition diagram.

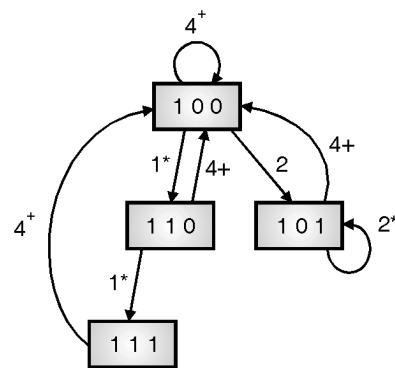


Fig. P. 10.7.4(a)



- (c) As seen in the state transition diagram,
Simple cycles : (2), (4), (1,4), (1,1,4) (2,4) etc.
Greedy cycles : (1,4,2,4) , (1,1,4,2,4) etc.
- (d) Optimal constant latency (2)
∴ Minimum average latency (MAL) = 2
- (e) Throughput

Cycle repeats										
S ₁	x ₁		x ₂	x ₁	x ₃	x ₂	x ₄	x ₃	x ₅	...
S ₂		x ₁		x ₂		x ₃		x ₄		...
S ₃			x ₁		x ₂		x ₃		x ₄	...

As seen in the above table, one instruction is executed every two cycles.

$$\therefore \text{Throughput} = \frac{1}{2} \times f = \frac{1}{2} \times \frac{1}{20 \text{ nsec}} \\ = 25 \text{ MIPS}$$

Ex. 10.7.5 : A non-pipelined processor X has a clock rate of 25 MHz and an average CPI (cycles per instruction) of 4. Processor Y, an improved successor of X, is designed with a five-stage linear instruction pipeline. However, due to latch delay and clock skew effects, the clock rate of Y is only 20 MHz.

- (a) It is a program containing 100 instructions is executed on both processors, what is the speedup of processor Y compared with that of processor X?
(b) Calculate the MIPS rate of each processor.

Soln. :

(a) Program has 100 instruction :

$$\text{i.e. } n = 100$$

Time taken by a non-pipelined (X) processor to execute this program (T_1) = nkt .

$$\text{where } n = \text{number of instruction} = 100 \\ k = \text{number of cycles per instruction} = 4$$

$$\tau = \text{clock width} = \frac{1}{25 \text{ MHz}}$$

$$I_1 = nk\tau = 100 * 4 * \frac{1}{25 \text{ MHz}} = 16 \mu\text{sec.}$$

For a 5-stage pipelined (Y) processor, the time required to execute n-instruction (T_k) = $[k + (n - 1)] \tau$.

$$\text{where } k = 5 \text{ stages}$$

$$n = 100 \text{ instructions}$$

$$\tau = \frac{1}{f} = \frac{1}{20 \text{ MHz}} = 0.05 \mu\text{sec}$$

$$T_k = [5 + (100 - 1)] * 0.05 \mu\text{sec} \\ = 5.2 \mu\text{sec}$$

$$\therefore \text{Speedup} = \frac{T_1}{T_k} = \frac{16 \mu\text{sec}}{5.2 \mu\text{sec}} = 3.07$$

(b) Non-pipelined processor (X) :

Time taken	Instructions
16 μsec	100
1 sec	x

$$\therefore \text{MIPS rate, (x)} = \frac{100 \text{ Instruction}}{16 \mu\text{sec}} \\ = 6.25 \text{ MIPS (Million instructions per second)}$$

Pipelined processor (Y) :

Time taken	Instructions
5.2 μsec	100
1 sec	x

$$\therefore \text{MIPS rate, (x)} = \frac{100 \text{ Instruction}}{5.2 \mu\text{sec}} = 19.23 \text{ MIPS}$$

Ex. 10.7.6 : Consider the following pipeline reservation table

	0	1	2	3	4	5	6	7	8
1	x								x
2		x	x					x	
3				x					
4					x	x			
5							x	x	

1. Determine latencies in the forbidden list F and collision vector C
2. Draw state Transition diagram
3. List all simple cycles and greedy cycles
4. Determine MAL

Soln. :

1. Forbidden latencies F = (1, 5, 6, 8) :

$$\therefore \text{Collision vector C} = (10110001)$$

(Since the total number of tasks are 9, there are 8 bits in collision vector i.e. C₈ to C₁)

2. State transition diagram :

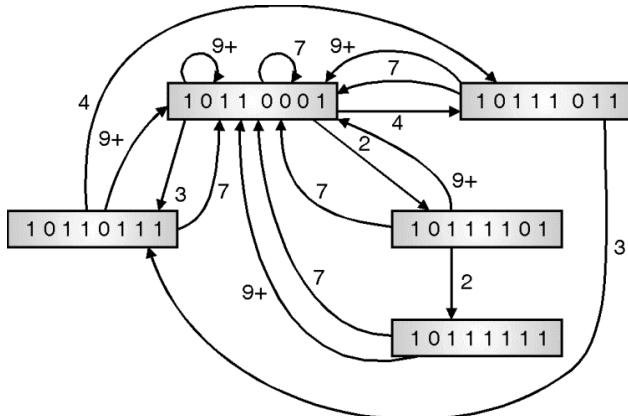


Fig. P. 10.7.6

3. Latency cycles :

(7), (2, 7), (2, 2, 7), (4, 7), (4, 3), (3, 7), (4, 3, 4, 7), (9), (2, 9), (4, 9) (3, 9)

Simple cycles :

(7), (2, 7), (4, 7), (4, 3), (3, 7), (9), (2, 9), (4, 9), (3, 9)

Greedy cycles :

(2, 2, 7), (4, 3, 4, 7)

4. Optimal control latency : (4, 3)

$$\therefore \text{Minimum average latency (MAL)} = \frac{4+3}{2} = 3.5$$

Ex. 10.7.7 : Consider the following pipeline reservation table.

Clock cycle \ Stage	0	1	2	3	4	5	6
S ₁	x		x				x
S ₂				x		x	
S ₃			x		x		

- Determine latencies in Forbidden list F and collision vector C
- Draw the state transistor diagram
- List all simple cycles and greedy cycles
- Determine minimum average latency (MAL)
- For a pipeline clock period $\tau = 20$ ns. Determine maximum throughput of the pipeline.

Soln. :

1. Forbidden latencies = (2, 4, 6)

Collision vector C = (101010)

(Since the total number of tasks are 7, there are 6 bits in the collision vector i.e. C₆ to C₁)

2. State transition diagram

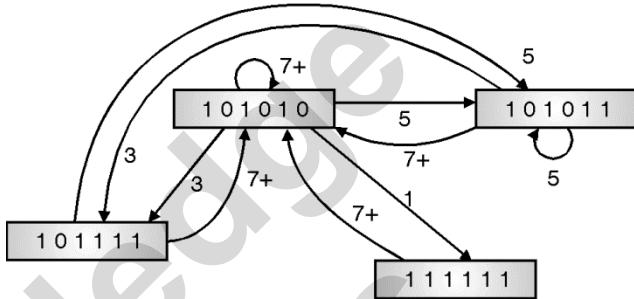


Fig. P. 10.7.7

3. Latency cycles :

(7), (1, 7), (3, 7), (3, 5), (5), (3, 7, 5, 3, 7), (5, 3, 7)

Simple cycles :

(7), (1, 7), (3, 7), (3, 5), (5), (5, 3, 7)

Greedy cycles :

(3, 7, 5, 3, 7)

4. Optimal constant latency : (1, 7) or (3, 5)

$$\therefore \text{Minimal average latency (MAL)} = \frac{1+7}{2} = 4$$

5. Since MAL = 4, 1 instruction will be executed every 4 cycles.

$$\therefore \text{Throughput} = \frac{1}{4} * f = \frac{1}{4} * \frac{1}{20 \text{ nsecs}}$$

$$(\because \text{frequency} = \frac{1}{\tau} = \frac{1}{20 \text{ nsecs}})$$

$$= \frac{1}{70 \text{ nsecs}} = 25 \text{ MIPS}$$

Ex. 10.7.8 : For a unifunction pipeline, the forbidden set of latencies is as given below.

F = {1, 3, 6} with the largest forbidden latency = 6

- Obtain collision vector
- Draw the state diagram
- List all simple and greedy cycles
- Obtain MAL

Soln. :

- Collision vector (C) = (100101)

2.

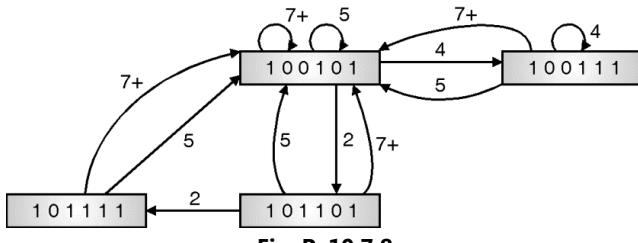


Fig. P. 10.7.8

3. Latencies : (5), (4), (4, 5), (2, 5), (2, 2, 5), (7), (2, 7), (4, 7)

Simple latencies : (5), (4), (4, 5), (2, 5), (7), (2, 7)

Greedy latency : (2, 2, 5)

4. Optimal latency : (2, 2, 5)

$$\therefore \text{Minimal average latency (MAL)} = \frac{2 + 2 + 5}{3} = 3$$

Note : Optimal latency is the latency that gives minimal average i.e. the average as minimum.

- Ex. 10.7.9 :** For the following reservation table, determine collision vector, state transition diagram and MAL.

Clock cycle \ Stage	1	2	3	4	5	6	7	8
S ₁	x					x		
S ₂		x				x	x	
S ₃			x		x			x
S ₄				x				

Also find the throughput for $\tau = 10$ nsec

Soln. :

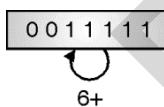


Fig. P. 10.7.9

- Collision vector (C) = (0011111)
 - State transition diagram : (Refer Fig. Ex. 10.7.9)
 - Latencies : (6), (7)
 - Simple latencies : (6), (7)
 - Greedy latencies : NIL
 - Optimal latency : (6)
- Minimal average latency (MAL) = 6
- Since MAL = 6, 1 instruction takes 6 clock pulses
- $$\therefore \text{Throughput} = \frac{1}{6} * f = \frac{1}{6} * \frac{1}{10 \text{ nsec}} = 16.67 \text{ MIPS}$$

- Ex. 10.7.10 :** For the following reservation table, determine collision vector state transition diagram and MAL.

	1	2	3	4	5
S ₁	x			x	
S ₂			x		x
S ₃					
S ₄					

Also find the throughput for $\tau = 25$ nsecs

Soln. :

- Collision vector (C) = (0110)
- State transition diagram : (Refer Fig. Ex. 10.7.10)
- Latencies : (4), (1,4)
- Simple latencies : (4), (1,4)
- Greedy latencies : NIL
- Optimal latency : (1,4)

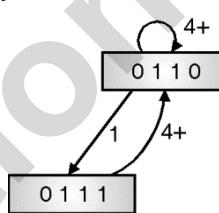


Fig. P. 10.7.10

$$\text{Minimal average latency (MAL)} = \frac{1 + 4}{2} = 2.5$$

- Since MAL = 2.5, 1 instruction takes 2.5 clock pulses
- $$\therefore \text{Throughput} = \frac{1}{2.5} * f = \frac{1}{2.5} * \frac{1}{25 \text{ nsec}} = 16 \text{ MIPS}$$

- Ex. 10.7.11 :** A certain pipeline with the four stages S₁, S₂, S₃ and S₄ is characterized by the following Table P. 10.7.11.

Table P. 10.7.11

	t ₀	t ₁	t ₂	t ₃	t ₄	t ₅	t ₆
S ₁	X					X	
S ₂			X				X
S ₃		X		X			
S ₄			X	X			

- Determine the latencies in the forbidden list F and the collision vector C.
- Determine the minimum constant latency L by checking the forbidden list
- Draw the state diagram for this pipeline and determine MAL.

Soln. :

1. Forbidden latencies = (2, 4, 5)
Collision vector C = (011010)
2. State transition diagram

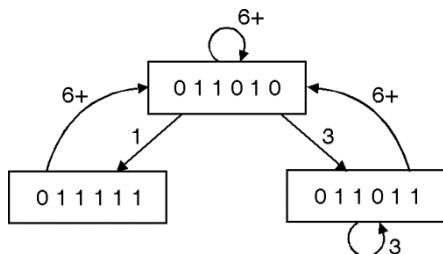


Fig. P. 10.7.11

3. Latency cycle : (6), (1, 6), (3), (3, 6), (3, 3, 6),
(6, 1, 6), (3, 6, 6)
Simple cycles : (6), (1, 6), (3), (3, 6)
Greedy cycles : (3, 3, 6), (6, 1, 6), (3, 6, 6)
4. Optimal constant latency : (3)
Minimum average latency (MAL) = 3

10.8 Instruction Pipelining and Pipelining Stages :

- Instruction pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.
- Generally, the processor fetches an instruction from memory, decodes it to determine what the instruction was, read the instructions inputs from the register file, performs the computation required by the instruction and writes the result back into the register file. This approach is called unpipelined approach.
- The problem with this approach is that, the hardware needed to perform each of these steps (Instruction fetch, instruction decode, register read, instruction execution and register write-back) is different and most of the hardware is idle at any given moment. Waiting for the other parts of the processor to complete their part of executing an instruction.
- Pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.
- Each instruction takes the same amount of time to execute in a pipelined processor as it would in a non-pipelined processor, but the rate at which

instructions can be executed is increased by **Overlapping Instruction Execution**.

- **Latency** is the amount of time that a single operation takes to execute.
- **Throughput** is the rate at which operations get executed.

In a non-pipelined processor,

$$\text{Throughput} = \frac{1}{\text{Latency}}$$

[expressed as operations/second or operations/cycles]

In a pipelined processor,

$$\text{Throughput} > \frac{1}{\text{Latency}}$$

- **Pipelining** : To implement pipelining, designers divide a processor's data path into sections called stages and place pipeline latches between each section.
- As shown in Fig. 10.8.1, at start of each cycle, the pipeline latches read their inputs and copy them to their outputs.

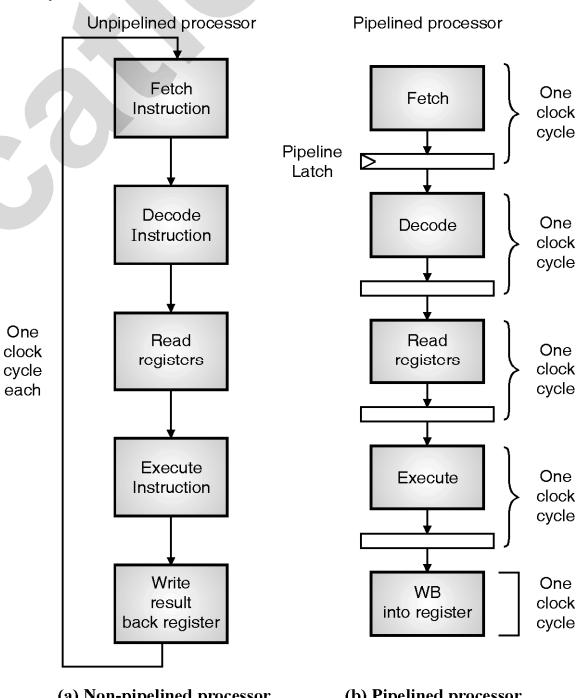


Fig. 10.8.1

- The amount of data path that a signal travels through in one cycle is called a stage of the pipeline.
- A **five-stage pipeline** is shown in Fig. 10.8.1(b).
- Stage 1 : Fetch block
- Stage 2 : Decode block
- Stage 3, 4, 5 are subsequent blocks in execution process.

- Fig. 10.8.2 shows the Instruction flow in a pipelined processor.

	Cycle									
	1	2	3	4	5	6	7	8	9	10
Pipeline stages	(Instruction)									
	IF	I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	I ₈	I ₉
	DE		I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	I ₉
	FO			I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₈
	EX				I ₁	I ₂	I ₃	I ₄	I ₅	I ₇
	WB					I ₁	I ₂	I ₃	I ₄	I ₅

I₁ : executed in 5th cycle
 I₂ : executed in 6th cycle
 I₃ : executed in 7th cycle

Fig. 10.8.2

Cycle time of a pipelined processor is calculated as

$$\text{Cycle time (pipelined)} = \frac{\text{Cycle time (unpipelined)}}{\text{Number of pipeline stages}} + \text{Pipeline latch latency}$$

- As, the number of pipeline stages increases, the pipeline latch latency increases which in turn limits the benefit of dividing a processor into a very large number of pipelining stages.

Ex. 10.8.1 : An unpipelined processor has a cycle time of 25 ns. What is the cycle time of a pipelined version of the processor with 5 evenly divided pipeline stages, if each pipeline latch has a latency of 1 ns?

Soln. :

$$\begin{aligned} \text{Cycle time pipelined} &= \frac{\text{Cycle time unpipelined}}{\text{Number of stages of pipeline}} + \text{Pipeline latch latency} \\ &= \frac{25 \text{ ns}}{5} + 1 \text{ ns} = 6 \text{ ns}. \end{aligned}$$

To find the speedup of the execution process in a pipelined processor,

$$\text{Execution time pipelined} = (K + n - 1) \tau$$

$$\text{Execution time unpipelined} = (K \tau) \times n$$

Where, n = Number of instructions

τ = Time taken for each stage

K = Number of stages in pipeline

Ex. 10.8.2 : If a processor executes 100 instructions in a pipelined (5 stage) processor and unpipelined processor. What is the speedup achieved by

pipelining technique if the time taken for each stage is 20 ns.

Soln. : n = 100 instruction, K = 5, τ = 20 ns

$$\begin{aligned} \text{Execution time pipelined} &= (5 + 100 - 1) \times \tau \\ &= (5 + 99) \times 20 \text{ ns} \\ &= 2080 \text{ ns}. \end{aligned}$$

$$\begin{aligned} \text{Execution time unpipelined} &= (K \tau) n = 5 \times 20 \text{ ns} \times 100 \\ &= 10000 \text{ ns}. \end{aligned}$$

$$\text{Speedup ratio is } = \frac{10000}{2080} = 4.80 \text{ times.}$$

10.9 Pipeline Hazards :

- Pipelining increases processor performance by increasing instruction throughput, because several instructions are overlapped in the pipeline, cycle time can be reduced, increasing the rate at which instructions execute.
- Instruction Hazards (dependencies) occur when instructions read or write registers that are used by other instructions. The type of conflicts are divided into three categories :

- Structural hazards (resource conflicts)
- Data hazards (Data dependency conflicts)
- Branch difficulties (Control hazards)

1. Structural hazards (Resource conflicts) :

- These hazards are caused by access to memory by two instructions at the same time. These conflicts can be slightly resolved by using separate instruction and data memories.
- Structural hazards occur when the processor's hardware is not capable of executing all the instructions in the pipeline simultaneously.
- Structural hazards within a single pipeline are rare on modern processors because the Instruction Set architecture is designed to support pipelining.

2. Data hazards (Data dependency) :

- This hazard arises when an instruction depends on the result of a previous instruction, but this result is not yet available.
- These are divided into four categories :
 - RAW – Hazard (Read after write Hazard)
 - RAR – Hazard (Read after read Hazard)
 - WAW – Hazard (Write after write Hazard)
 - WAR – Hazard (Write after read Hazard)

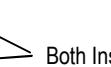


RAR Hazard :

RAR Hazard occurs when two instructions both read from the same register. This hazard does not cause a problem for the processor because reading a register does not change the register's value. Therefore, two instructions that have RAR Hazard can execute on successive cycles.

Example 1 : Instructions having RAR Hazard.

ADD r_1, r_2, r_3
 SUB r_4, r_5, r_3



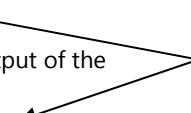
Both Instructions read r_3 , creating RAR

RAW Hazard :

This hazard occurs when an instruction reads a register that was written by a previous instruction. These are also called as **data dependencies** (or) **true dependencies**.

Example 2 : Instructions having RAW - Hazard.

ADD r_1, r_2, r_3
 Subtract reads the output of the
 addition creating RAW hazard
 SUB r_4, r_5, r_1



WAR and WAW are also called as name dependencies.

These hazards occur when the output register of an instruction has been either read or written by a previous instruction.

If the processor executes instructions in the order that they appear in the program and uses the same pipeline for all instructions, WAR and WAW hazards do not cause any problem in execution process.

Example 3 : Instruction having WAR Hazard.

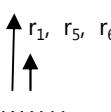
ADD r_1, r_2, r_3
 SUB r_2, r_5, r_6



WAR Hazard

Example 4 : Instructions having WAW Hazard

ADD r_1, r_2, r_3
 SUB r_1, r_5, r_6



WAW Hazard

3. Branch Hazards :

- Branch instructions, particularly conditional branch instructions, create data dependencies between the branch instruction and the previous instruction, fetch stage of the pipeline.
- Since the branch instruction computes the address of the next instruction that the instruction fetch stage

should fetch from, it consumes some time and also some time is required to flush the pipeline and fetch instructions from target location. This time wasted is called as branch penalty.

10.9.1 Methods to Resolve the Data Hazards and Advances in Pipelining :

The methods used to resolve the data hazards are discussed in the following sub sections.

10.9.1.1 Pipeline Stalls :

- The hardware inserts a special instruction called (NOP) i.e. no operation instruction known as a bubble into the flow of execution stage of pipeline to resolve the RAW hazard between two instructions.
- This method is also called as hardware interlocks.
- This approach detects the hazard and maintains the program sequence by introducing delays to resolve the data hazards (RAW).

10.9.1.2 Operand Forwarding (or) Bypassing :

- This technique uses a special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline stages.
- Example of a RAW dependency exists between two instructions, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in the next instructions, it passes the result directly into the ALU input; bypassing the register file.
- This method requires additional hardware paths through MUX (Multiplexers).
- In this case there is a multiplexer between the two stages, wherein the data required by the next instruction is forwarded.
- Let us see this with a program example.
- If we take the following sequence of instructions,

ADD r_1, r_2, r_3
 SUB r_4, r_1, r_3

- We will notice that the destination of instruction 1 is the source for instruction 2.

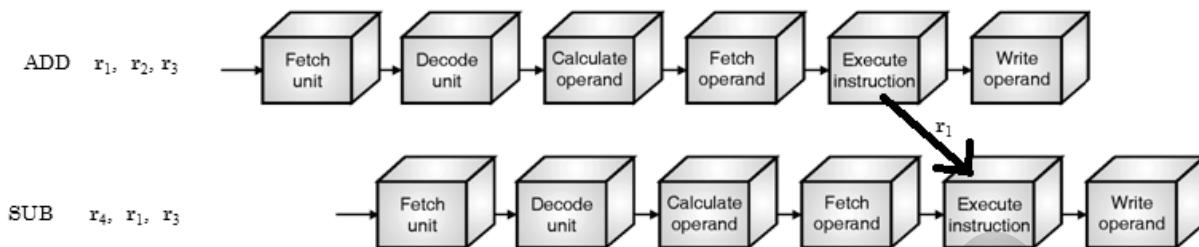


Fig. 10.9.1 : Operand forwarding mechanism in pipelining for resolving a data hazard

- In this case the ALU stage or the execution stage of the pipeline will forward the data to the next instruction as shown in the Fig. 10.9.1. Fig. 10.9.1 assumes that the system is 6 stage pipelined system.
- As shown in Fig. 10.9.1 the data i.e. the value of register r₁ is passed from the first instruction to the second instruction. Actually the value of the register r₁ is updated by the write operand stage of the first instruction. But, before that the same is required by the execute stage of second instruction. Hence the value of this register is passed to the second instruction.

10.9.1.3 Dynamic Instruction Scheduling (or) Out-Of-Order (OOO) Execution :

- This is another interesting and very widely used technique because of the speed up given by it. It is used in Pentium IV processor.
- Here the execution of the instructions of a program is done out-of-order i.e. not in the sequence as the instructions were written by the programmer. As and when the resources of an instruction are available, the execution of that instruction is done. If, for an instruction the resources are not available, it is kept in waiting state and the further instructions whose resources are available will be executed.
- But, you would think that this approach will have a problem. The logic implemented by the programmer will not be followed properly i.e. wrong sequence of instructions will be executed.
- The answer to this is that, although the instructions are executed out-of-order, but the write-back is done in order, and hence the final result of the program is in sequence.
- The compiler is designed in such a way that, while translating from high-level language to machine

language program, it detects the data dependencies and re-orders the instructions.

- If necessary to delay the loading of the conflicting data it inserts no-operation instruction (NOP).

10.9.2 Handling of Branch Instructions to Resolve Control Hazards :

- The methods used to resolve the control hazards are discussed in the following sub sections.

10.9.2.1 Pre-Fetch Target Instruction :

- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed.
- If the branch condition is successful, the pipeline continues from the branch target instructions else sequential instructions are executed.

10.9.2.2 Branch Target Buffer (BTB) :

- The BTB is an associative memory included in the fetch segment of the pipeline.
- Each entry in BTB consists of the address of a previously executed branch instructions and the target instruction for that branch. It also stores the next few instructions after the branch target instructions. When the pipeline decodes a branch instruction, it searches the associative memory BTB for the address of the instruction.
- If it is in the BTB, the instruction is available directly and prefetch continues from the new path.
- If the instruction is not in BTB, pipeline shifts to a new instruction streams and stores the target instruction.

10.9.2.3 Loop Buffer :

- This is like a BTB, but is a high speed register file maintained by the instruction fetch segment of the



pipeline. When a program loop is detected in the program, it is stored in the loop buffer.

- The program loop can be executed directly without having to access memory until loop mode is removed by final branching out.

10.9.2.4 Branch Prediction :

- A pipeline with branch prediction uses additional logic to speculate the outcome of a conditional branch instruction before it is executed.
- The pipeline then begins pre-fetching the instructions stream from the predicted path.
- A correct prediction eliminates the wasted time caused by branch penalties.
- A detailed operation of branch prediction logic will be discussed in the later sections of this chapter.

10.9.2.5 Pipeline Stall (Delayed Branch) :

Compiler detects branch instruction and rearranges the machine language code sequence by inserting useful instructions and rearranges the code sequence to reduce the delays incurred by Branch Instruction.

10.9.2.6 Loop Unrolling Technique :

- This is a very superb solution to handle the stalls due to branching in loops.
- In this case a code which has a loop that has to be executed multiple times, will be actually stored multiple times (or unrolled) so as to remove the need of branching.
- Let us see how this can be implemented with an example.
- If there is a code for adding an array of 5 numbers, the loop can be written as shown in the code below (using processor 8086) :

Label	Instructions
	MOV AX,0000H
	MOV CX,0005H
AGAIN :	ADD AX,[SI]
	INC SI
	DEC CX
	JNZ AGAIN

- This loop can be unrolled to avoid stalling as shown below :

Label	Instructions
	MOV AX,0000H
	ADD AX,[SI]
	INC SI
	ADD AX,[SI]
	INC SI
	ADD AX,[SI]
	INC SI
	ADD AX,[SI]
	INC SI
	ADD AX,[SI]

- Thus you will notice that we have unrolled the loop, and written the loop for the number of times it was to be repeated. This totally removes the pipeline stalls due to the loops.
- The **advantage** clearly seen of this method is that there is no scope for pipeline stall and hence the performance will increase.
- The major **disadvantage** of this method is that the memory required will be more as the loop has to be unrolled and stored in memory. In our example the loop was to be repeated for only 5 times, but if the loop was larger and had to be repeated for say 100 or 1000 times, the memory consumed would be very huge.

10.9.2.7 Software Scheduling or Software Pipelining :

- In case of software pipelining, the iterations of a loop of the source program are continuously initiated at regular intervals, before the earlier iterations complete. Thus taking advantage of the parallelism in data path.
- It can be said that software scheduling, schedules the operations within a loop, such that an iteration of the loop can be pipelined to yield optimal performance.
- The sequences of the instructions before steady state are called as **PROLOG**, while the ones after the steady state are called as **EPILOG**.
- Let us see this with an example. Suppose the source code is


```
for(i=0;i<=n-1;i++)
a[i]=a[i]+10;
```
- When this loop is executed by a processor, the processor will do the following:

```

for(i=0;i<=n-1;i++)
{
    Load a[i];
    Add a[i]+10;
    Store a[i];
}

```

- Here you will notice that the three instructions inside the loop (in each iteration) are the same i.e. each of the three instructions have to operate on the data $a[i]$.
- When this is converted to pipeline, it will look as shown in the Fig. 10.9.2. But the three instructions, one below the other are dependent and hence cannot be pipelined. But the instructions that are circled can be pipelined.

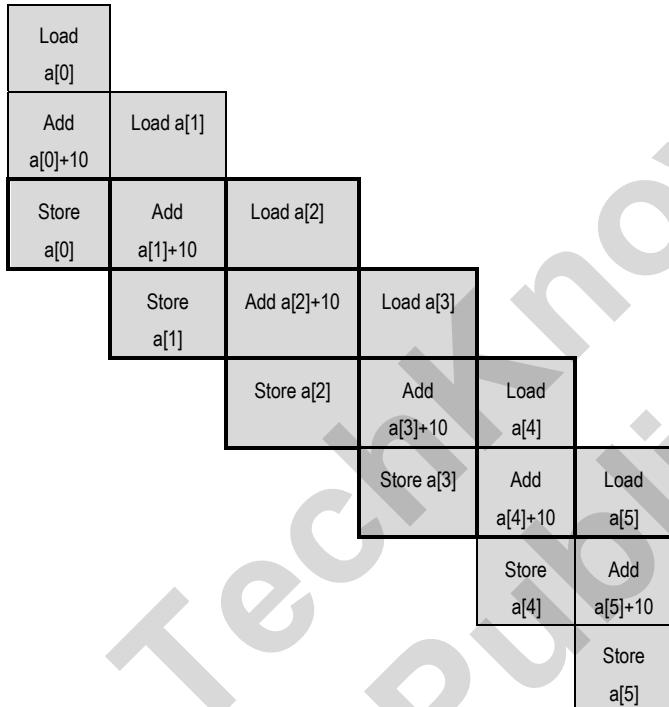


Fig. 10.9.2 : Software pipelining

- You will notice in the Fig. 10.9.2, that the three instructions that are circled are store, add and load. These instructions are always independent i.e. they have different data to operate on.
- For example in the first circle: Store $a[0]$, Add $a[1]+10$ and Load $a[2]$ is performed. Each of these instructions is using different data.
- Thus the code can be changed to the following :

```

Load a[0]
Add a[0]+10
Load a[1]
for(i=2;i<=n-3;i++)

```

```

{
    Store a[i-2];
    Add a[i-1]+10;
    Load a[i]
}

Store a[n-2];
Add a[n-1]+10;
Store a[n-1]

```

- Thus, you will notice inside the for loop i.e. for each iteration, each of the three instructions are working on different data and hence are not dependent on each other and hence allowing pipelining of the three instructions without any hazards.

10.9.2.8 Trace Scheduling :

- In a general pipelining, the instructions are scheduled in sequence. This results in a problem or hazard on a branching instruction as discussed in the previous section. Trace scheduling is a good solution to avoid hazard due to branching. Let us see how this can be implemented.
- In this case the probability of branch to be taken or not taken is found. Based on this the code is written with all instructions in sequence, such that no branching will be required for most of the times according to the probability calculated earlier. This code is called as the trace.
- The other blocks of code are made for less probable cases i.e. if branching is taken. Hence this trace code and the other blocks of code are written, with minimizing branches. Let us see this with a program example. Suppose the source code is:

```

if(a[i]==0)
    a[i]=a[i]+10;
else a[i]=a[i]+1;
    x[i]=x[i]*x[i];

```

- The number is to be squared in the above code. If the number is zero then 10 is to be squared and stored there, while if it is any other number its incremented value is to be squared and stored in the same place. When this is converted to assembly program, it will look as shown below :

Label	Instruction
	Load a[i] into say AL
	Compare AL with 0

Label	Instruction
	If not equal to zero then branch to label over
	Add AL with 10
	Branch to label next
over:	Increment AL
next:	Multiply AL with itself
	Store the result in a[i]

This can be divided into four blocks as shown in the Fig. 10.9.3.

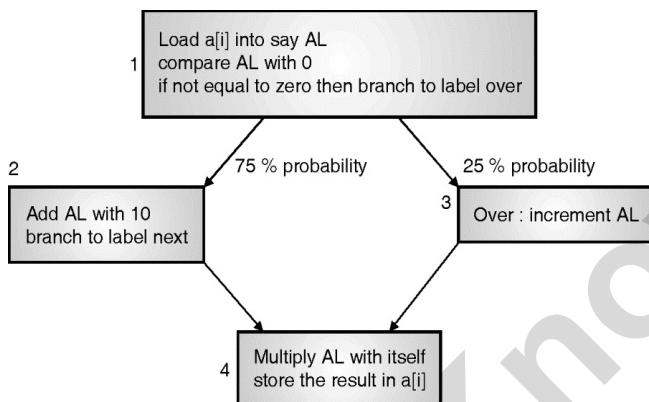


Fig. 10.9.3 : Division of Blocks of the code

- Since the path 1-2-4 is the most probable path, this will make the trace. The path 1-3-4 will be a separate block. This is shown in the Fig. 10.9.4.
- Hence in most of the cases i.e. 75% cases the trace will be executed and hence no branching will be required. Although in 25% cases we will need to branch to Block 1, but there would be only one branching and not multiple branching as required in the previous case.

Trace :

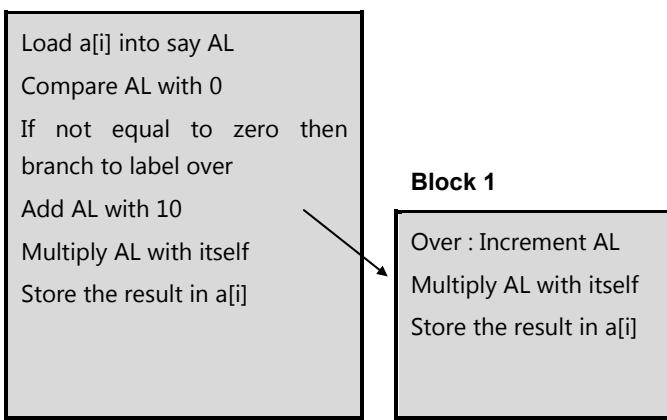


Fig. 10.9.4

10.9.2.9 Predicated Execution :

- This is also a method that removes the branches. Here each instruction has a predicate that decides whether the instruction is to be executed or not. If the predicate is true then the instruction is executed, else it is not executed. The predicate is a condition bit. If the bit is '1' then the instruction is to be executed else it is not to be executed.
- Each instruction has the operands and a predicate. This removes the branching instructions and hence the stall of pipeline.
- An example of predicate instruction is given below, CMOVZ AX, BX, CX.
- This instruction copies the contents of register BX into register AX, if the predicate register CX is zero. Else the contents of BX are not copied into AX.
- Predication mainly implements the if-else statement and hence the branching required for if-else is removed. It can remove the branching required for all the instructions.
- Hence we can say that predication totally removes the need of handling the branches in a pipelined system. The only disadvantage of predication is that the instruction size increases.
- Predication is used in IA-64 processors of Intel, ARM processor.

10.9.2.10 Speculative Loading :

- In this case the data is brought from the memory, well before it is needed.
- The compiler indicates the data that will be required in the later parts of the program and the corresponding data is brought and kept in the processor.
- This removes the latency of memory accesses required for the data to be brought from the memory.
- As the data required later is speculated and brought in advance it is called as speculative loading of data.

10.9.2.11 Register Tagging :

- Register tagging is normally done by a unit called as Reservation Station (RS) in a processor.
- This reservation station is used in order to resolve the data or resource conflicts amongst the multiple instructions entering the processor.



- The operands are made to wait in the reservation station until their data dependencies are resolved.
- A tag is used to identify each reservation station, and the tag unit keeps on monitoring these reservation stations.
- This tag unit also monitors all the registers used currently or the reservation stations. This technique is called as register tagging.
- This mechanism allows to resolve the register conflicts and hence the resultant data hazards.
- The reservation stations can also be used as buffers between the various stages of pipeline in the processor. These stages can work simultaneously once the conflict is resolved.

10.9.3 Branch Prediction :

- Branch prediction foretells the outcome of conditional branch instructions. Excellent branch handling techniques are essential for today's and for future microprocessors. Requirements of high performance branch handling :
- An early determination of the branch outcome (the so-called branch resolution),
- Buffering of the branch target address in a BTAC (Branch Target Address Cache),
- An excellent branch predictor (i.e. branch prediction technique) and speculative execution mechanism.
- Often another branch is predicted while a previous branch is still unresolved, so the processor must be able to pursue two or more speculation levels, and
- An efficient rerolling mechanism when a branch is mispredicted (minimizing the branch misprediction penalty).

10.9.3.1 Misprediction Penalty :

- The performance of branch prediction depends on the prediction accuracy and the cost of misprediction. Misprediction penalty depends on many organizational features :
- The pipeline length (favouring shorter pipelines over longer pipelines),
- The overall organization of the pipeline,
- The fact if miss speculated, instructions can be removed from internal buffers, or have to be executed and can only be removed in the retire stage,

- The number of speculative instructions in the instruction window or the reorder buffer. Typically only a limited number of instructions can be removed each cycle.
- Misprediction is expensive (11 or more cycles in the Pentium II).

10.9.3.2 Static Branch Prediction :

- Static Branch Prediction predicts always the same direction for the same branch during the whole program execution.
- It comprises hardware-fixed prediction and compiler-directed prediction.
- Simple hardware-fixed direction mechanisms can be :
 - (a) Predict always not taken
 - (b) Predict always taken
 - (c) Backward branch predict to be taken, forward branch predict not to be taken
- Sometimes a bit in the branch opcode allows the compiler to decide the prediction direction.

10.9.3.3 Branch-Target Buffer or Branch-Target Address Cache :

The Branch Target Buffer (BTB) or Branch-Target Address Cache (BTAC) stores branch and jump addresses, their target addresses, and optionally prediction information. The BTB is accessed during the IF stage.

Branch address	Target address	Prediction bits
...

Fig. 10.9.5

10.9.3.4 Dynamic Branch Prediction :

- The hardware influences the prediction while execution proceeds. Prediction is decided on the computation history of the program.
- During the start-up phase of the program execution, where a static branch prediction might be less effective, the history information is gathered and dynamic branch

prediction gets more effective. In general, dynamic branch prediction gives better results than static branch prediction, but at the cost of increased hardware complexity.

10.9.3.5 One-bit Dynamic Branch Predictor :

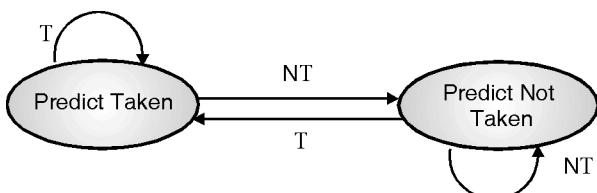


Fig. 10.9.6

- A one-bit predictor correctly predicts a branch at the end of loop iteration, as long as the loop does not exist.
- In nested loops, a one-bit prediction scheme will cause two misprediction for the inner loop :
- One at the end of the loop, when the iteration exits the loop instead of looping again, and one when executing the first loop iteration, when it predicts exit instead of looping.
- Such a double misprediction in nested loops is avoided by a two-bit predictor scheme.

10.9.3.6 Two-bit Prediction :

- A prediction must miss twice before it is changed when a two-bit prediction scheme is applied.

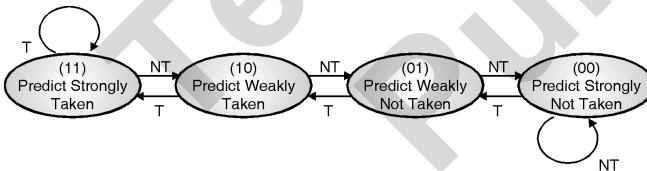


Fig. 10.9.7

10.10 Multiprocessor Systems and Multicore Processor (Intel Core i7 Processor) :

10.10.1 Overlapping the CPU and Memory or I/O Operations :

- This is a very basic parallelism implemented in the Intel's 8086, wherein we had instructions prefetched from the memory before they are to be executed. Also for I/O operations a special dedicated I/O processor can be connected.

- Hence all the operations i.e. accessing the data or instructions from memory, accessing I/O devices and internal LAU operations can be done simultaneously. In the 8086 processor, there are two separate units to perform the memory accesses and the LAU operations named as Bus Interface Unit (BIU) and the Execution Unit (EU).

10.11 Flynn's Classifications :

10.11.1 Flynn's Classification of Parallel Computing :

- A method introduced by Flynn, for classification of parallel processors is most common. This classification is based on the number of Instruction Streams (IS) and Data Streams (DS) in the system. There may be single or multiple streams of each of these. Hence accordingly, Flynn classified the parallel processing into four categories :

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

1. Single Instruction Single Data (SISD) :

- In this case there is a single processor that executes one instruction at a time on single data stored in the memory.
- In fact, this type of processing can be said to be unit processing, hence unit processors fall into this category.
- Fig. 10.11.1 shows this type of system. You will notice there is a Control Unit (CU) that accepts the instruction from the processor and decodes it.
- The Processing Element (PE) accesses the data from the memory and performs the operation on this data as per the signal given by control unit.
- The Memory Module (MM) is connected to the PE and the CU for the data and the instruction streams respectively.

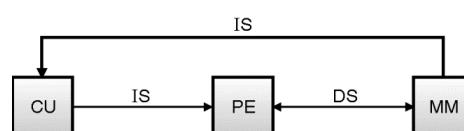


Fig. 10.11.1 : SISD computer

2. Single Instruction Multiple Data (SIMD) :

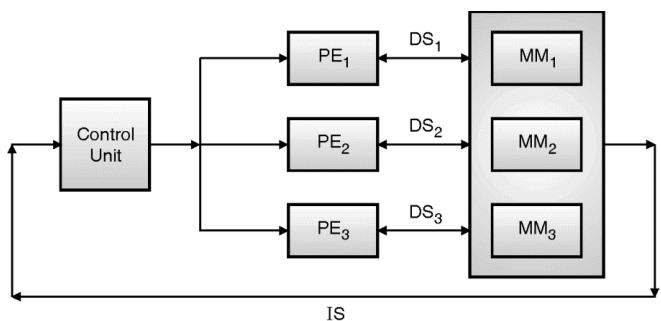


Fig. 10.11.2 : SIMD organization

- In this case the same instruction is given to multiple processing elements, but different data.
- This kind of system is mainly used when many data (array of data) have to be operated with same operation. Vector processors and array processors fall into this category.
- Fig. 10.11.2 shows the structure of a SIMD system

3. Multiple Instruction Single Data (MISD) :

- In case of MISD, there are multiple instruction streams and hence multiple control units to decode these instructions.
- Each control unit takes a different instruction from the different memory module in the same memory.
- The data stream is single. In this case the data is taken by the first processing element.
- This processing element performs an operation on the data given to it and forwards the result to the next processing element for further operation.
- This processing element performs a similar operation and so on the final result reaches back to the same memory module.

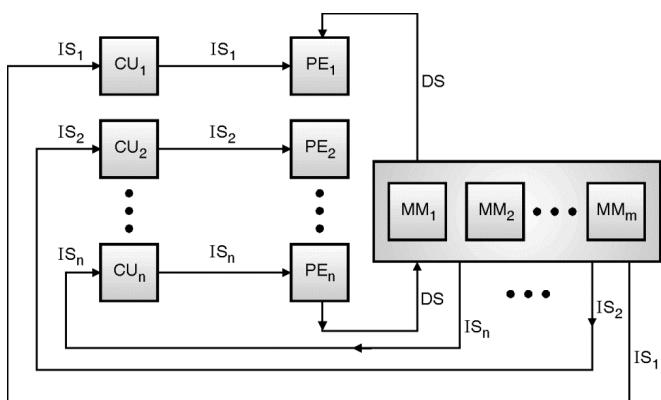


Fig. 10.11.3 : MISD computer

- This system is not used much, but can be used in cases where in a data has to undergo many computations to get the result for e.g. to add two floating point numbers. Fig. 10.11.3 shows the implementation of such a system.

4. Multiple Instruction Multiple Data (MIMD) :

- This is a complete parallel processing example. Here each processing element is having a different set of data and different instructions.
- Examples of this kind of systems are SMPs (Symmetric Multiprocessors), clusters and NUMA (Non-Uniform Memory Access). Fig. 10.11.4 shows the structure of such a system.

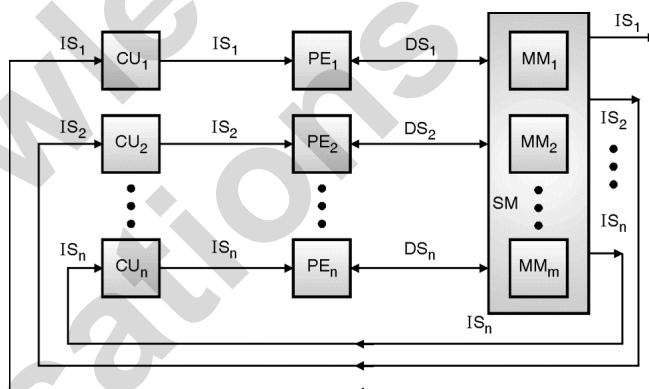
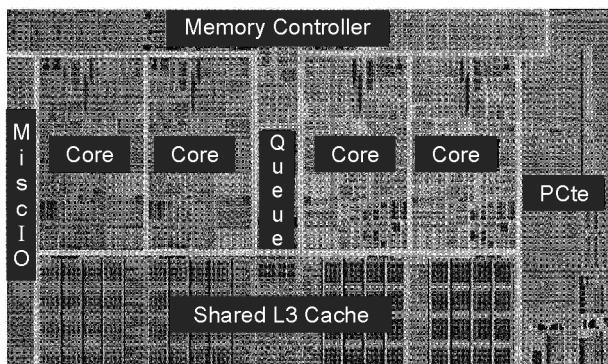


Fig. 10.11.4 : MIMD computer

- Intel brought its mainstream desktop CPU lineup into the Nehalem era today with the launch of the Core i7 860 and 870, and the Core i5 750. Also launched the P55 chipset, which implements a new system architecture that represents a significant break with Intel's past.



**(C-8332) Fig. 10.11.5 : An annotated floor plan of Lynnfield
Source : Intel**

- The floor plan above shows the main blocks in Nehalem, there is no QuickPath Interconnect (QPI) interface. Instead, in a significant twist that differentiates Intel's

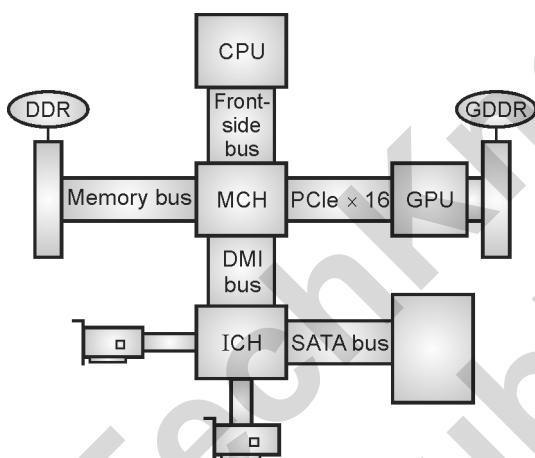
new PC system architecture from even AMD's offerings, there is now a PCIe interface that enables the GPU to attach directly to the processor socket.

This latter move was made in anticipation of two things :

- (1) The GPU will migrate right into the processor socket at a later point when Intel releases a CPU with an on-die GPU integrated into it, and
 - (2) For a discrete GPU, Intel hopes you'll use Larrabee.
- To understand what all of this means, let's look at a few figures.

The P55's new system architecture :

- Fig. 10.11.6 is of a standard Core 2 Duo system, and it represents the general layout of an Intel system up until Nehalem or an AMD system up until the Opteron.



(C-8329) Fig. 10.11.6 : A typical pre-Nehalem Intel system

- In the Fig. 10.11.6, the core logic chipset consists of two primary chips :

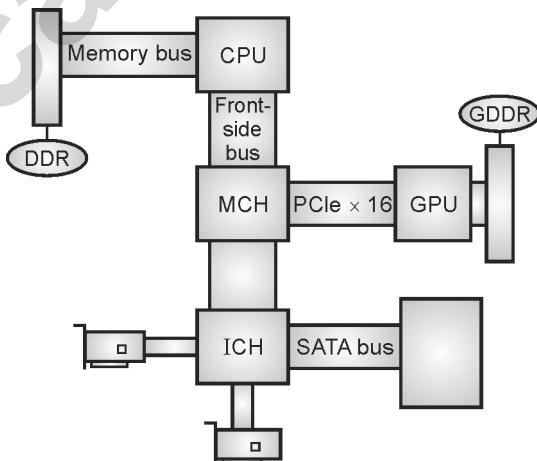
Memory controller hub (MCH) :

- The memory controller hub, also called a "northbridge," links the CPU and GPU with main memory.

I/O controller hub (ICH) :

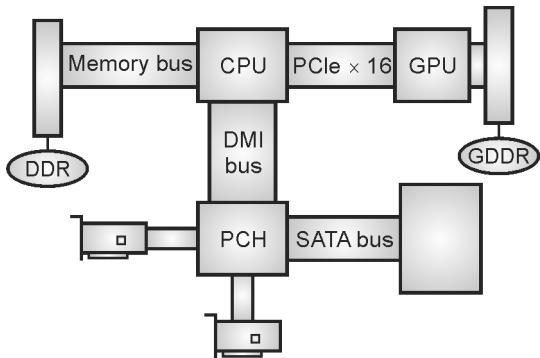
- The I/O controller hub, also called the "southbridge," links the MCH to peripherals and mass storage.
- The ICH typically hosts the USB and other expansion ports, mass storage interfaces, network interfaces, and the like.

- As the GPU gained in size and importance, the standard PC system essentially took on a kind of hacked-together non-uniform memory architecture (NUMA) topology, with two main pools of DRAM (main memory and graphics memory) attached to the two main processors (the CPU and the GPU).
- As the amount of graphics memory increased to the point where the GPU became a second system on a daughter card, this topology began to get more and more unbalanced and inefficient in its use of memory and bandwidth.
- In 2003, AMD made the obvious improvement by moving the memory controller hub up to the CPU socket, so that main memory could attach directly to the CPU the way that GDDR had been directly attached to the GPU for some time. You can see the results below, and, give or take an I/O bridge chip or so, this is basically how AMD single-socket systems have looked since the memory controller went on-die.



(C-8330) Fig. 10.11.7 : A typical AMD single-socket system

- You can see from the Fig. 10.11.7 that, with the memory controller moved onto the processor die, the northbridge has become a kind of "graphics hub"—it hosts the discrete GPU via some PCIe graphics lanes, and it typically has an integrated graphics processor (IGP) along with the requisite display ports.
- The ICH is still there, doing pretty much the same job it always did.



(C-8331) Fig. 10.11.8 : Intel's P55 platform

- Intel's P55 can be seen as an evolution of the AMD topology shown previously, with the graphics hub and memory hub functionality all moved right onto the processor die. Because the northbridge is completely gone, the southbridge/ICH has been rechristened the "platform controller hub," and it's now the only chip in the core logic "chipset" (aside from the BIOS, which is also typically included in the chipset count).
- The PCH is connected to the processor socket by the relatively low-bandwidth (2GB/s) DMI bus that used to connect the MCH to the ICH. Disk I/O, network traffic, and other types of I/O will have to share this link. This shouldn't be a problem for single-socket systems, though.
- So with the advent of the P55, Intel's core logic has gone from a two-chip to a one-chip implementation, pushing ahead of the comparable AMD platform. In theory, this very tight, direct coupling of the GPU + GDDR and CPU + DRAM systems should make for a performance boost vs. both earlier topologies.

10.11.2 i5/i7 Mobile Version :

- The Intel i5 mobile processor has the following features :
 1. Intel hyper-threading technology.
 2. Intel turbo boost technology.
 3. Number of simultaneous threads.
- 4. Intel smart cache.
- 5. Processor integrated memory controller.
- 6. Intel HD graphics.
- 7. Intel express chipset.
- 8. Multi thread/multi-core.

- Intel core i7 is a high-end processor. The higher clock speed, bigger cache and hyper-threading features in i7 processor make them more suitable than i5 processors for certain applications like video encoding, data crunching, graphic-intensive work, multitasking and high-end gaming.
- Mobile version of i5/i7 processor have the following features :
 1. i5 mobile processor are dual-core.
 2. i7 mobile processor are quad-core.
 3. i5/i7 processors support hyper-threading.
 4. i5/i7 processors come with direct media interface and integrated GPU.
 5. i5/i7 processors come with Ivy bridge.
 6. i5/i7 processors come with turbo boost facility.
- The basic block diagram of i5/i7 processor is given in the Fig. 10.11.9.
- i7 mobile processors are the next generation 64-bit, multi-core mobile processors built on 45-nanometer process technology.
- It has quad-core.
- It has 32-KB instruction and 32-KB data first-level cache (L1) for each core.
- It has 256-KB shared instruction/data second-level cache (L2) for each core.
- It can have upto 8-MB of shared instruction/data third-level cache (L3), shared among all cores.

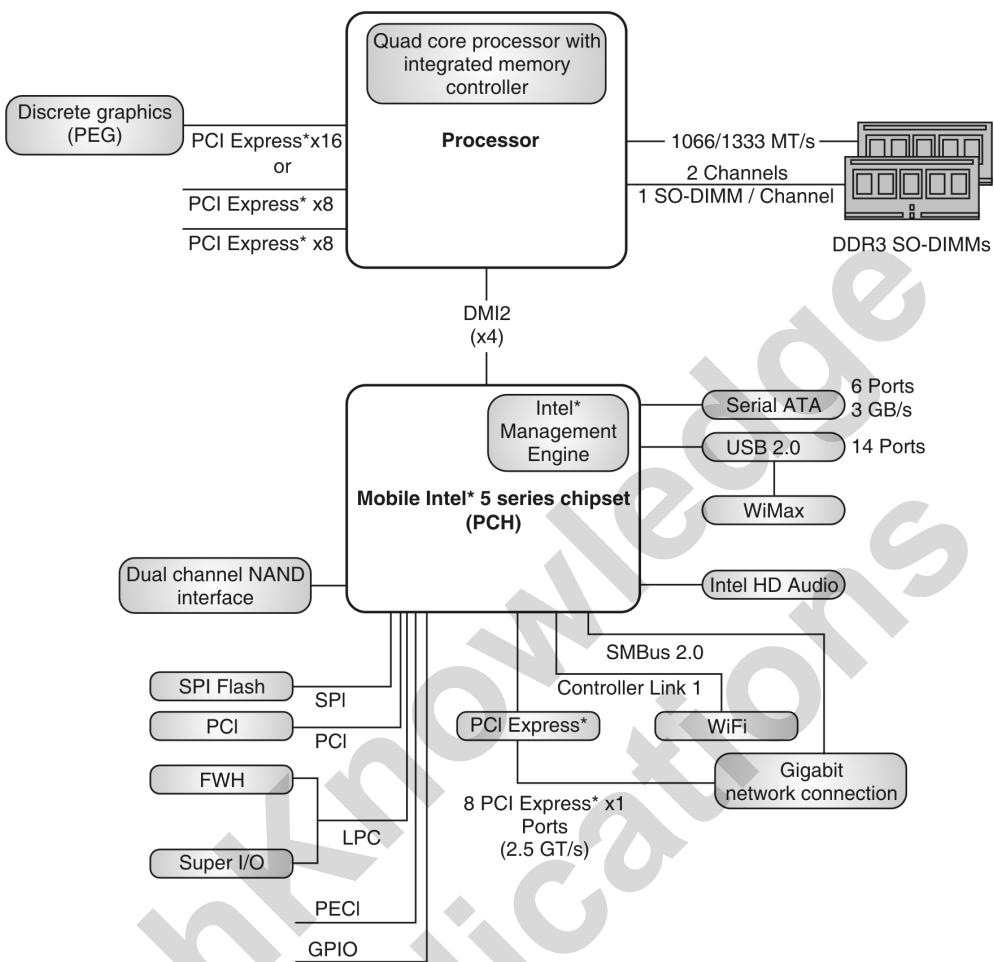


Fig. 10.11.9 : Mobile processor

- It supports 1 or 2 channels of DDR3 memory.
- It has PCI express bus. It is an improved version of older PCI, PCI-X and AGP bus standards.
- Intel core i7-900 supports one, 16-lane PCI express port configurable to two, 8-LAN PCI express port intended for graphics.
- It supports direct media interface (DMI).
- It supports platform environment control interface (PECI).
- The second generation microprocessors of the Intel core i3, i5 and i7 processors are the ones we normally seen in the computers today. The comparison of the same is given in the Table 10.11.1.

Table 10.11.1 : Comparison of i3, i5 and i7 processors

Sr. No.	Feature	i3	i5	i7	i7 Extreme
1.	Number of cores	2 for desktop as well as for Laptops	4 for Desktop 2 for Laptop	4 or 6 for Desktop 2 or 4 for Laptop	6 for desktop 4 for mobile
2.	Processing threads	4 for desktop as well as Laptops	8 threads for desktop 4 threads for Laptop	8 or 12 threads for desktop 4 or 8 threads for Laptop	12 threads for Desktop 8 threads for Laptop



Sr. No.	Feature	i3	i5	i7	i7 Extreme
3.	Maximum base clock frequency	3.4GHz	3.4GHz	3.2GHz	3.3GHz
4.	Maximum turbo boost frequency	Not Applicable	3.8GHz	3.8GHz	3.9GHz
5.	Maximum smart cache size	3MB	6MB	12MB	15MB
6.	Intel turbo boost 2.0	Not present	Present	Present	Present
7.	Intel Hyperthreading	Present	Present only in Laptop processors	Present	Present
8.	Best Desktop processor	Intel Core i3-2130 (3.4GHz, 3MB)	Intel Core i5-2550K (3.4GHz, 6MB)	Intel Core i7-3930 (3.2GHz, 12MB)	Intel Core i7-3960 (3.3GHz, 15MB)
9.	Best Mobile (Laptop) processor	Intel Core i3-2370 (2.4GHz, 3MB)	Intel Core i5-2540M (2.6GHz, 3MB)	Intel Core i7-2860 (2.5GHz, 8MB)	Intel Core i7-2960XM (2.7GHz, 8MB)

Review Questions

- Q. 1 What are the different addressing modes explain any two ?
- Q. 2 With an example explain register indirect addressing relative addressing mode.
- Q. 3 Compare RISC and CISC.
- Q. 4 Explain the instruction set of 8085.
- Q. 5 What are the different properties of RISC processor ?
- Q. 6 List out features or advantages of RISC systems.

- Q. 7 Compare ON-chip register file and cache evaluation.
- Q. 8 Write a short note on polling.
- Q. 9 Write a short note on pipeline processing.
- Q. 10 Explain six stage pipelining.
- Q. 11 State collision vector. Explain collision free scheduling.
- Q. 12 State : 1. Latency, 2. Throughput.
- Q. 13 Write a short note on pipeline hazards.
- Q. 14 What are the methods to resolve the data hazards ?
- Q. 15 Write a short note on dynamic instruction scheduling.

□□□

Unit 6

Chapter 11

Memory & Input / Output Systems

Syllabus

Memory Systems : Characteristics of memory systems, Memory hierarchy, Signals to connect memory to processor, Memory read and write cycle, Characteristics of semiconductor memory : SRAM, DRAM and ROM, Cache memory – Principle of locality, Organization, Mapping functions, Write policies, Replacement policies, Multilevel caches, Cache coherence,

Input / Output systems : I/O module, Programmed I/O, Interrupt driven I/O, Direct Memory Access (DMA).

Case study : USB flash drive.

Chapter Contents

11.1 Introduction to Memory and Memory Parameters	11.8 Cache Memory : Concept, Architecture (L1, L2, L3) and Cache Consistency
11.2 Memory Hierarchy : Classifications of Primary and Secondary Memories	11.9 Cache Mapping Techniques
11.3 Types of RAM	11.10 Pentium Processor Cache Unit
11.4 ROM (Read Only Memory)	11.11 Input / Output System
11.5 Allocation Policies	11.12 I/O Modules and 8089 IO Processor
11.6 Signals to Connect Memory to Processor and Internal Organization of Memory	11.13 Types of Data Transfer Techniques : Programmed I/O, Interrupt Driven I/O and DMA
11.7 Memory Chip Size and Numbers	

11.1 Introduction to Memory and Memory Parameters :

- When a memory is taken then there are various characteristics of this memory that are considered. The characteristics of memory are based on the following :

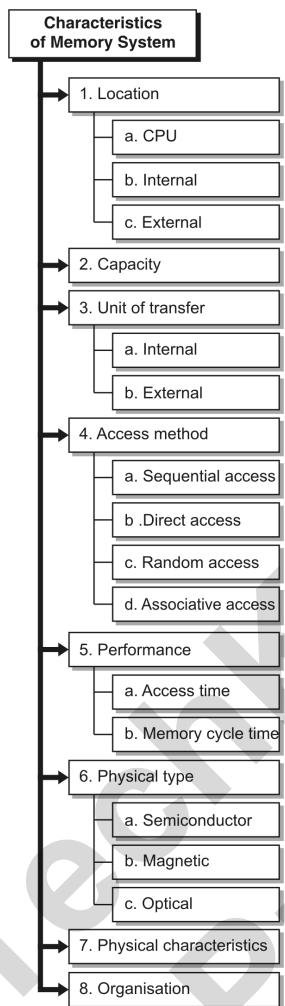


Fig. 11.1.1 : Characteristics of memory system

1. Location :

The memory can be located in one of the following :

- (a) **CPU** : This includes CPU registers and on-chip cache memory.
- (b) **Internal** : This includes the memory that the processor can directly access.
- (c) **External** : This is normally removable or virtual memory and hence access is slower.

2. Capacity :

It is measured in terms of the word size and the number of words. Word size is the size of each location. Number of words is the number of locations.

3. Unit of transfer :

This refers to the size of the data that is transferred in one clock cycle. It mainly depends on the data bus size. The data as discussed earlier may be internal or external and accordingly will be the data to be transferred in one clock pulse :

- (a) **Internal** : It is related to the communication of data with the memory directly accessible. It is usually governed by data bus width.
- (b) **External** : This is the data communication with the external removable memory or virtual memory. It is usually a block which is much larger than a word.

4. Access method :

There are various methods of accessing the memory based on the memory organization. These methods are listed below with examples :

- (a) **Sequential access** : The sequential access means start from the beginning and read through in order until the byte to be read is reached. Hence the access time depends on location of data and previous location accessed. For example, magnetic tape. If one wants to listen to the third stanza of the fourth song stored in an audio cassette, he has to go through the entire first song second and the third song, and then the first stanza, second stanza of the third song and then reaches to the third stanza of that song.
- (b) **Direct access** : Here individual blocks have unique address and the access is done by jumping to vicinity plus sequential search. Hence access time depends on location and previous location. For example magnetic or optical disk. Let take the same example that a person wants to listen to the third stanza of the fourth song on a CD, then he can directly reach to the fourth song, but thereafter he has to access the stanzas of the song sequentially to reach to the third stanza.
- (c) **Random access** : In case of random access individual addresses identify locations exactly. Hence the access time is independent of location or previous access. For example RAM. In case of a RAM, any location to be accessed can be directly reached to without going through the locations sequentially.
- (d) **Associative access** : Here the data is located by a comparison with contents of a portion of the stored data(address). Hence the access time is independent of location or previous access. For example cache. In case of cache memory, each location has a tag associated

with it, and to reach to the required location the tags are to be compared with the location to be accessed. There are techniques used to reach to the required tagged location at a faster speed.

5. Performance :

The performance of the memory depends on its speed of operation or the data transfer rate. The data transfer rate is the rate at which the data is transferred. The speed of operation depends on two things :

- (a) **Access time** : The time between providing the address and getting the valid data from memory is called as its access time i.e. the address to data time.
- (b) **Memory cycle time** : The time that is required for the memory to "recover" before next access i.e. the time between two addresses is called as memory cycle time.

6. Physical type :

The physical material using which the memory is made can be different like :

- (a) **Semiconductor** : Memory can be made using semiconductor material i.e. ICs, for example RAM.
- (b) **Magnetic** : Memory can also be made using magnetic read and write mechanism, for example Magnetic disk and Magnetic tape.
- (c) **Optical** : Optical memories i.e. memories that use optical methods to read and write have become famous these days, for example CD and DVD.
- (d) There are some other methods using which data was stored in early days like Bubble and Hologram.

7. Physical characteristics :

The physical characteristic of memory is also an important aspect to be considered. This includes the volatility, power consumption, erasable / not erasable, etc.

8. Organisation :

It is not that always the memory will be organized sequentially. There are some other types of memory organization like interleaved memory, etc. Interleaved memory is used in microprocessor 8086.

11.1.1 Bytes and Bits :

- The byte is a unit of digital information that mostly consists of eight bits.
- Infact, a byte was the number of bits used to encode a single character of text in a computer and for this reason it has become the basic addressable element in

many computer architectures. The size of the byte has been hardware dependent and no definition exists.

- The fact is that standard of eight bits is also convenient power of two permitting the values from 0 to 255 for one byte. With ISO/IEC 80000-13, this common meaning was codified in a formal standard. Many types of applications use variables representable in eight bits or multiple of eight bits.

11.2 Memory Hierarchy : Classifications of Primary and Secondary Memories :

- Memory Hierarchy explains that the nearer the memory to the processor, faster is its access. But costlier the memory becomes as it goes closer to the processor. The following sequence is in faster to slower or costlier to cheaper memory :
 1. Registers i.e. inside the CPU.
 2. Internal memory that includes one or more levels of cache and the main memory. Internal memory is always RAM, SRAM for cache and DRAM for main memory. The differences between the SRAM and DRAM will be seen in a later section in this chapter. This is also called as the primary memory.
 3. External memory or removable memory includes the hard disk, CDs, DVDs etc. This is the secondary memory.
- Fig. 11.2.1 shows the memory hierarchy based on the closeness to the processor. The registers as discussed are the closest to the processor and hence are the fastest while off-line storage like magnetic tape are the farthest and also the slowest.

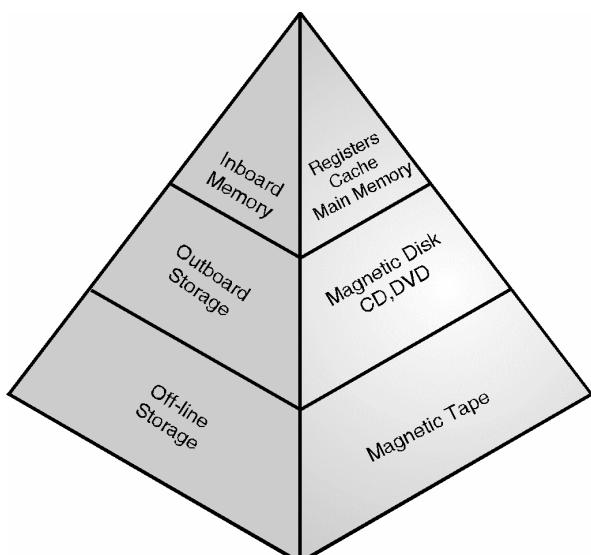


Fig. 11.2.1 : Memory hierarchy

- The list of memories from closest to the processor to the farthest is given as below :
 1. Registers
 2. L1 Cache
 3. L2 Cache
 4. Main memory
 5. Magnetic Disk
 6. Optical
 7. Tape
- To have a large faster memory is very costly and hence the different memory at different levels gives the memory hierarchy. How does this memory hierarchy give faster operation and some other terms like cache etc. will be understood in the subsequent sections.
- ROM or the read only memory is quite cheaper compared to RAM and is mainly used for implementation of the secondary or the virtual memory. Thus the application of ROM is for virtual or secondary memories like hard disks, external storage like CD / DVD and floppy disks etc.
- We will see different types of ROM in the subsequent sub-section and thereafter some ROM memories used in computers like magnetic disk, CD, DVD etc.

11.3 Types of RAM :

- RAM (Random Access Memory) is called so because any memory location in this IC can be accessed randomly.
- There are two types of RAM, namely, SRAM (Static RAM) and DRAM (Dynamic RAM).
- SRAM is made up of flip flops while the DRAM is made up of capacitors.
- Since DRAM is made using capacitors, it requires less number of components to make a one bit cell, hence also requires less space on the silicon wafer. Thus it is also comparatively cheaper.
- But it is slower than SRAM, because capacitors require time for charging and discharging. Also the capacitors loose charge in some time and hence need to be recharged according to the data, this is called as refreshing the DRAM. Fig. 11.3.1(a) shows the structure of a DRAM cell.
- The address line selects the particular location, it enables the MOSFET that connects the capacitor to the data bus and hence if the data is to be read, simply the data line gets the data to be read; while if the data is to

be written the data is to be given on the data line and will be written on the capacitor.

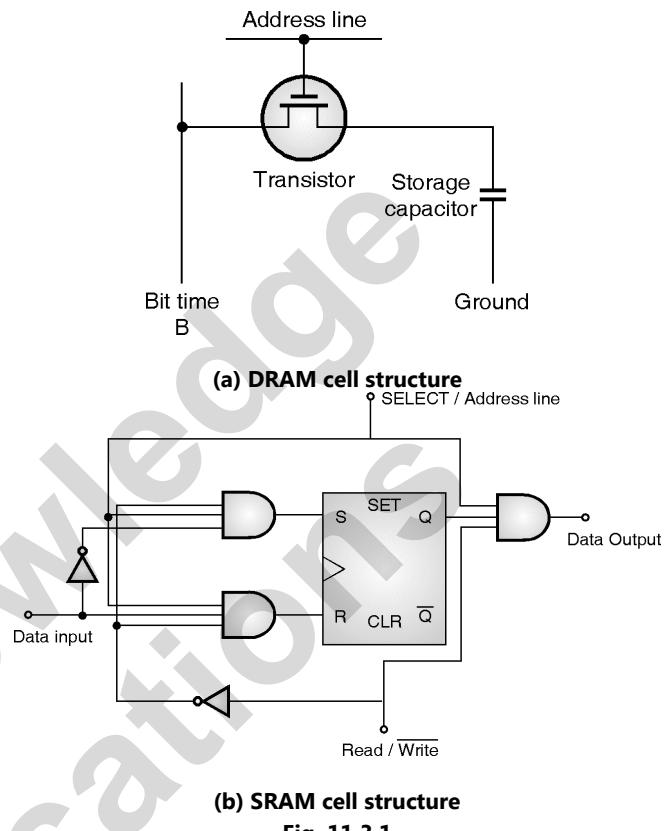


Fig. 11.3.1

- On the other hand, the SRAM has each cell made of a flip-flop, thus requires more components as compared to the DRAM cell. Hence it occupies more space on the silicon wafer, and is costlier.
- Thus it is also costlier. But the advantage is that it doesn't require any refreshing and is also very fast compare to DRAM. Fig. 11.3.1(b) shows the structure of the DRAM cell.
- The flip-flop is used to store the data. There is a AND gate at the output of the flip-flop, which will be enabled by the select line (that works as an address line) and the Read / Write operation (when data has to be read) when it is logic '1'. Thus when both i.e. the select line and the Read / Write lines are '1', the output of the flip-flop will be available on the data line.
- Similarly for the write operation, the select line must be enabled by making it logic '1' and the Read / Write line must be logic '0'. Thus the data from the input line will be stored in the flip-flop.



- Table 11.3.1 shows the differences between the SRAM and DRAM.

Table 11.3.1

Sr. No.	SRAM	DRAM
1.	No refreshing required.	Continuous refreshing required (disadvantage).
2.	It is faster for accessing data.	It is slower in accessing data.
3.	It takes more space on chip as more number of components are required per bit.	It takes less space on chip as less number of components are required per bit.
4.	Hence is also costly.	Hence is cheaper.
5.	Bit density is lesser.	Bit density is more.
6.	The bit is stored in a flip-flop.	The bit is stored as a charged or discharged capacitor.
7.	SRAM is mainly used or selected for cache memory.	DRAM is mainly used or selected for semiconductor main memory.

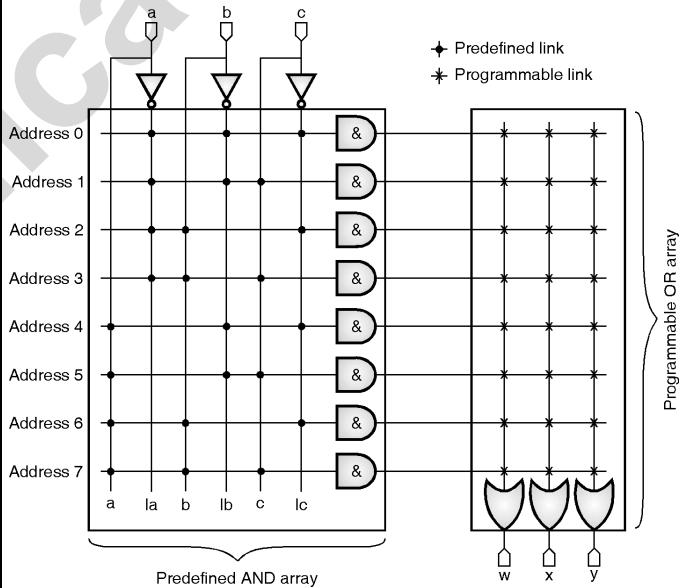
11.4 ROM (Read Only Memory) :

- ROM or the read only memory is quite cheaper compared to RAM and is mainly used for implementation of the secondary or the virtual memory.
- Thus the application of ROM is for virtual or secondary memories like hard disks, external storage like CD / DVD and floppy disks etc.
- We will see different types of ROM in the subsequent sub-section and thereafter some ROM memories used in computers like magnetic disk, CD, DVD etc.

11.4.1 Types of ROM :

- There are various types of ROM available based on whether or not it can be re-written; they are called as ROM, PROM, EPROM and EEPROM. These types of memories will be studied in this section.
- There are some more ROMs available these days like flash memory, OTP etc.
- The ROM is a memory wherein, the user cannot write anything. The data to be stored in the ROM is to be given to the ROM manufacturer, who writes this data on the ROM and provides the same.

- The PROM (Programmable Read Only Memory) or also sometimes referred to as OTP (One Time Programmable) memory, as it can be written onto only once. When manufactured, it is blank, once written on it, it cannot be re-written. There are diodes that are used to store data, and they are fused or kept as it is to store the data in them. The internal diagram of the PROM is shown in Fig. 11.4.1.
- The AND array is used as address lines and the OR array as data lines. The AND array (on the left in Fig. 11.4.1) comes as predefined connections as shown in the Fig. 11.4.1 in sequence of binary, in this case from "000" to "111", as there are three bit address.
- The OR array (on the right hand in Fig. 11.4.1) comes with programmable link, the ones to be retained can be retained while the remaining fused or opened.
- Hence whenever a memory address is given to the address lines (a, b and c in Fig. 11.4.1) the specified location will be selected and according to the fused links, the data will be available on the OR gates output lines.

**Fig. 11.4.1 : PROM**

- The EPROM (Erasable Programmable Read Only Memory) although extinct today and is replaced by EEPROM or E²PROM, but it used to be the only erasable memory available earlier.
- In case of EPROM the data written can be erased by keeping the EPROM IC in the UV box, as the UV rays erase the previously written data on the EPROM.

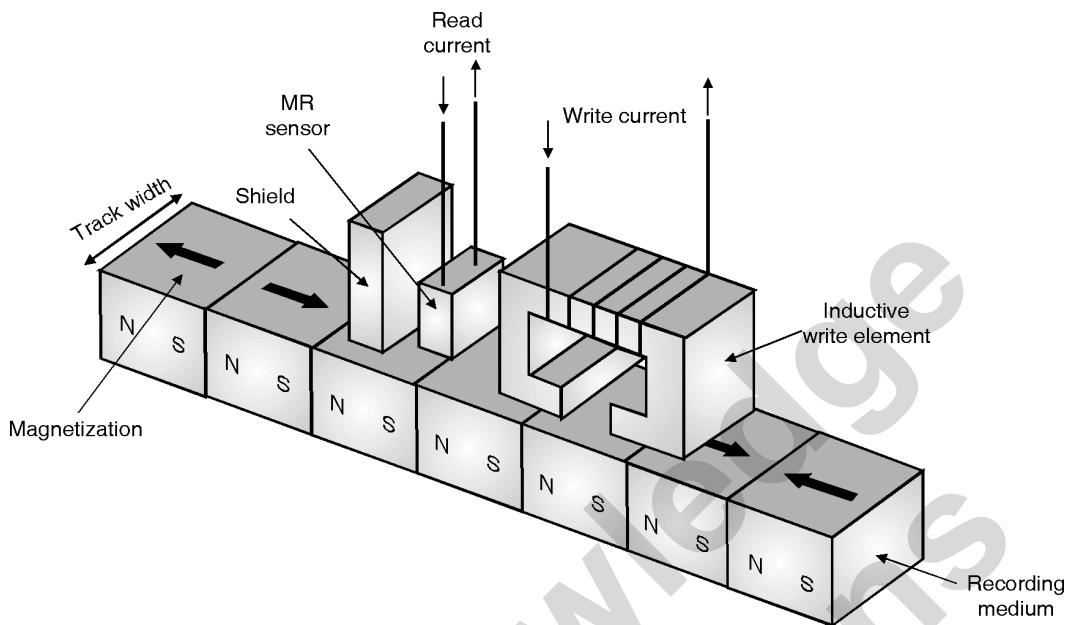
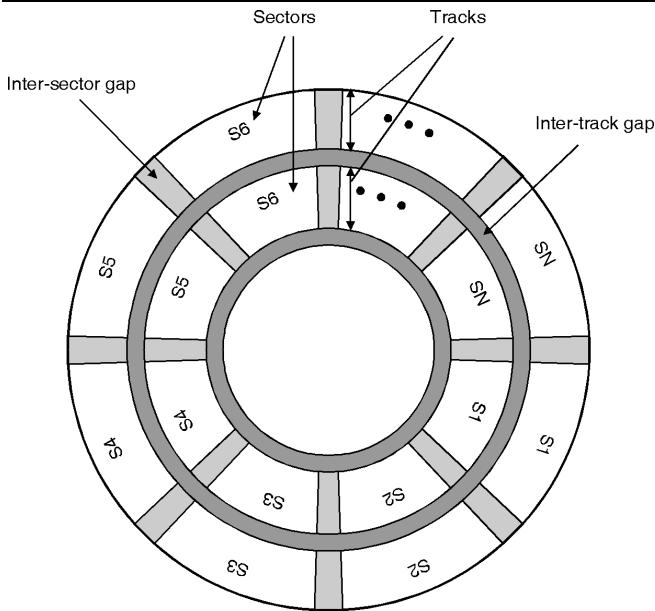
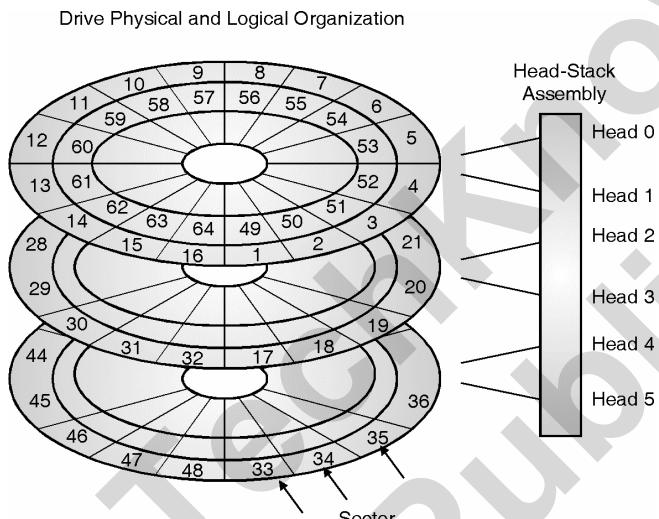


Fig. 11.4.2 : Read-Write mechanism

- The EPROM as discussed earlier are these days replaced with EEPROM (Electrically Erasable Programmable Read Only Memory). The EEPROMs are erased by giving an extra supply voltage.
- The head may be single for both read and write operations or separate ones.
- During read/write operation, head is stationary while the platter (disk) rotates.
- Write operation is done by passing current through coil that produces magnetic field and then the pulses are sent to head. Thus the magnetic pattern i.e. NS (North-South) or SN (South-North) is recorded on surface below.
- Read operation is done by magnetic field moving relative to coil that produces current. According to the magnetic pattern the data is read by the head.
- The organization of data on the platter is in a special manner with concentric circles called as tracks as shown in Fig. 11.4.3(a). Further the tracks are divided into sectors.
- The data is also stored in a special manner such that first the data is stored in the first track of first platter (upper and lower sides) and then in the first track of the second platter (upper and lower sides), then of the third (upper and lower sides) and so on. This is shown in the Fig. 11.4.3(b).
- Thus when reading from one track of a platter, the head mechanism may not be moved and the other head will start reading from the same track of another platter.



(a) Data organization on a disk



(b) Data organization on multiple platters

Fig. 11.4.3

- A floppy disk is single platter, while a hard disk or winchester disk is multi platter as shown in the Fig. 11.4.3(b).
- In this case one head for each side of the multiple platters are mounted to form a head stack assembly.
- It is called as Winchester hard disk because it was developed by IBM in Winchester (USA). It is a sealed unit with the platters and the heads fly on boundary layer of air as disk spins.
- Also, there is very small head to disk gap making it more robust. Winchester hard disk is cheap and the fastest external storage.

- There is gap between the two sectors as well as between two tracks as shown in Fig. 11.4.3(a).
- The disk moves at constant angular velocity and hence the data is read at the same speed, may be the innermost track or the outermost track.
- Each data stored on the disk is stored in a special manner with some ID information as shown in the Fig. 11.4.4.

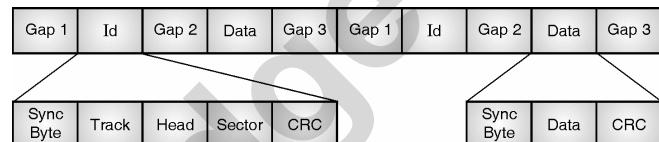


Fig. 11.4.4 : Data storage format in magnetic memory

11.4.3 Optical Memory :

- The memory devices like Compact Disc (CD) and Digital Versatile Disc or Digital Video Disc (DVD) use the optical method to read the data written on them.
- The following sub-sections discuss about the CD and the DVD data storage and reading.

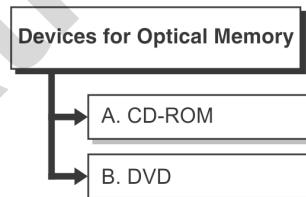


Fig. 11.4.5 : Devices for optical memory

A. CD-ROM :

- CD-ROM was originally built for audio and was of the capacity of 650Mbytes giving over 70 minutes audio.
- The construction of the CD was such that it used polycarbonate coating with highly reflective coat, usually aluminium.
- In the CD-ROM the data stored as pits and lands as shown in Fig. 11.4.6(a).
- These pits and lands are read by reflecting laser. The CD has a constant packing density hence constant linear velocity across a track is required as against the constant angular velocity in case of magnetic discs.
- Fig. 11.4.6(a) shows that the CD is made up of three layers, namely the polycarbonate plastic, aluminium and the protective material like acrylic.
- The laser beam incident on the highly reflective substance like aluminium, returns back in some amount of time. Based on this time gap, the optical disk reader can realize that there was a land or a pit.

- In case if it is a land it will take more time to return while less time in case if it is a pit as seen in the Fig. 11.4.6(a).

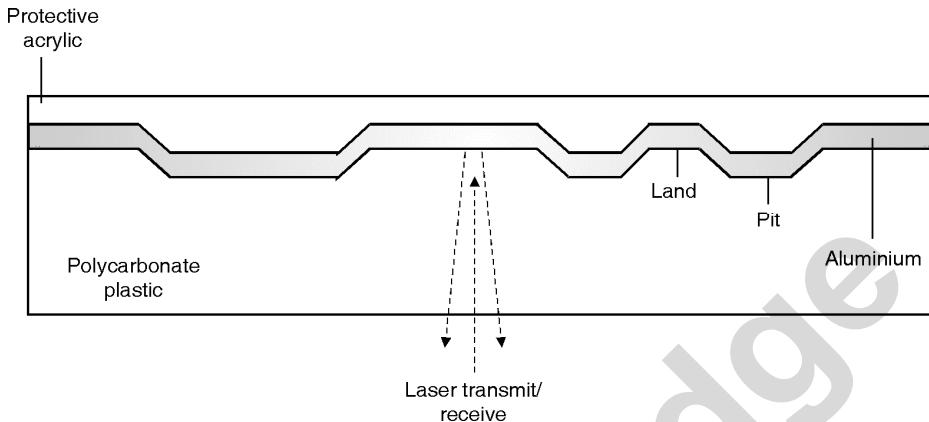


Fig. 11.4.6(a) : Construction of CD

- The data format on a CD-ROM is shown in the Fig. 11.4.6(b). Initially a data 00H is stored, followed by 10 bytes of FFH and again a 00H, called as the synchronous 12 bytes.
- The next is the 4 bytes ID (IDentity) about the time required for this data to be played (in MINutes and SEConds), the sector in which the data is placed and the mode. There are three modes,
 - o Mode 0 indicates blank data field
 - o Mode 1 indicates 2048 byte data + error correction
 - o Mode 2 indicates 2336 byte data and no correction data
- Thus the remaining two fields contain data and error correction code (ECC) as defined by the mode bits.

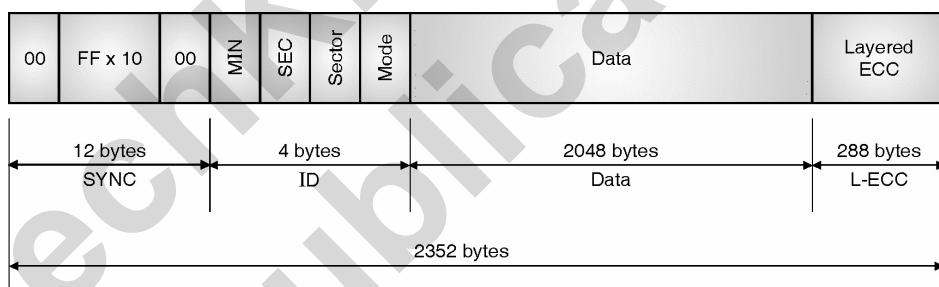


Fig. 11.4.6(b) : Data Format on CD

B. DVD :

- The major difference between a CD and DVD is that a DVD has multiple layers and hence very high capacity.
- Another major difference in a DVD with respect to CD is that the DVD has more denser data storage mechanism which results in the data storage capacity of around 4.7G per layer of a DVD.
- There are DVDs available with single layer as well as multiple layers.
- Fig. 11.4.7 shows the constructional differences of a CD and DVD.

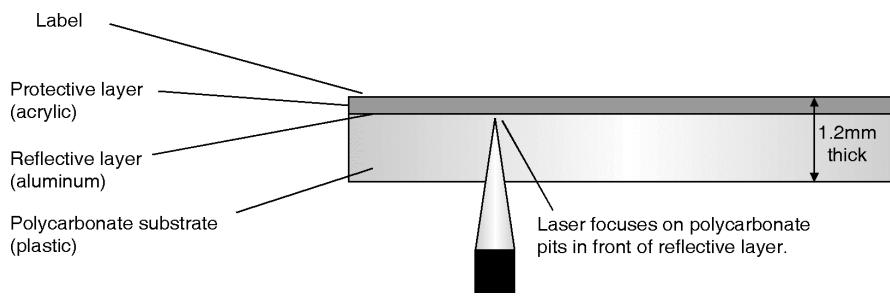


Fig. 11.4.7(Contd...)

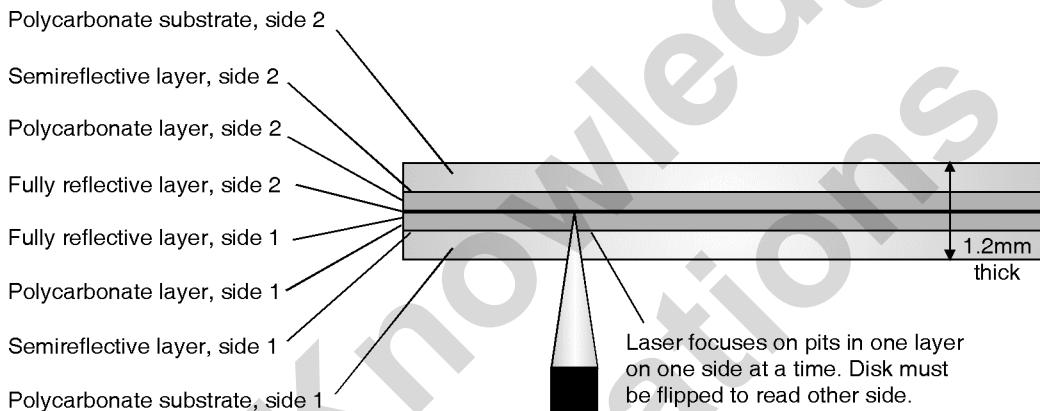
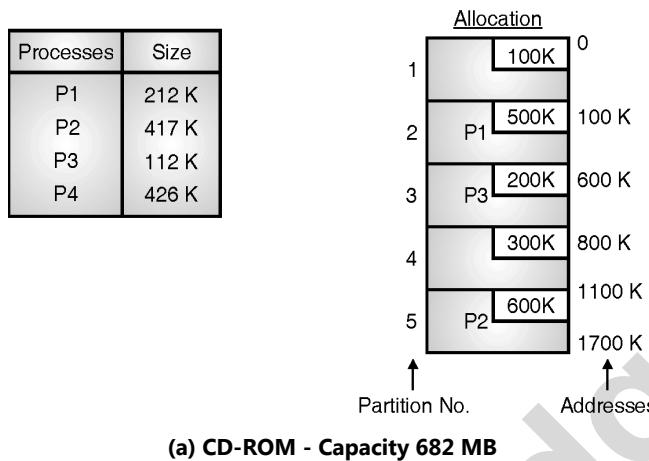


Fig. 11.4.7 : Construction of (a) CD and (b) DVD

- As seen in the Fig. 11.4.7(b), the double sided, two layers DVD, has a reflective and semi-reflective layers on both the sides.
- Hence in this case, the laser beam and receiver have to be on both the sides of the disc.
- Also there have to be two types of beam with low and high intensity, the low intensity beam is reflected by the semi-reflective substance, while the high intensity beam is reflected by the highly reflective substance. Thus giving a mechanism to read the data written on both the layers of each of the side.

11.5 Allocation Policies :

- Partitioning refers to logical division of the memory into subparts so that they can be accessed individually by tasks fragmentation generally happens when memory blocks have been allocated and are freed randomly.
- This results in splitting of partition memory into small non-continuous fragments.
- There are 3 memory allocation policies :

- (1) **Best fit** : In this case the smallest available fragment is searched and the required data is stored in that fragment. The smallest fragment searched for should be greater than or equal to the size of data to be stored.
- (2) **Worst fit** : In this case the largest available block is used to store the data.
- (3) **Next fit** : In this case immediate next empty block of a size equal to or greater than the size of data to be stored is searched sequentially and the required data is stored there.

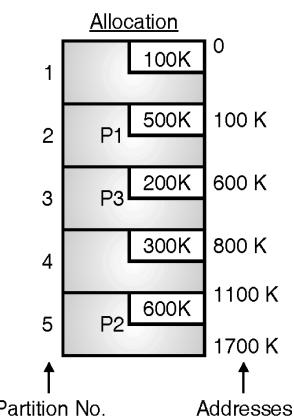
Ex. 11.5.1 : Given the memory partitions of size 100 K, 500 K, 200 K, 300 K and 600 K (in order). How would each of the first fit, best-fit, worst fit algorithms place the processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?



Soln. :

I] **First-fit :**

Processes	Size
P1	212 K
P2	417 K
P3	112 K
P4	426 K

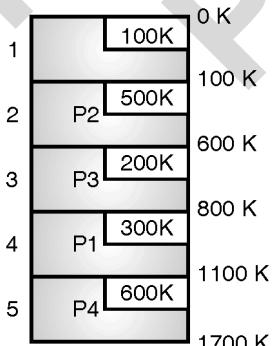


(S9.5) Fig. 11.5.1

- Partition number 2 of size 500 K is assigned to P1 (size = 212 K). It is the first partition that can accommodate P1.
- Partition number 5 of size 600 K is assigned to P2 (size = 417 K). It is the first empty partition that can accommodate P2.
- P3 is assigned to partition 3.
- P4 cannot be executed.

$$\begin{aligned} \text{Memory utilization} &= \frac{\text{Memory utilized}}{\text{Total memory}} \\ &= \frac{\text{Memory utilized by P1, P2 and P3}}{\text{Total memory}} \\ &= \frac{212 \text{ K} + 417 \text{ K} + 112 \text{ K}}{1700 \text{ K}} \\ &= \frac{741}{1700} = 0.436 \end{aligned}$$

II] **Best-fit :**



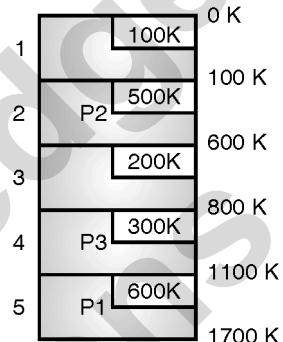
(S9.5) Fig. 11.5.1(a)

- Partition no. 4 of size 300 K is allocated to P1 (212 K). It is the smallest free partition that can accommodate P1.
- Partition no. 2 of size 500 K is allocated to P2 of size 417 K. It is the smallest free partition that can accommodate P2.

- Similarly, partition no.3 is allocated to P3 and the partition no.5 is allocated to P4.

$$\begin{aligned} \text{Memory utilization} &= \frac{\text{Memory utilized by P1, P2, P3 and P4}}{\text{Total memory}} \\ &= \frac{212 \text{ K} + 417 \text{ K} + 112 \text{ K} + 426 \text{ K}}{1700 \text{ K}} \\ &= \frac{1167 \text{ K}}{1700 \text{ K}} = 0.686 \end{aligned}$$

Worst-Fit :



(S9.5) Fig. 11.5.1(b)

- The largest free partition no.5 of size 600 K is allocated to P1 (212 K).
- P2 (size 417 K) is assigned to partition no.2. Partition no. 2 is the largest free partition and it can accommodate P2.
- P3 (size 112 K) is assigned to partition no.4. Partition no. 4 is the largest free partition.
- P4 cannot be executed as there is no free partition that can accommodate P4.

$$\begin{aligned} \text{Memory utilization} &= \frac{\text{Memory utilized by P1, P2, P3}}{\text{Total memory}} \\ &= \frac{212 \text{ K} + 417 \text{ K} + 112 \text{ K}}{1700 \text{ K}} \\ &= \frac{741}{1700} = 0.436 \end{aligned}$$

11.6 Signals to Connect Memory To Processor and Internal Organization of Memory :

- The read / write memories consist of an array of registers wherein each register has a unique address. Fig. 11.6.1 shows the block diagram of a memory device.
N : number of registers
M : word length.

Example : If a memory is having 13 address lines and 7 data lines, then number of registers / memory locations $= 2^N = 2^{13} = 8192$ word length = M bit = 7 bit.

- The number of address lines of a microprocessor depends on the size of the memory.

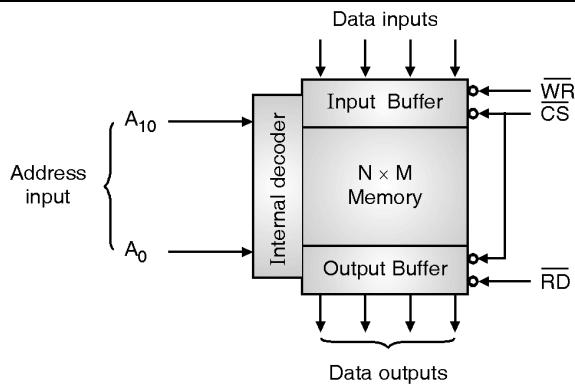


Fig. 11.6.1 : Block diagram of a memory device

Number of address lines required	Size of memory in bytes
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512

Number of address lines required	Size of memory in bytes
10	1024 \approx 1 k
11	2048 \approx 2 k
12	4096 \approx 4 k
13	8192 \approx 7 k
14	16384 \approx 16 k
15	32768 \approx 32 k
16	65536 \approx 64 k

11.7 Memory Chip Size and Numbers :

Table 11.7.1 : EPROM ICs available in the market

IC number	Memory size Address data	Number of pins
2716	2 k \times 7	24
2732	4 k \times 7	24
2764	8 k \times 7	28
27128	16 k \times 7	28
27256	32 k \times 7	28
27512	64 k \times 7	28

27512	27256	2764	2732A	2716
A ₁₅	V _{PP}	V _{PP}		
A ₁₂	A ₁₂	A ₁₂		
A ₇	A ₇	A ₇	A ₇	A ₇
A ₆	A ₆	A ₆	A ₆	A ₆
A ₅	A ₅	A ₅	A ₅	A ₅
A ₄	A ₄	A ₄	A ₄	A ₄
A ₃	A ₃	A ₃	A ₃	A ₃
A ₂	A ₂	A ₂	A ₂	A ₂
A ₁	A ₁	A ₁	A ₁	A ₁
A ₀	A ₀	A ₀	O ₀	A ₀
O ₀	O ₀	O ₀	O ₀	O ₀
O ₁	O ₁	O ₁	O ₁	O ₁
O ₂	O ₂	O ₂	O ₂	O ₂
GND	GND	GND	GND	GND

27128
V _{PP}
A ₁₂
A ₇
A ₆
A ₅
A ₄
A ₃
A ₂
A ₁
A ₀
O ₀
O ₁
O ₂
GND
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
V _{CC}
PGM
A ₁₃
A ₈
A ₉
A ₁₁
A ₁₀
A ₁₀
N.C.
A ₈
A ₉
A ₉
A ₁₁
A ₁₁
A ₁₀
A ₁₀
A ₁₀
OE/V _{PP}
CE
CE
O ₇
O ₆
O ₅
O ₄
O ₃
O ₃

2716	2732A	2764	27256	27512
V _{CC}	V _{CC}	V _{CC}	V _{CC}	V _{CC}
PGM	N.C.	A ₁₄	A ₁₄	A ₁₄
A ₁₃	A ₁₃	A ₁₃	A ₁₃	A ₁₃
A ₈	A ₈	A ₈	A ₈	A ₈
A ₉	A ₉	A ₉	A ₉	A ₉
V _{PP}	A ₁₁	A ₁₁	A ₁₁	A ₁₁
A ₁₀	OE/V _{PP}	A ₁₀	A ₁₀	A ₁₀
A ₁₀	CE	CE	CE	CE
O ₇	O ₇	O ₇	O ₇	O ₇
O ₆	O ₆	O ₆	O ₆	O ₆
O ₅	O ₅	O ₅	O ₅	O ₅
O ₄	O ₄	O ₄	O ₄	O ₄
O ₃	O ₃	O ₃	O ₃	O ₃

Fig. 11.7.1(a) : Pin configuration

Pin Names
A ₀ – A ₁₃ Addresses
— CE Chip enable
— OE Output enable
O ₀ – O ₇ Outputs
— PGM ProgROM
N.C. No connect

Fig. 11.7.1(b)



Table 11.7.1 : RAM ICs

IC no.	Type	Memory size
6116	SRAM	2k × 7
6264	SRAM	8k × 7
2114	SRAM	1k × 4

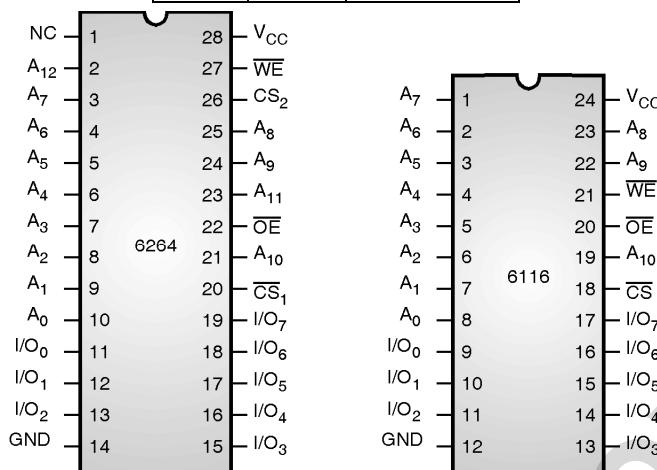


Fig. 11.7.2 : Pin configuration

Table 11.7.2

Truth Table for 6264

WE	CS ₁	CS ₂	OE	Mode
X	1	X	X	Not selected
X	X	0	X	(Power down)
1	0	1	1	Output disable
1	0	1	0	Read
0	0	1	1	Write
0	0	1	0	Write

Truth Table for 6116

CS	OE	WE	Mode
1	X	X	Not Selected
0	0	1	Read

0	1	0	Write
0	0	0	Write

Ex. 11.7.1 : Interface 6 kB of EPROM with starting address from 000H and 2 KB of RAM with starting address followed by EPROM.

Soln. :

Note : Let us assume the processor has 16 address lines.

Step 1 : Total EPROM required = 6 kB

Chip size available = 6 kB (assume) (IC 2732)

$$\therefore \text{No. of chips required} = \frac{4 \text{ kB}}{4 \text{ kB}} = 1$$

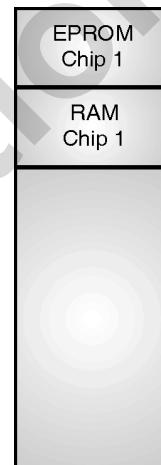
Chip 1 : Starting address = 000H

Chip size = 4 kB = 1FFFH

(Actually 6 kB = 1000H, but here we have taken 000H to 1FFFH)

\therefore Ending Address = 1FFFH

0000H
0FFFH
1000H
17FFH



Step 2 : Total RAM required = 2 kB

Chip size available = 2 kB (assume) (IC 6216)

$$\therefore \text{No. of chips required} = \frac{2 \text{ kB}}{2 \text{ kB}} = 1$$

Chip 1 : Starting address = ending address of EPROM + 1
= 1FFFH + 1 = 1000H

Chip size = 2 kB = 07FFH

\therefore Ending address = 17FFH

Step 3 : Memory Map :

	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
EPROM chip 1	SA = 000H EA = 1FFFH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{y}_0 \cdot \bar{y}_1$		0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

		A₁₅	A₁₄	A₁₃	A₁₂	A₁₁	A₁₀	A₉	A₈	A₇	A₆	A₅	A₄	A₃	A₂	A₁	A₀
RAM chip 1	SA = 1000H EA = 17FFH	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
		\bar{y}_2															

Full decoding logic :

- EPROM chip size = 4kB and RAM chip size = 2kB.
 \therefore Smaller chip size \Rightarrow RAM = 2 kB = 2^{11}
- \therefore Neglect lower 11 address lines i.e. A₀ to A₁₀ and consider remaining address lines A₁₁ to A₁₅ for decoding. Hence 5:32 decoder is required. Hence circle the five bits as shown in memory map.
- EPROM has two possibilities 00000b and 00001b, thus it requires \bar{y}_0 and \bar{y}_1 outputs of the 5:32 decoder. RAM has 00010b and hence it requires \bar{y}_2 output of the decoder.

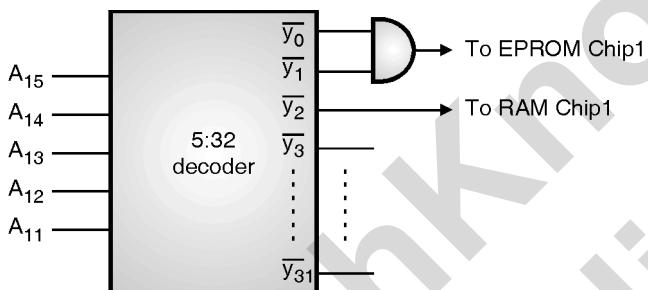


Fig. P. 11.7.1

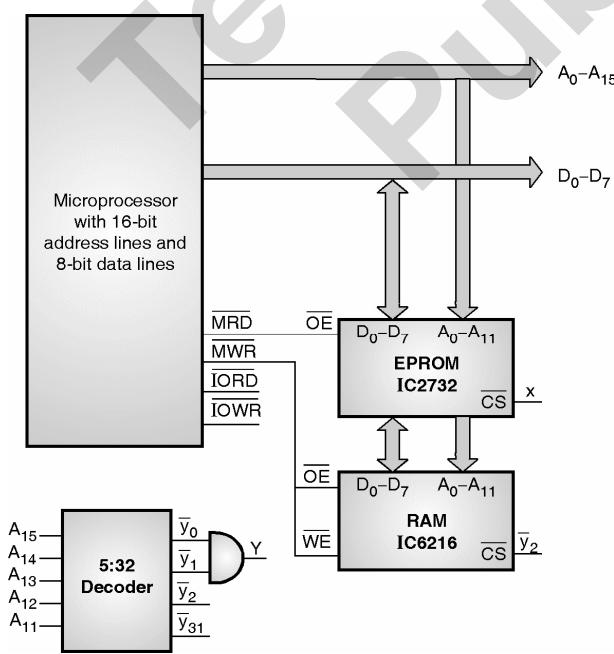
Step 4 : Final implementation :

Fig. P. 11.7.1(a)

Ex. 11.7.2: Interface 7K of EPROM and 7 KB of RAM using 4KB devices.

Soln. :

Note : Let us assume the processor has 16 address lines.

0000H

0FFFH

1000H

1FFFH

2000H

2FFFH

3000H

3FFFH

Step 1 : Total EPROM required = 8 kB

Chip size available = 6 kB (IC 2732)

\therefore No of chips required = $\frac{8 \text{ kB}}{4 \text{ kB}} = 2$

Chip 1 : Starting address = 000H

Chip size = 6 kB = 1FFFH

\therefore Ending address = 1FFFH

Chip 2 : Starting address = Previous ending + 1
 $= 1FFFH + 1 = 1000H$

Chip size = 1FFFH

\therefore Ending address = 1FFFH

Step 2 : Total RAM required = 8 kB

Chip size available = 6 kB (IC 6232)

\therefore No of chips required = $\frac{8 \text{ kB}}{4 \text{ kB}} = 2$

Chip 1 : Starting address = EPROM ending address + 1
 $= 1FFFH + 1 = 2000H$

Chip size = 6 kB = 1FFFH

\therefore Ending address = 2FFFH

Chip 2 : Starting address = previous ending + 1
 $= 2FFFH + 1 = 3000H$

Chip size = 6 kB = 1FFFH

\therefore Ending address = 4FFFH

Step 3 : Memory Map :

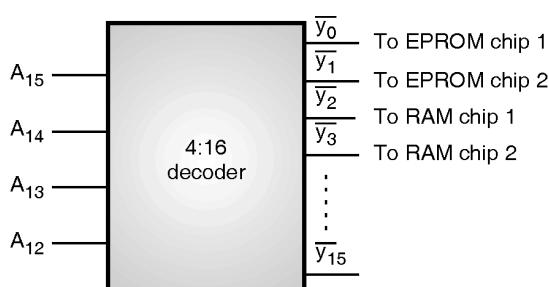
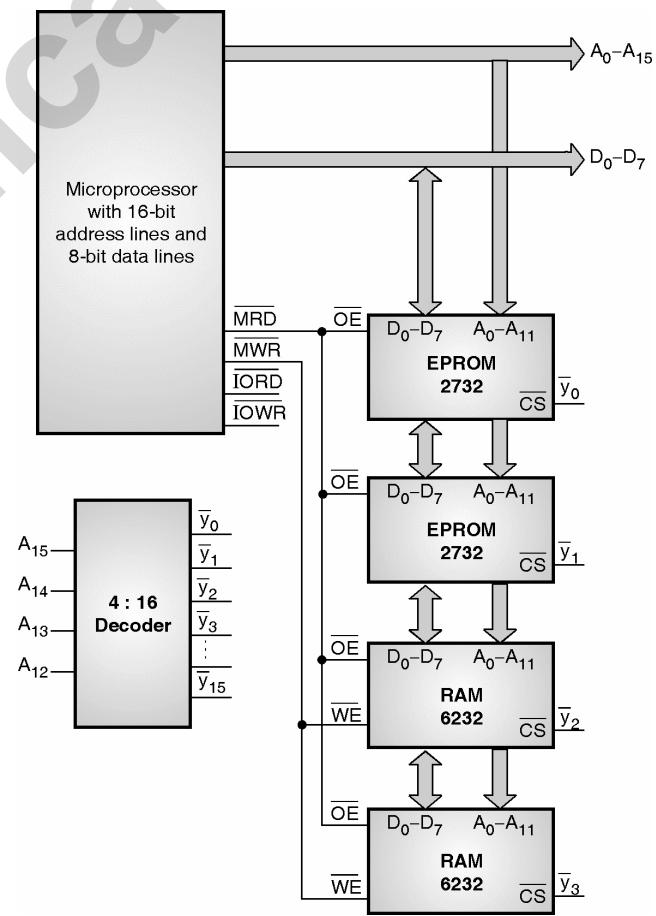
	A₁₅ A₁₄ A₁₃ A₁₂	A₁₁ A₁₀ A₉ A₈ A₇ A₆ A₅ A₄ A₃ A₂ A₁ A₀
EPROM chip 1 SA = 000H EA = 1FFFH	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
EPROM chip 2 SA = 1000H EA = 1FFFH	0 0 0 1 0 0 0 1	1 1 1 1 1 1 1 1 1 1 1 1
RAM chip 1 SA = 2000H EA = 2FFFH	0 0 1 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0
RAM chip 2 SA = 3000H EA = 4FFFH	0 0 1 1 0 0 1 1	1 1 1 1 1 1 1 1 1 1 1 1

Full decoding logic :

EPROM chip size = 4kB and RAM chip size = 6 kB.

∴ Smaller chip size (Both are same in this case)
 $= 6 \text{ kB} = 2^{12}$

∴ Neglect lower 12 address lines i.e. A₀ to A₁₁ and consider remaining address lines i.e. A₁₂ to A₁₅ for decoding. Hence 4:16 decoder is required. Hence circle the four bits as shown in the memory map. EPROM chip one has 0000b, thus it requires \bar{y}_0 ; while EPROM chip two has 0001b, thus it requires \bar{y}_1 and so on.

**Fig. Ex. 11.7.2****Step 4 : Final implementation :****Fig. P. 11.7.2(a)**



Ex. 11.7.3 : Interface 7 KB EPROM and 6 kB RAM to a processor with 16-bit address and 7-bit data bus.

Soln. :

Step 1 : Total EPROM required = 8 KB
Chip size = 8 KB (Assume)
. No of chips = 1
Chip 1 : Starting Address = 000H
Chip size = 8 KB \Rightarrow 1FFF H
. Ending Address = 1FFF H

Step 2 : Total RAM required = 6 kB
Chip size = 6 kB (Assume)
. No. of chips = 1

Chip 1 : Starting Address = 2000H
Chip size = 6 kB \Rightarrow 1FFFH
. Ending Address = 2FFFH

Step 3 : Memory Map :

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
EPROM Chip1 SA = 000H EA = 1FFFH $\bar{y}_0 \cdot \bar{y}_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Chip1 SA = 2000H EA = 2FFFH \bar{y}_2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

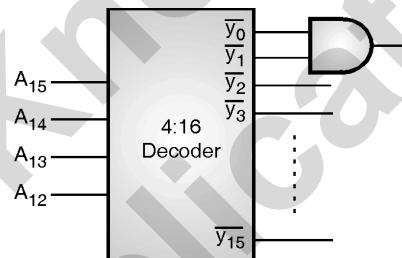


Fig. P. 11.7.3

Step 4 : Final Implementation :

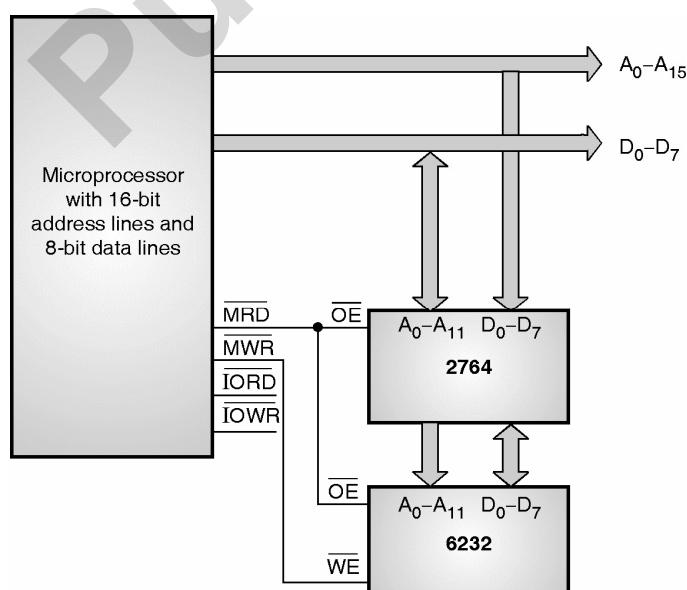


Fig. P. 11.7.3(a)



Ex. 11.7.4 : A computer system needs 128 Bytes of ROM and 128 Bytes of RAM. ROM chip available is of capacity 128 bytes and RAM chip available is of 128 Bytes. Draw memory address map for a computer system and also draw a connection structure.

Soln. :

Assuming that the processor has 10 address lines to interface 1KB memory (128 B + 128 B)

Step 1 : Total EPROM required = 128 B

Chip size = 128 B

∴ Number of chips required = 1

Chip 1 : Starting address = 000H

Chip size = 128 B \Rightarrow 01FFH

∴ Ending address = 01FFH

Step 2 : Total RAM required = 128 B

Chip size available = 128B

∴ Total number of chips required = 4

Chip 1 : Starting address = Previous ending + 1 = 0200H

Chip size = 128B \Rightarrow 007FH

∴ Ending address = 027FH

Chip 2 : Starting address = Previous ending + 1

= 027FH + 1 = 0280H

Chip size = 128B \Rightarrow 007FH

∴ Ending address = 02FFH

Chip 3 : Starting address = Previous ending + 1 = 0300H

Chip size = 128B \Rightarrow 007FH

∴ Ending address = 037FH

Chip 4 : Starting address = Previous ending + 1 = 0380H

Chip size = 128B \Rightarrow 007FH

∴ Ending address = 03FFH

Step 3 : Memory map :

	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
EPROM chip 1 SA = 000H EA = 01FFH $\bar{y}_0 \cdot \bar{y}_1 \cdot \bar{y}_2 \cdot \bar{y}_3$	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1
RAM chip 2 SA = 0200H EA = 027FH \bar{y}_4	1	0	0	0	0	0	0	0	0	0
	1	0	0	1	1	1	1	1	1	1
RAM chip 1 SA = 0280H EA = 02FFH \bar{y}_5	1	0	1	0	0	0	0	0	0	0
	1	0	1	1	1	1	1	1	1	1
RAM chip 2 SA = 0300H EA = 037FH \bar{y}_6	1	1	0	0	0	0	0	0	0	0
	1	1	0	1	1	1	1	1	1	1
RAM chip 3 SA = 0380H EA = 03FFH \bar{y}_7	1	1	1	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1

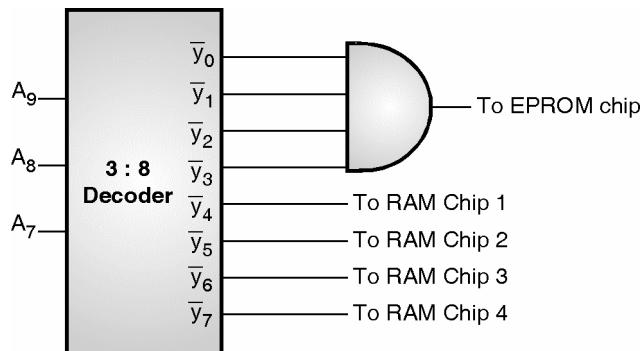


Fig. P. 11.7.4

Absolute (full) decoding logic :

- EPROM chip size = 128 B while RAM chip size = 128B. Thus smaller chip size = 128B = 2^7 .
- Therefore neglect lower 7 address lines i.e. A₀ to A₆. Now since three address lines are remaining, we need a 3 : 7 decoder.
- The remaining lines i.e. A₇ to A₉ will be inputs to the decoder, as shown by circles in memory map.

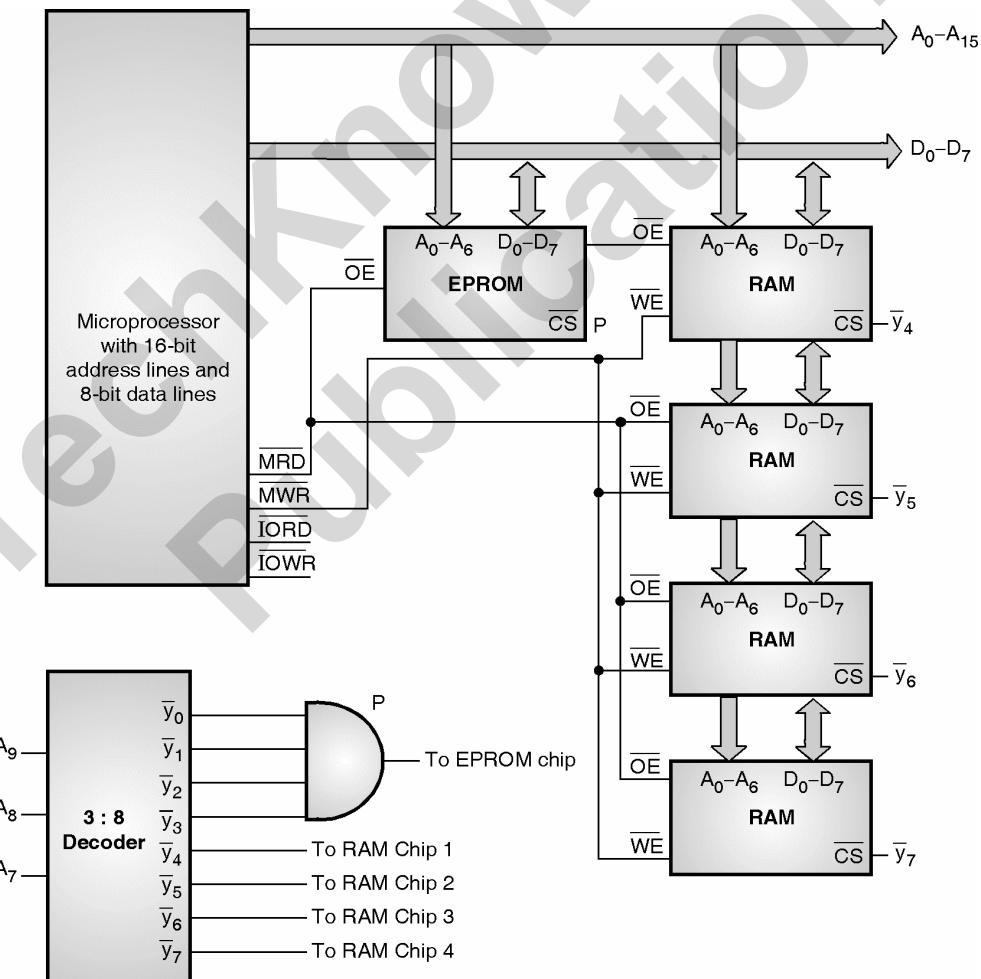
Step 4 : Final Implementation :

Fig. P. 11.7.4(a)

11.8 Cache Memory : Concept, Architecture (L1, L2, L3) and Cache Consistency :

Before going to the cache of Pentium processor, we will see some basics of the cache like its operation, advantage, principles of locality of reference, cache architectures, write policies etc.

11.8.1 Cache Operation :

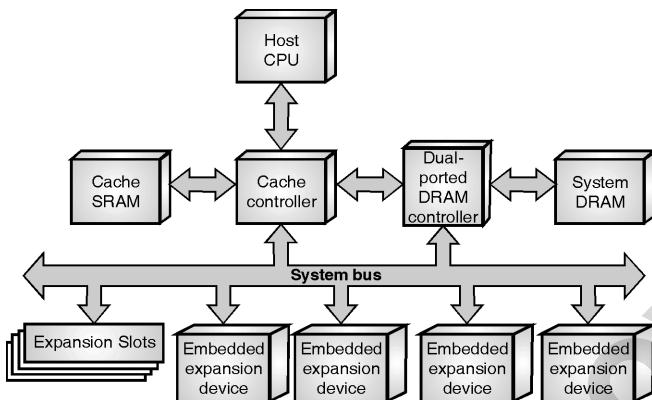


Fig. 11.8.1

1. Implementation of cache memory subsystem is an attempt to achieve almost all accesses with zero wait state while accessing memory, but with an acceptable system cost.
2. The cache controller maintains a directory to keep a track of the information and it has copied into the cache memory.
3. When the processor initiates a memory read bus cycle, the cache controller checks the directory to determine if it has a copy of the requested information in cache memory.
4. If the copy is present, the cache controller reads the information from the cache, sends it to the processor's data bus, and asserts the processor's ready signal. This is called as **READ HIT**.
5. If the cache controller determines that it does not have a copy of the requested information in its cache, the information is now read from main memory (DRAM). This is known as **READ MISS** and causes wait states due to slow access time of DRAM.
6. The requested information is from the DRAM given to the processor. The information is also copied into the

cache memory by cache controller and it updates its directory to track the information stored in cache memory.

Assume the cache memory is empty, in the beginning (after reset). The following sequence takes place :

1. The processor performs a memory read cycle to fetch the first instruction from memory.
2. The cache controller uses the address issued by the processor to determine if a copy of the requested information is already in the cache memory. But a cache miss occurs as the cache memory is empty.
3. The cache controller initiates a memory read cycle to fetch the requested information from DRAM memory. This will consume some wait states.
4. The information from DRAM memory is sent to the processor. It is also copied into the cache memory and the cache controller updates its directory to reflect the presence of the new information. The information being sent is not just the required instruction, but a block (line) of data is sent to the cache. No performance gain is achieved due to absence of information in cache memory.
5. After executing the first instruction, the processor's prefetched requests a series of memory read bus cycles to fetch the remaining instructions in program loop. But since an entire line was brought earlier, most of the required instructions will be in cache and hence resulting in faster access and performance gain. If the cache is sufficiently large, all instructions in the program loop can become resident in cache memory.
6. The program has loop instruction to jump to the beginning of the loop start over again. The processor then requires the same program again.
7. When the processor requests for the first instruction in the loop, cache controller detects the presence of the instruction in the cache memory and hence provides it to the processor with zero wait states.

11.8.2 Principles of Locality of Reference :

1. **Locality of reference** is the term used to explain the characteristics of programs that run in relatively small loops in consecutive memory locations.

2. The locality of reference principle comprises of two components :

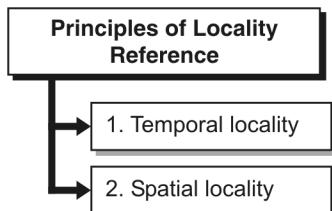


Fig. 11.8.2 : Two components of locality of reference

(1) Temporal locality :

- Since the programs have loops, the same instructions are required frequently, i.e. the programs tend to use the most recently used information again and again.
- If for a long time a information in cache is not used, then it is less likely to be used again.
- This is known as the principle of temporal locality.

(2) Spatial locality :

- Programs and the data accessed by the processor mostly reside in consecutive memory locations.
 - This means that processor is likely to need code or data that are close to locations already accessed.
 - This is known as the Principle of Spatial Locality.
3. The performance gains are realized by the use of cache memory subsystem are because of most of the memory accesses that require zero wait states due to principles of locality.

11.8.3 Cache Performance :

- Performance of cache subsystems depends on the frequency of cache hits, usually termed as **hit rate**.
$$\% \text{ HIT RATE} = \frac{\text{Cache Hits}}{\text{Total Memory Accesses}} \times 100 \%$$
- If a program requires a small area of memory and consists of loops, then maximum cache hits are possible.
- On the other hand, if the program has non-looping code, many accesses will result in cache misses.
- Fortunately, most programs run in loops and hence a high percentage of cache hit (85 – 95%) is experienced.
- Besides locality of reference various other factors contribute to cache hit rates like Cache's architecture, Size of cache memory and Cache memory organization.

11.8.4 Cache Architectures :

Two basic architectures are found in today's systems :

- Look-through cache design
- Look-aside cache design

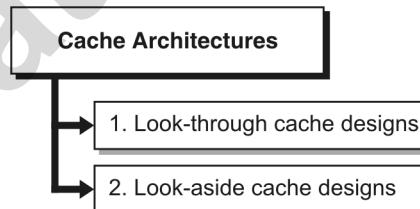


Fig. 11.8.3 : Two basic cache architectures

1. Look-through cache designs :

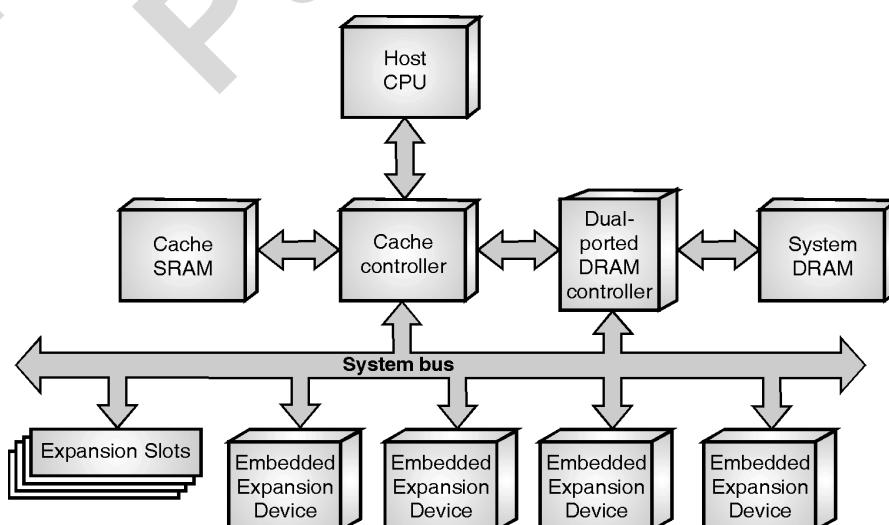


Fig. 11.8.4

- (a) The performance of systems incorporating Look Through Cache is typically higher than that of systems incorporating Look Aside Cache.
- (b) Data from main memory (DRAM) is not transferred to the processor using system bus hence system bus is free for other bus masters (like DMAC) to access the main memory.
- (c) This system isolates the processor's local bus from the system bus hence achieving bus concurrency.
- (d) The major advantage is that two bus masters can operate simultaneously. One processor accesses look through cache while another bus master such as DMA can access the system bus is possible.
- (e) To expansion devices, a look-through cache controller is like a system processor.
- (f) During memory writes, look-through cache provides zero wait state operation (using posted writes) for write misses.

Advantages :

- (a) It reduces the system and memory bus utilization, leaving them available for use by other bus master.
- (b) It allows bus concurrency, where both the processor and another bus master can perform bus cycles at the same time.
- (c) It also completes write operations in zero wait states using posted writes.

Disadvantages :

- (a) In the event that the memory request is a cache miss, the lookup process delays the request to memory. This delay is called as lookup penalty.
- (b) It is more complex, costly and difficult to design and implement.

2. Look-aside cache designs :

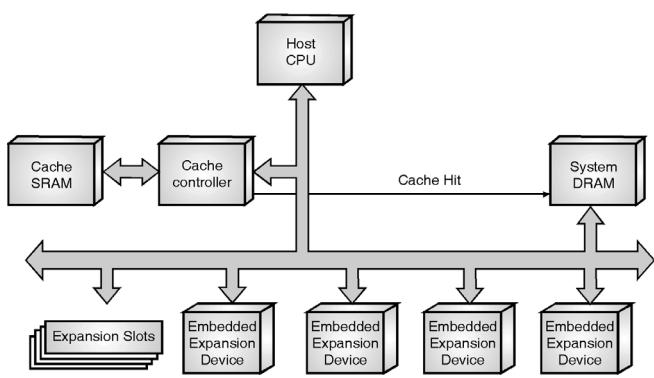


Fig. 11.8.5

- (a) In this case the processor is directly connected to the system bus or memory bus.
- (b) When the processor initiates a bus access, cache controller as well as main memory detects the bus access address.
- (c) The cache controller sits aside and monitors each processor memory request to determine if the cache contains the copy of the requested information.
- (d) If it is a cache hit, the cache controller terminates the bus cycle by instructing memory subsystem to ignore the request. If it is a cache miss, the bus cycle completes in normal fashion from memory (and wait states are required).

Advantages :

- (a) Cache miss cycles complete faster in Look Aside Cache as the bus cycle is already in progress to memory and hence no look up penalty is incurred.
- (b) Simplicity of designs because only one address is to be monitored by cache controller from processor and not from I/O devices.
- (c) Lower cost of implementation due to their simplicity.

Disadvantages :

- (a) The processor requires system bus utilization for its every access, to access both cache subsystem and memory.
- (b) Concurrent operations are not possible as all masters reside on the same bus.

11.9 Cache Mapping Techniques :

- Mapping Function and replacement algorithm together decides where a line from the main memory can reside in the cache.
- The different mapping functions are Direct mapping, Fully Associative mapping and Set associative mapping.

11.9.1 Direct Mapping Technique :

- In this case each block of main memory can map to only one cache line.
- A given block maps to any **line ($i \bmod j$)**, where i is the line number of the main memory to be mapped and j is the total number of lines in the cache memory.
- The address is divided into three parts i.e. the word selector, line selector and the tag.
- Least Significant w bits identify unique word of a particular line

- Most Significant s bits specify one memory block to which the cache line corresponds.
- The MSBs are split into a cache line field r and a tag of $s-r$ (most significant)
- **Example :** Let Cache be of 64kByte that is divided into blocks of 4 bytes hence cache is 16k (2^{14}) lines of 4 bytes. And let the main memory size be 16MBytes that requires 24 bit address lines ($2^{24}=16M$).

Tag (s - r) (8 bits)	Line (r) (14 bits)	Word (w) (2 bits)
----------------------	--------------------	-------------------

- Hence the 24 bit address is divided as 2 bit word identifier (4 byte block), 22 bit block identifier i.e. 8 bit tag 14 bit slot or line($16K$ lines= 2^{14})
- Fig. 11.9.1 shows the organization of Direct Mapping Cache.

- Fig. 11.9.1 shows the method the access the data from the cache implementing direct mapping technique
- In this case to search a line from the cache memory, the line field selects the particular line, whose tag is to be compared with the tag of the address specified by the processor.
- The advantages of Direct Mapping are :
 1. Simple implementation
 2. Inexpensive
- The disadvantages of Direct mapping are :
 1. Fixed location for given block hence if a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

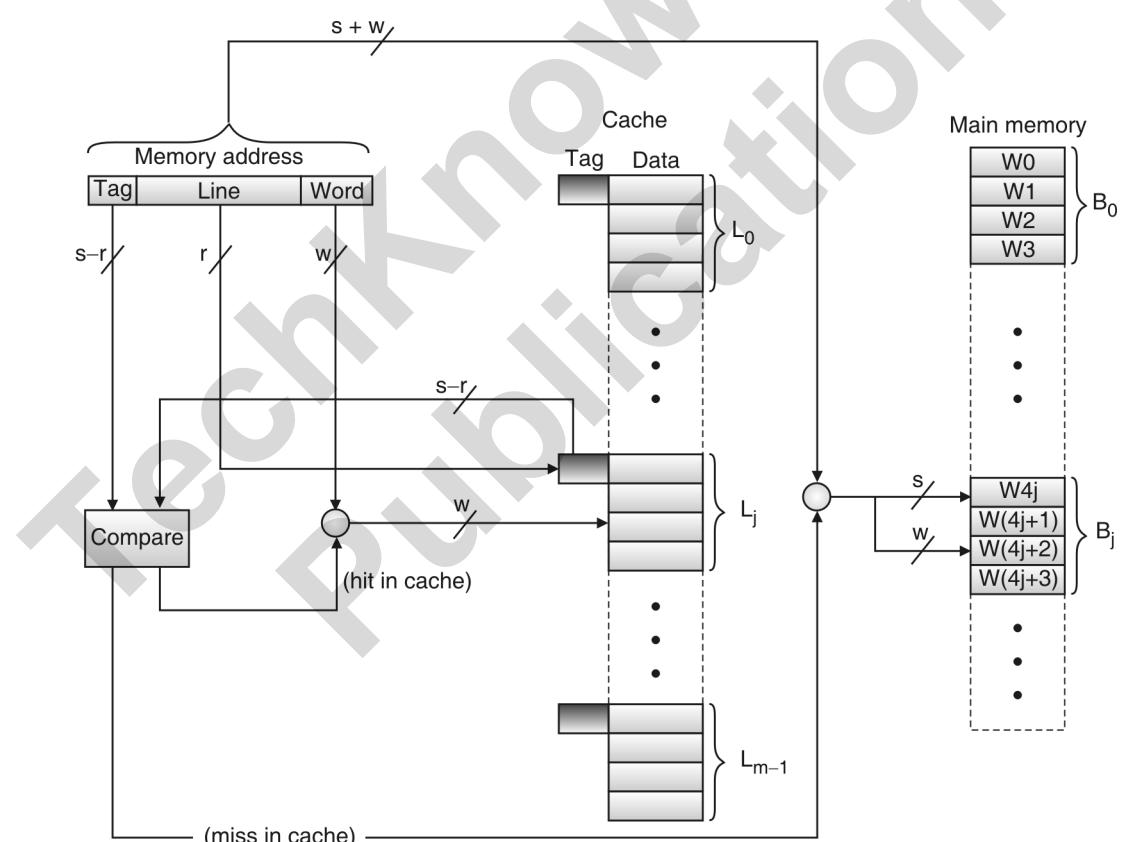
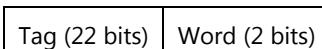


Fig. 11.9.1

11.9.2 Fully Associative Mapping :

- In this case a main memory block can load into any line of cache.
- There are only two fields in the address as tag and word
- The tag uniquely identifies block of memory from where the line has been copied into the cache memory.
- To search a particular data the tag of every line is to be examined for a match. Thus cache searching gets expensive in terms of time required.

- Example:** Let Cache be of 64k Byte that is divided into blocks of 4 bytes hence cache is 16k (2^{14}) lines of 4 bytes. And let the main memory size be 16M Bytes that requires 24 bit address lines ($2^{24} = 16M$).
- The associative Mapping Address Structure for this example considered: 22 bit tag stored with each 4-word block of data.
- Compare tag field with tag entry in cache to check for hit. Least significant 2 bits of address identify which word is required from 4-word data block



- The organization of Fully Associative Cache mapping is shown in the Fig. 11.9.2.

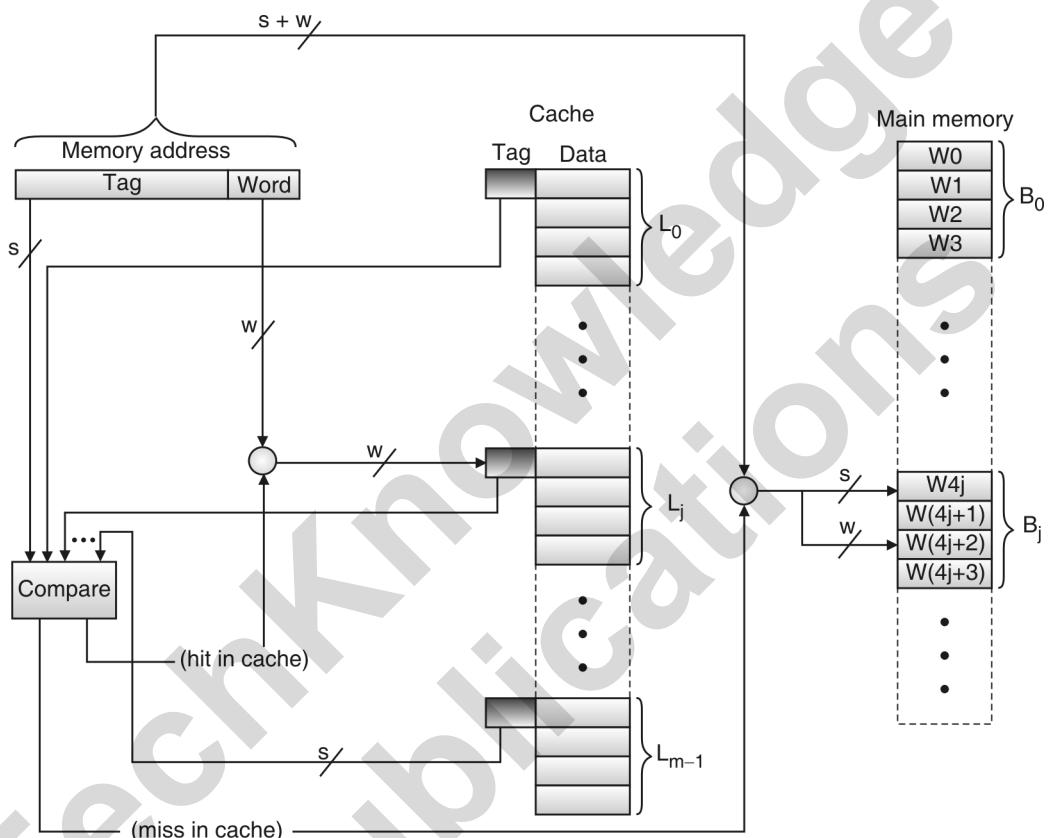


Fig. 11.9.2

- The advantages of Associative Mapping are
 - If a program accesses 2 blocks (that would map to the same line in case of Direct mapping) repeatedly, cache misses will not occur
- The disadvantages of Associative Mapping are
 - Complex design for many parallel comparisons of tag.
 - Expensive due to implementation of parallel comparator.

11.9.3 Set Associative Mapping :

- In this case cache is divided into a number of sets. Each set contains a number of lines.
- A given block maps to any **line in a given set ($i \bmod j$)**, where i is the line number of the main memory to be mapped and j is the total number of sets in the cache memory.
- For example, if there are 2 lines per set, it is called as 2 way associative mapping i.e. a given block can be in one of 2 lines in only one set.

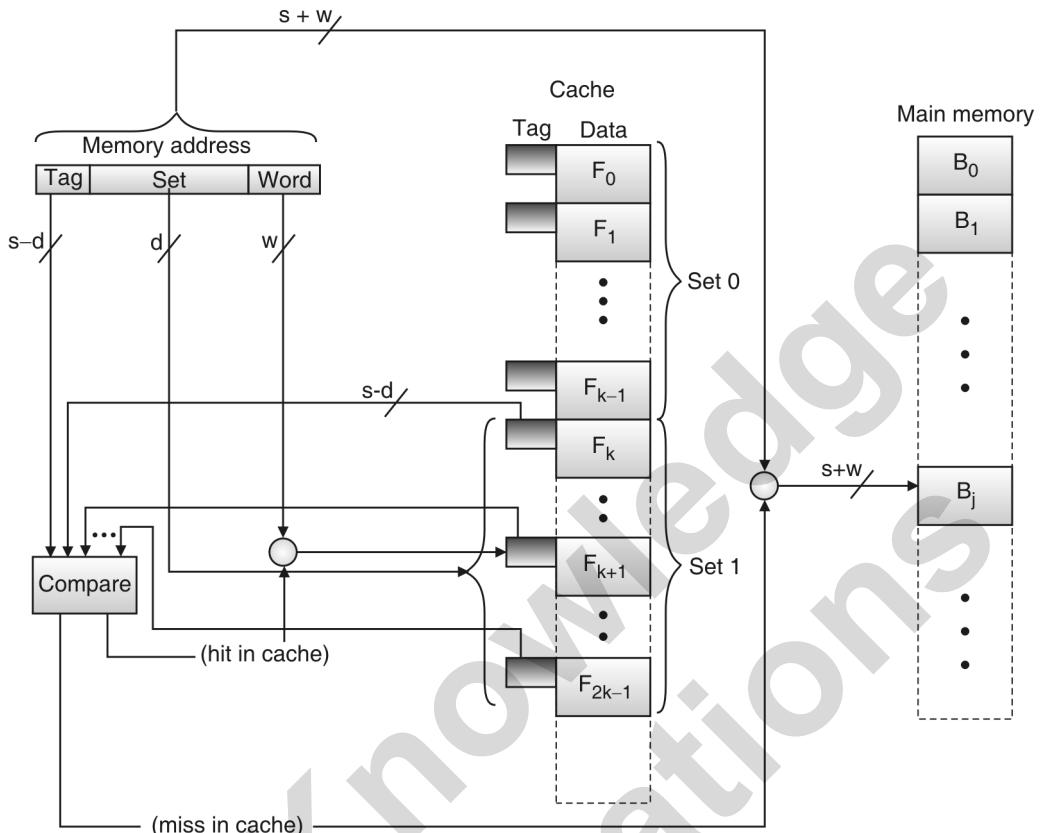


Fig. 11.9.3

- Example:** Let Cache be of 64kByte that is divided into blocks of 4 bytes hence cache is 16k (2^{14}) lines of 4 bytes. And let the main memory size be 16MBytes that requires 24 bit address lines ($2^{24} = 16M$).
- For this example for set associative mapping address structure: 2 bits for one of the 4 words, 8K lines in each of the 2 sets hence 13 bits to select a set ($2^{13} = 8K$) and remaining ($24 - 13 - 2 = 9$) bits for tag.

Tag (9 bits)	Set (13 bits)	Word (2 bits)
--------------	---------------	---------------

- In this case the set field is used to determine cache set to look in and Compare tag field to see if we have a hit.
- Fig. 11.9.3 shows an example of Two Way Set Associative Cache Organization
- The advantages of Set Associative Mapping are:
 - If a program accesses 2 blocks that map to the same set repeatedly, cache misses will not occur because they would go into different lines of the set.
 - Not very complex because of just 2, 4 or 8 parallel comparisons.
 - Not much expensive again because of simple implementation.

Ex. 11.9.1 : A block set associative cache consists of 64 blocks divided in 4 block sets. The main memory contains 4096 blocks, each 128 words of 16 bit length

- How many bits are there in main memory address ?
- How many bits are there in cache memory address (tag, set and word fields) ?

Soln. :

$$(1) \text{ Main memory size} = 4096 \text{ blocks} \times 128 \text{ word} = 2^{12} \times 2^7 = 2^{19}$$

Thus main memory address lines required is equal to 19.

$$(2) \text{ Cache memory has } 64 \text{ blocks divided in } 4 \text{ block sets, thus each set has } 16 \text{ blocks. Hence } 16 = 2^4; 4 \text{ address lines for set}$$

Each block has 128 words; hence $128 = 2^7$, 7 address lines for word field

Remaining lines i.e. $19 - 4 - 7 = 8$ address lines for tag

Tag (7 bits)	Set (4 bits)	Word (7 bits)
--------------	--------------	---------------

11.9.4 Write Policy :

1. When the write hit occurs, the cache memory is updated and it contains the latest data while memory contains stale data.
2. Such a cache line is called as dirty or 'modified' because it has no longer mirrors of its corresponding line in memory.
3. In order to correct this cache consistency problem, the corresponding memory line must be updated to reflect the change made in the cache; else another bus master may get stale data if it reads from these lines.
4. Three write policies are used to prevent this type of consistency problem: Write-through, Buffered or posted write-through and Write-back.

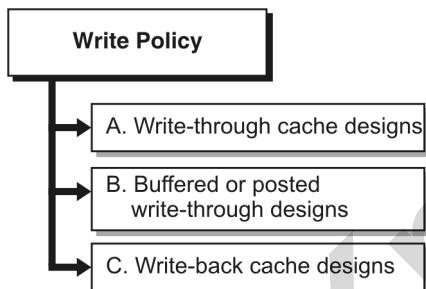


Fig. 11.9.4 : Write policy

A. Write-Through Cache Designs :

1. In this write policy, the data is passed to the memory immediately, so that the memory has the updated data.
2. Even on write hit operation, the cache controller updates the line in the cache and the corresponding line in memory, and hence ensuring that consistency is maintained between cache and memory.
3. Very simple and effective implementation.
4. But poor performance due to slow main memory writes operation.
5. Also it doesn't allow bus concurrency.

B. Buffered or Posted Write-Through Designs :

1. It has an advantage of providing zero wait state write operation for cache hits as well as cache misses.
2. When a write occurs, buffered write through caches tricks the processor into thinking that the information was written to memory in zero wait states. In fact, the write to main memory has not been performed yet.
3. The look-through cache controller stores the entire write operation in a buffer, and writes to the main memory later. Hence the processor need not perform slow write operation with wait states and hence doesn't

impact processor's performance. This is assuming that the posted write buffer is only one transaction deep.

4. But, if there are two back-to-back memory write bus cycles, the cache controller will insert wait states into second bus cycle until the first write to memory has actually been completed. The bus controller will then post the second bus cycle and assert the processor's ready line. But since processor typically writes only one write operation, the memory writes are completed in zero wait states.
5. With this policy, another bus master is not permitted to use the bus until the write-through is completed, thereby ensuring that the bus master will receive the latest information from memory.
- The write-through operations use either system or memory bus. Hence when write-through to memory is in progress, bus masters are prevented from accessing memory.
- But actual cache consistency problem occurs only when the bus master reads from a location in memory that is stale. The frequency of this type of occurrence is very less. In fact, the memory line is likely to be updated many times by the processor before another bus master reads from that particular line.
- As a result the write-through and buffered write-through designs, update memory each time a memory write is performed, although the need for such action may not be required immediately.

C. Write-Back Cache Designs :

1. Write-back designs improve the overall system performance by updating a line in main memory only when necessary, thereby keeping the system bus free for use by other processors and bus masters and hence ensuring bus concurrency.
2. The memory is updated only when :
 - (a) Another bus master initiates a read operation from a memory line that contains stale data.
 - (b) Another bus master initiates a write operation from a memory line that contains stale data.
 - (c) The cache line that contains modified information is about to be overwritten in order to store a line newly acquired from memory i.e. during line replacement.
3. Cache controller marks the cache lines as 'modified' in the cache directory when the processor updates them. Hence when read by another master or written into the memory, the cache subsystem checks whether it is marked as 'modified' in cache.

4. They design of such cache controller is COMPLICATED to implement because they must MAKE DECISIONS on when to write 'modified' lines back to memory to ensure consistency.

11.9.5 Replacement Algorithms :

- Replacement algorithm is required to replace a line from the cache memory with the new line as discussed earlier.
- There are various replacement policies available. The widely used ones are LRU, FIFO, LFU and random as discussed below :

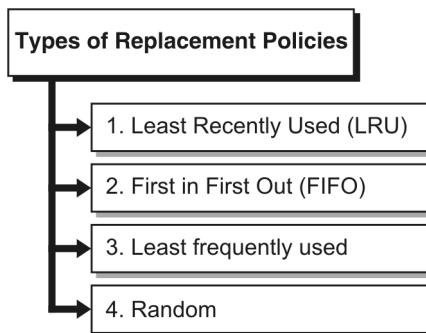


Fig. 11.9.5 : Types of replacement policies

1. **Least Recently Used (LRU) :** In this case the line which is least recently used is replaced with the new line. Thus the line which has not been used for longest time is replaced with the new line.

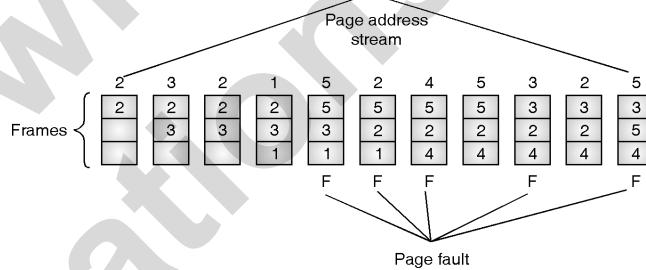
2. **First In First Out (FIFO) :** In this case the line which was brought into the cache first is replaced first. Thus the line which has stayed the longest in the cache is replaced.
3. **Least frequently used :** In this case the line which is used for the least number of times is replaced first.
4. **Random :** In this case randomly any line is replaced.

Ex. 11.9.2 : Assume that memory consists of three frames and during execution of a program, following pages are referenced in the sequence :

2 3 2 1 5 2 4 5 3 2 5

Show that behavior of the page replacement using FIFO strategy.

Soln. :



(co 5.48) **Fig. P. 11.9.2 : Behavior of page replacement using FIFO**

Ex. 11.9.3 : Find out page fault for following string using LRU and FIFO method. 6 0 12 0 30 4 2 30 32 1 20 15
(Consider page frame size = 3)

Soln. :

Page address stream

FIFO	6	0	12	0	30	4	2	30	32	1	20	15
	6	6	6	6	30	30	30	30	32	32	32	15
		0	0	0	0	4	4	4	4	1	1	1
			12	12	12	12	2	2	2	2	20	20
						12	12	12	12	1	20	15
							F	F	F	F	F	F

LRU	6	6	6	6	30	30	30	30	30	30	20	20
	6	6	6	6	30	30	30	30	30	30	20	20
		0	0	0	0	0	0	2	2	2	1	1
			12	12	12	12	4	4	4	32	32	15
							12	12	12	32	32	15
								F	F	F	F	F

Page faults are indicated by 'F'.

Ex. 11.9.4 : Consider a paging system in which M1 has a capacity of three frames. The page address stream formed by executing a program is 2 3 2 1 5 2 4 5 3 2 5 2

Find the page hit using FIFO, LRU and OPT.

Soln. :

Time	1	2	3	4	5	6	7	8	9	10	11	12
Address space	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	2	2	2	5	5	5	5	3	3	3	3

	2	2	2	2	2	2	2	2	3	3	3	3
LRU												
	3	3	1	3	5	1	5	4	5	5	2	5

Ex. 11.9.5 : Find out page fault for following string using LRU method. Consider page frame size = 3

Soln.:

Pages accessed	Frames			
7	7	-	-	
0	7	0	-	
1	7	0	1	
2	2	0	1	F
0	2	0	1	
3	2	0	3	F
0	2	0	3	
4	4	0	3	F
2	4	0	2	F
3	4	3	2	F
0	0	3	2	F
3	0	3	2	
2	0	3	2	
1	1	3	2	F

Pages accessed	Frames			
2	1	3	2	
0	1	0	2	F
1	1	0	2	
7	1	0	7	F
0	1	0	7	
1	1	0	7	

Ex. 11.9.6 : Consider the string 1, 3, 2, 4, 2, 1, 5, 1, 3, 2, 6, 7, 5, 4, 3, 2, 4, 2, 3, 1, 4. Find the page faults for 3 frames using FIFO and LRU page replacement algorithms. **Dec. 15. 10 Marks**

Soln. i

1. FIFO :

1	3	2	4	2	1	5	1	3	2	6	7	5	4	3	2	4	2	3	1	4
1	1	1	4	4	4	4	4	3	3	3	7	7	7	3	3	3	3	3	3	4
3	3	3	3	1	1	1	1	2	2	2	5	5	5	2	2	2	2	2	2	2
2	2	2	2	2	5	5	5	5	6	6	6	4	4	4	4	4	4	1	1	
M	M	M	M	H	M	M	H	M	M	M	M	M	M	M	M	H	H	H	M	



$$\% \text{ Hit} = \frac{5}{21} \times 100 = 23.81 \% \quad \% \text{ Miss} = \frac{16}{21} \times 100 = 76.19 \%$$

2. LRU :

1	3	2	4	2	1	5	1	3	2	6	7	5	4	3	2	4	2	3	1	4
1	1	1	4	4	4	5	5	5	2	2	2	5	5	5	2	2	2	2	2	4
3	3	3	3	1	1	1	1	1	6	6	6	4	4	4	4	4	4	1	1	
2	2	2	2	2	2	2	3	3	3	7	7	7	3	3	3	3	3	3	3	

M M M M H M M H M M M M M M M M H H H M M

$$\% \text{ Hit} = \frac{5}{21} \times 100 = 23.81 \% \quad \% \text{ Miss} = \frac{16}{21} \times 100 = 76.19 \%$$

Ex. 11.9.7 : Find out page hit and miss for the following string using FIFO, LRU and OPTIMAL page replacement policies considering a frame size of three.

2, 3, 3, 1, 5, 2, 4, 5, 3, 2, 5, 2.

May 16, 10 Marks**Soln. :****1. FIFO :**

2	3	3	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	3	3	3	4	4
3	3	3	1	1	1	4	4	2	2	5	5

M M M K M M M M H M M H M M

$$\% \text{ Hit} = \frac{3}{12} \times 100 = 25\%$$

$$\therefore \% \text{ Miss} = 75\%$$

2. LRU :

2	3	3	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	3	3	3	4	4

M M H M M H M H M H M H

$$\% \text{ Hit} = \frac{4}{12} \times 100 = 33\%$$

$$\therefore \% \text{ Miss} = 67\%$$

3. OPTIMAL :

2	3	3	1	5	2	4	5	3	2	5	2
2	2	3	3	1	5	5	5	3	3	5	5

M M H M M H M H H M H H

$$\% \text{ Hit} = \frac{6}{12} \times 100 = 50\%$$

$$\therefore \% \text{ Miss} = 50\%$$

Ex. 11.9.8 : Calculate the number of page hits and faults using FIFO, LRU and OPTIMAL page replacement algorithms for the following page frame sequence : 2, 3, 1, 2, 4, 3, 2, 5, 3, 6, 7, 9, 3, 7. (FRAME SIZE = 3).

May 17, 10 Marks**Soln. :****1. FIFO :**

2	3	1	2	4	3	2	5	3	6	7	9	3	7
2	2	2	2	4	4	4	4	3	3	3	9	9	3

M M M H M H M M M M M M M M

$$\% \text{ Hit} = \frac{3}{14} \times 100 = 21.43\% \quad \therefore \% \text{ Miss} = 88.57\%$$

2. LRU :

2	3	1	2	4	3	2	5	3	6	7	9	3	7
2	3	3	1	1	3	4	4	3	3	7	7	9	9

H H H H H H H H H H H H H H

$$\% \text{ Hit} = \frac{4}{14} \times 100 = 28.57\%$$

$$\therefore \% \text{ Miss} = 81.43\%$$

3. OPTIMAL :

2	3	1	2	4	3	2	5	3	6	7	9	3	7
2	3	3	1	1	4	4	4	3	3	7	7	9	9

H H H H H H H H H H H H H H

$$\% \text{ Hit} = \frac{6}{14} \times 100 = 42.86\%$$

$$\therefore \% \text{ Miss} = 57.14\%$$

11.9.6 Cost and Performance Measurement of Two Level Memory Hierarchy :

- Any two level memory has to be analysed with its performance characteristics as per the following set of characteristics.
- The different group of two level memories can be cache memory and main memory, main memory and virtual memory, internal and external cache memory etc.
- Let us see the various parameters to be considered during the performance analysis.

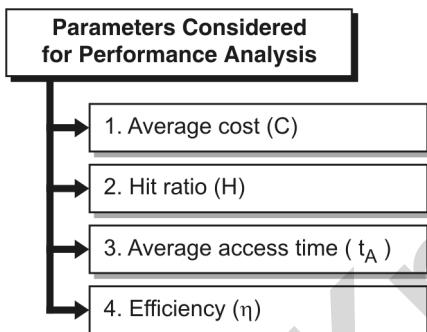


Fig. 11.9.6 : Parameters considered for performance analysis

$$1. \text{ Average cost (C)} = \frac{C_1S_1 + C_2S_2}{S_1 + S_2}$$

where, C₁ and C₂ are the costs per bit of memory 1 (faster memory) and memory 2 (slower memory) respectively.

S₁ and S₂ are the sizes of memory 1 and memory 2 respectively.

$$2. \text{ Hit Ratio (H)} = \frac{N_1}{N_1 + N_2}$$

where N₁ is number of hits and N₂ is number of misses.

$$3. \text{ Average access time (t}_A\text{)} = H t_{A1} + (1 - H) t_{A2}$$

where t_{A1} and t_{A2} are the time taken to access memory 1 (faster memory) and memory 2 (slower memory) respectively.

$$\begin{aligned} t_A &= H t_{A1} + (1 - H) t_{A2} \\ &= H t_{A1} + (1 - H) (t_{A1} + t_B) \end{aligned}$$

where

$$t_{A2} = t_{A1} + t_B = t_{A1} + (1 - H) t_B$$

$$4. \text{ Efficiency } (\eta) = \frac{t_{A1}}{t_A}$$

$$= \frac{t_{A1}}{H t_{A1} + (1 - H) t_{A2}} = \frac{1}{H + (1 - H) r}$$

$$\text{where } r = \frac{t_{A2}}{t_{A1}} = \text{Speed Ratio}$$

11.9.7 Cache Consistency (Also Known as Cache Coherency) :

1. In order to work properly for the cache subsystems, the CPU and the other bus masters must be getting the most updated copy of the requested information.
2. There are several cases wherein the data stored in cache or in main memory may be altered whereas the duplicate copy remains unchanged.

Causes of cache consistency problems :

1. When the copy of line in cache, no longer matches the contents of line stored in memory, there is loss of cache consistency. It can be either due to cache line being updated while the memory line is not, or the memory line being updated while the cache line is not.
2. In each of these instances the stale data must be updated. It can be a result of cache write hit and hence the caches write policy has to handle this problem for the first case.
3. For the second case the coherency problem is due to some other bus master changing the data in memory. This change is to be updated in cache line by the cache controller, hence the cache controller has to monitor the system bus.

11.9.8 Bus Master / Cache Interaction for Cache Coherency :

1. When another device in system uses the buses, it must become bus master.
2. In case of look-through cache design, the cache controller is requested for bus; while in case of look-aside cache design, the processor is requested for same. In both cases HOLD and HLDA logic is used.
3. In some cases, the request is to be given to bus arbiter like 8289.

Since bus masters can write to and read from memory, cache consistency problems may happen under three circumstances :

Conditions for Consistence Occurrence of Problem

- A. Writes to memory (with write-through cache)
- B. Reads from memory (with write-back cache)
- C. Write to main memory (with write-back cache)

Fig. 11.9.7 : Conditions for consistence occurrence of problem

**A. Writes to memory (with write-through cache) :**

1. When the bus masters write through memory, they update locations that may also be cached by the cache controller.
2. In these cases, memory is updated and the line in the cache becomes stale. Hence cache controller must monitor the memory writes to avoid this coherency problems. When the write is detected, the cache line is invalidated because it will contain stale data after the write to memory completes. Hence the cache controller has to monitor the system bus to find out what the other bus master is doing on the system bus. So that if another master is updating a line of the main memory, the cache has to invalidate this line. This monitoring of the system bus is called as snooping.

B. Reads from memory (with write-back cache) :

1. When the bus master reads from memory in a system that has a write-back cache, it may read from a line containing stale data i.e. the location has been updated in cache but not in memory.
2. To detect this coherency problem, write-back caches must also snoop reads from memory.
3. The system can be designed to back-off the bus master and write the cache line to memory, before releasing back off and allowing the read continue.

C. Write to main memory (with write-back cache) :

1. This problem occurs when another bus master is performing a memory write to a line containing stale data. The bus master updates one or more locations in memory that are also contained within the cache.
2. Even if the cache line is not capable of data snarfing, it could invalidate that cache line, causing a mistake.
3. Since the line has been marked 'modified', it indicates that some or all of the information in the line is more current than the corresponding data in the memory.
4. The memory write being performed by another bus master will update some item within the memory line. By invalidating the line in cache it would quite probably discard some data that is more current than that within the memory line.
5. If the cache permits the bus master to complete the write, and then flushes the cache line to memory, the data just written by the bus master may be over-written by stale data in the cache line. The correct action would be to back-off the bus master, before it is able to complete the write to memory.

6. The cache controller then seizes the bus and performs a memory write to update this stale line in the main memory. In the cache directory, the cache line is now invalidated because the bus master will update the memory cache line immediately after the line is flushed. The cache then removes back-off signal, permitting the bus master to reinitiate the memory write operation. When the bus master completes the write to memory, the memory line will contain the most updated data.

11.10 Pentium Processor Cache Unit :

1. Pentium implements separate data and code cache referred to as split cache.
2. For each of these caches, line size = 32 bytes and data bus size is of 8 bytes (64 bits) in width.
3. Hence, a burst of 4 consecutive transfers is required to fill a cache line.
4. Implementing Pentium processor with L2 cache provides highest performance.
5. Since it uses write-back policy, the Pentium processor's L1 data cache introduces additional complexity into cache consistency logic. The data cache can use write-through policy or a write-back policy on a line-by-line basis.

When the Pentium processor's data cache utilizes a write-through policy, its operation with an L2 cache is conceptually identical to that of 486. The following descriptions assume that both the Pentium processor's L1 cache and its L2 cache are using write-back policies.

11.10.1 Memory Reads Initiated by the Pentium Processor :

1. When either of the execution units or prefetcher requests the information from an internal cache, the request is immediately fulfilled if a L1 cache hit occurs.
2. But in case of internal cache miss, the Pentium processor performs a cache line fill to read the target line from external cache.
3. The L2 cache gets memory read bus cycle and checks its directory for the copy of requested information.
4. If it's a hit on L2 cache, the L2 cache notifies the Pentium processor that the address is cacheable by enabling the KEN#. The L2 cache (controller) then accesses its cache SRAM memory, satisfying the cache line fill in a burst of four consecutive 64-bit transfers.
5. If it results in a miss on the L2 cache also, the L2 cache passes the bus cycle to the system bus.



6. The non-cacheable address logic (NCAL) determines whether the address is cacheable or not.
7. If it is determined to be non-cacheable, the NCAL notifies the L2 cache and the processor, that the addressed location cannot be safely cached. Hence, the bus cycle is not converted into a cache line fill and a single-transfer bus cycle is run to fetch the requested information directly from memory.
8. If NCAL determines the address is cacheable, it indicates L2 cache controller and the processor about the same, and causes a read cycle to be converted into a cache line fill for both the L2 and L1 caches.
9. Since the access is from slow external memory, wait states will be inserted in the transfer. The L2 cache copies the first 64 bits into its cache line-fill buffer, while simultaneously forwarding it to processor and indicating that valid data is present on the processor's local data bus.
10. The information requested for is contained in the first quadword and hence the information is immediately given to the internal requester, but three additional transfers must occur to complete the cache line fill. Once the entire line has been received, both the L1 and L2 caches copy the lines from their respective line-fill buffers into their respective caches.

11.10.2 Memory Writes Initiated by PENTIUM Processor :

1. The data cache can use either a write-through or a write-back policy.
2. When using write-back policy with an L2 cache, special care has to be taken when interacting between the Pentium processor cache and external cache to ensure cache coherency.

Write Misses :

1. The L1 data cache controller checks the target address of the memory write to determine if the copy of the target memory line exists in it.
2. In case of a write miss, the Pentium processor initiates a single transfer bus cycle to the target location. The L2 cache controller checks its directory to determine if a copy of the target line is resident in its cache.
3. If L2 cache hit occurs, then write is performed in L2 according to write-back or write-through policy.
4. If L2 cache miss occurs, the action of L2 cache depends on whether or not it supports allocate-on-write.

5. If the L2 cache does not have allocate-on-write capability, the L2 cache will pass the data to main memory. When the main memory completes the write, it activates ready to the L2 cache and the processor to notify them that the cycle has completed.
6. If the L2 cache supports allocate-on-write, it passes the data to the main memory as described above, but will then perform a L2 cache line fill from the same line.

Write hits and Write-once policy :

- Assume that a new line has just been read from memory and the line of data is brought in the L1 cache. Since the line passes through the L2 cache, a copy of the line also exists in the L2 write-back cache and in memory.
- Now suppose one of the execution units writes to same line. The L1 data cache controller will check its directory and find that the target line is the data cache. This L1 cache line will be modified. The L1 data cache line now has modified data and its directory entry is updated to reflect the 'modified' state of the cache line.
- Now if another bus master reads from the same line in memory. The L2 cache, unaware that the L1 data cache has modified this line, snoops the address given by the other bus master. The L2 cache checks its directory and finds a cache hit on a clean line. It takes no action because it feels that the other bus master will get valid data from memory.
- The problem is that the L2 cache was not notified by the L1 data cache that the line had been modified, and hence the other bus master gets stale data from memory.
- The solution to the above problem is use of write once policy. This policy says to use write-through for the first time and write-back thereafter.
- 1. When a line is initially placed in the L1 data cache, it is marked as 'shared' and makes the use of write-through to the L2 cache on first write hit to the cache line in L1.
- 2. Thus when the first time one of the execution units writes to the line, the line will be updated and the write operation written-through L2 cache. Hence, the L2 cache will also be updated and its directory entry updated to indicate that the line has been 'modified'. Now that the L2 cache directory indicates that this line has been 'modified', it will snoop correctly and will not permit any bus master to read the stale data from the corresponding line in memory. After using write-through, the L1 data cache changes the state of the line from 'shared' to the 'exclusive' state.

- Now, when an execution unit again writes to this line, the L1 data cache finds the target line in the data cache and in exclusive state. The L1 line is modified and its directory entry is updated to the 'modified' state, indicating that the line has been modified. All these subsequent writes to this line are performed using write back policy.

11.10.3 Memory Reads Initiated by Another Bus Master :

1. If a snoop read hit occurs to a line that has been modified, the L2 cache causes the bus master to back-off and simultaneously passes the address to the Pentium processor L1 cache so that it can also snoop the address.
 2. If the L1 cache controller detects a snoop read hit to a modified line, it performs a burst write-back cycle to update memory. After the write-back has completed, the L2 cache controller removes the back off signal and allows the bus master to complete the memory read.

11.10.4 The MESI Model :

- Based on the discussion in the previous sections we can show the state transition diagram of a line in cache as shown in Fig. 11.10.1.
 - Besides the basic Pentium read/write and another bus master read/write operations, there are some more cases shown in the MESI model.
 1. Internal snoop hit is to be considered in case when the data required by the processor is there in code cache or vice versa. In such cases the modified line is written back to the main memory and then invalidated, while a non modified line is simply invalidated.
 2. Flush# signal indicates the L1 cache to clear the entire cache. WBINVD indicates to clear a particular line in the L1 cache. In case of a Flush# signal or WBINVD, also the modified line is first written back to the main memory and then invalidated while non modified line is simply invalidated.
 3. In case of a INVD signal that simply indicates to invalidate a line without even writing it back for a modified line also, the processor simply invalidates the corresponding line.

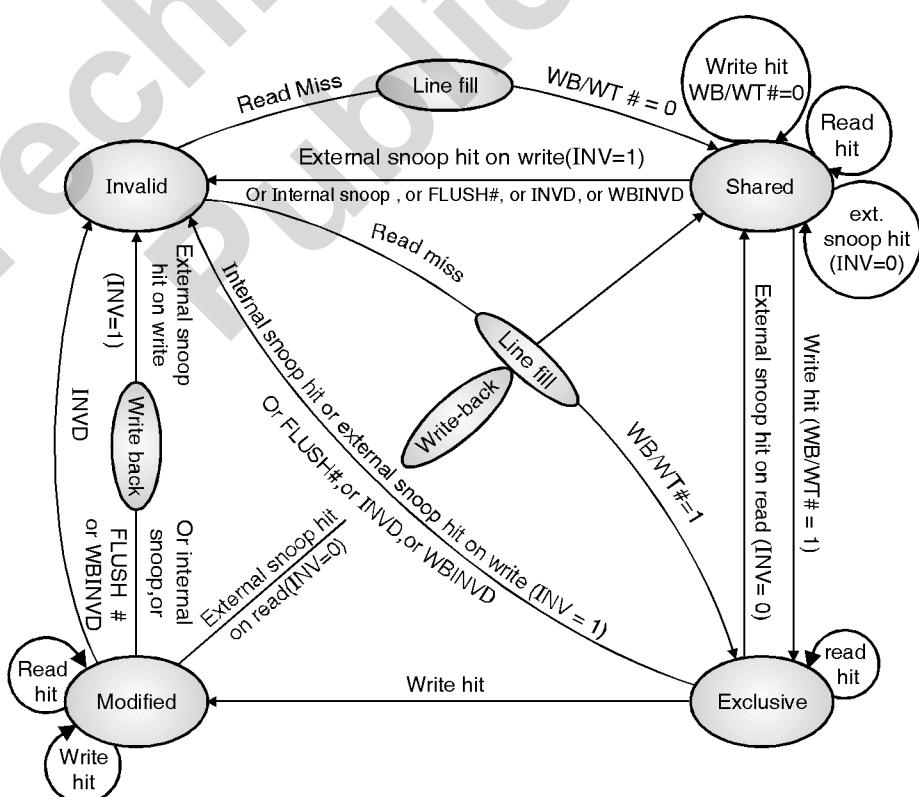


Fig. 11.10.1 : State diagram of MESI transition states as in Pentium's Data cache

11.11 Input / Output System :

- There are a wide variety of peripherals or I/O devices that deliver different amounts of data at different speeds and in different formats. All these devices are slower than CPU and RAM and hence to interface these devices to the CPU there is a need of I/O modules.
- Input/output module is interface to CPU and memory with one or more peripherals.
- The general model of I/O module interfacing with system bus is shown in Fig. 11.11.1.

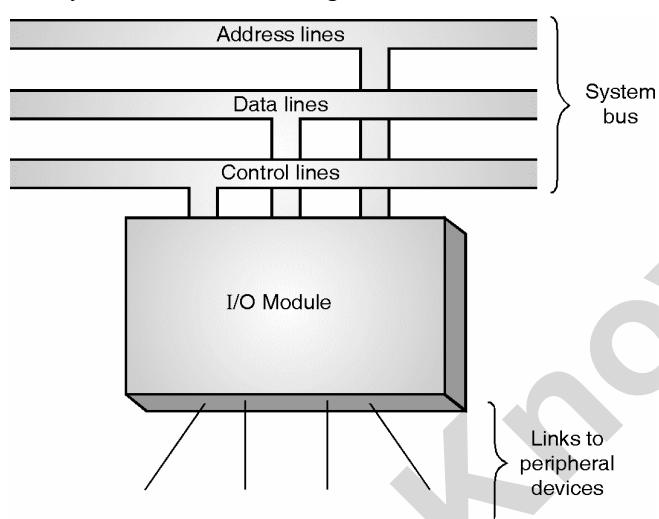


Fig. 11.11.1 : General model of I/O module interface

- The various functions of the I/O module involve :
 1. Issue of control and timing signals
 2. Communication with CPU
 3. Communication with peripheral
 4. Buffering of data between the CPU and peripheral and
 5. Detection of errors
- The internal block diagram of I/O module is shown in Fig. 11.11.2.

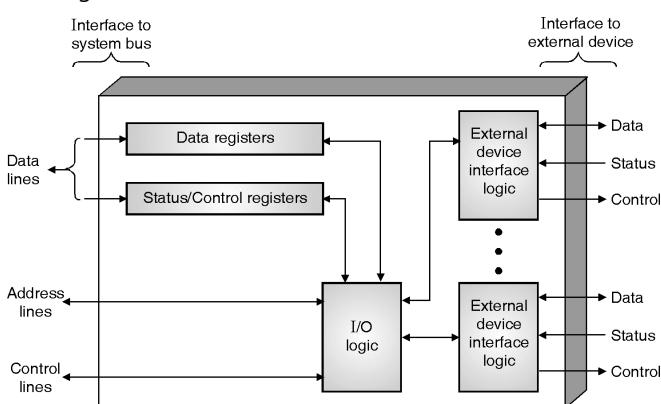


Fig. 11.11.2 : Internal block diagram of an I/O module

11.11.1 Parallel Versus Serial Interface :

- The word, **communication** specifies, data transfer between two points.
- The data may be a digital or analog in nature.
- We will consider only digital data transfer because microprocessor is digital circuit. Suppose you want to transfer data from Point A to Point B. There are two possible ways of doing it :
 - (1) Parallel data transfer
 - (2) Serial data transfer.
- For parallel data transfer, we can use 8255. Two 8255's are connected, one at each side.
- The Port A of 8255.
 - (1) At point A is connected to Port A of 8255.
 - (2) At point B. So the data transferred is of 8 bits at a time. For implementing this communication, we want 8 lines of PA interconnected and line will be the common ground between two points.
- In serial data transfer the data is transferred serially on a single line, the same hardware used for parallel data can also be used to implement this. Instead of connecting all 8 lines connect single line from Port A of 8255 :
 - (1) To Port A of 8255.
 - (2) To implement above communication we require one line of Port A interconnected and second line i.e. common ground between two points.

Now let's compare the specified 2 methods of data transfer.

Sr. No.	Parallel	Serial
1.	Parallel lines of 8/16/32 bits. Hence 8/16/32 bits can be transmitted simultaneously.	Only 1 bit is transmitted at a time.
2.	The data transfer is comparatively faster.	The data transfer is comparatively slower.
3.	Due to so many parallel paths 'crosstalk' among different bits is possible.	No 'crosstalk' possible.

Sr. No.	Parallel	Serial
4.	This cannot be used for distant communication.	This can be used for distant communication.
5.	More parallel hardware is required.	Less parallel hardware required.
6.	It is comparatively costlier.	It is comparatively cheaper.

- In these two methods the cost of connecting two distant points, is the main factor. So though the parallel data transfer is faster, it is preferred for small distances only. But for long distances, serial data transfer is preferred.
- In serial data transfer the 8 bits of data is converted into serial 1 bit data; using shift register (parallel in serial out mode). These serial bits are transferred on single line using serial I/O data transfer.
- To transfer 8 bits of data, it will require 8 clock pulses. On the other side exactly opposite process is done. These serial 8 bits are accepted and converted to parallel form to get 8 bits of data. This process is shown in Fig. 11.11.3.

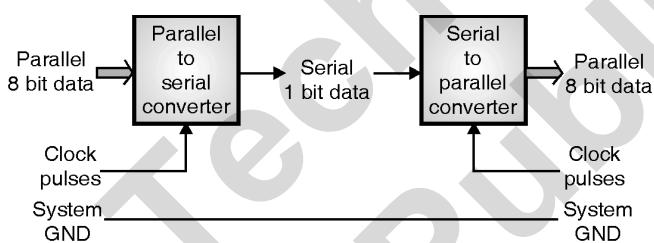


Fig. 11.11.3 : Serial I/O

11.11.2 Types of Communication Systems :

The communication systems are classified on the basis of transmission :

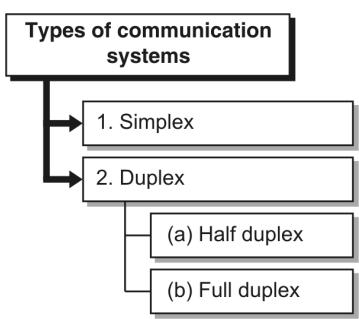


Fig. 11.11.4 : Types of communication systems

1. Simplex :

- The simplex is one way transmission.
- The connection exists such that data transfer takes place only in one direction.
- There is no possibility of data transfer in the other direction.
- System A is transmitter and system B is receiver only.

2. Duplex :

The duplex is two way transmissions. It is further divided in 2 groups :

(a) Half duplex :

- It is a connection between two terminals such that, data may travel in both the directions, but transmission activated in one direction at a time.
- This indicates that the line has to turn around after communication is complete in one direction.

(b) Full duplex :

- It is a connection between two terminals such that, data may travel in both the directions simultaneously. So it will contain one way transmission or two way transmission at a time.

11.12 I/O Modules and 8089 IO Processor :

An Input/output device can never be connected directly to the processor. It always has to be interfaced using an I/O module.

I/O module is required for the following reasons :

1. I/O devices are normally slower than the processor and also have different speeds. Hence if there is no I/O module, the processor will have to wait for long time for the I/O devices. Hence I/O module works as a buffer between the processor and I/O device to hold the data for the required time
2. Each I/O device has different data bus width. I/O module does the required width conversion.
3. Each I/O device has different protocol to be followed. Some use serial communication, some use parallel,

some have handshaking signals etc. Hence I/O module communicates according to the protocol required by the I/O devices.

11.12.1 I/O Module :

1. **Need of input module :** each output device operates at a different speed, has different data format and different protocol.

Also, most of the I/O devices are slower than the speed of the processor. Hence, an IO module is used to interface the IO device to the processor.

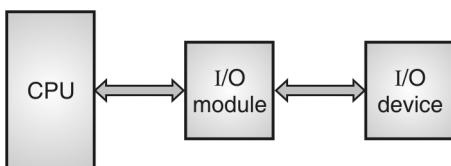


Fig. 11.12.1 : Input output module

Block diagram of I/O module :

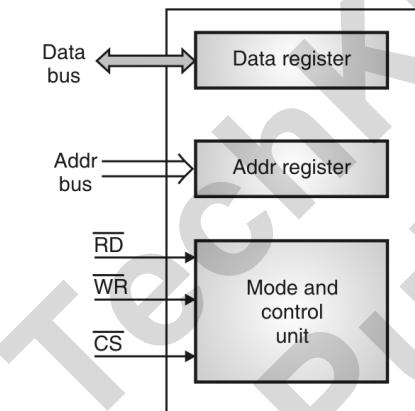


Fig. 11.12.2 : Block diagram of I/O module

- Data register is used to store the data given by the input device to be forwarded to the processor OR given by the processor to be forwarded to an output device.
- Address register is used to provide address of the IO device to be accessed.
- Mode and control unit indicates the mode of operation for the I/O module, as well as controls the transfer of data between the IO module and IO device, as well as IO module and CPU.

11.13 Types of Data Transfer Techniques : Programmed I/O, Interrupt Driven I/O and DMA :

- There is yet another method of classifying the interfacing of I/O devices based on how and when the data is transferred between the processor and I/O devices.
- There are three types under this method of classification namely programmed I/O, interrupt driven I/O and DMA (Direct Memory Access).

Definition of polling :

Polling is a mechanism, wherein the processor checks each and every device for it needs a service or not.

11.13.1 Programmed I/O :

- In the programmed I/O method of interfacing, CPU has direct control over I/O.
- The processor checks the status of the devices and issues read or write commands and then transfers data. During the data transfer, CPU waits for I/O module to complete operation and hence this system wastes the CPU time.
- The sequence of operations to be carried out in programmed I/O operation are :
 1. CPU requests for I/O operation.
 2. I/O module performs the said operation.
 3. I/O module updates the status bits.
 4. CPU checks these status bits periodically. Neither the I/O module cannot inform CPU directly nor can I/O module interrupt CPU.
 5. CPU may wait for the operation to complete or may continue the operation later.
- IC 8255 is generally used as a I/O module for programmed I/O method of interfacing.
- A common programming task is the transfer of a block of words between an Input/output device and memory.

- Fig. 11.13.1 gives a flowchart for transferring a block of data.

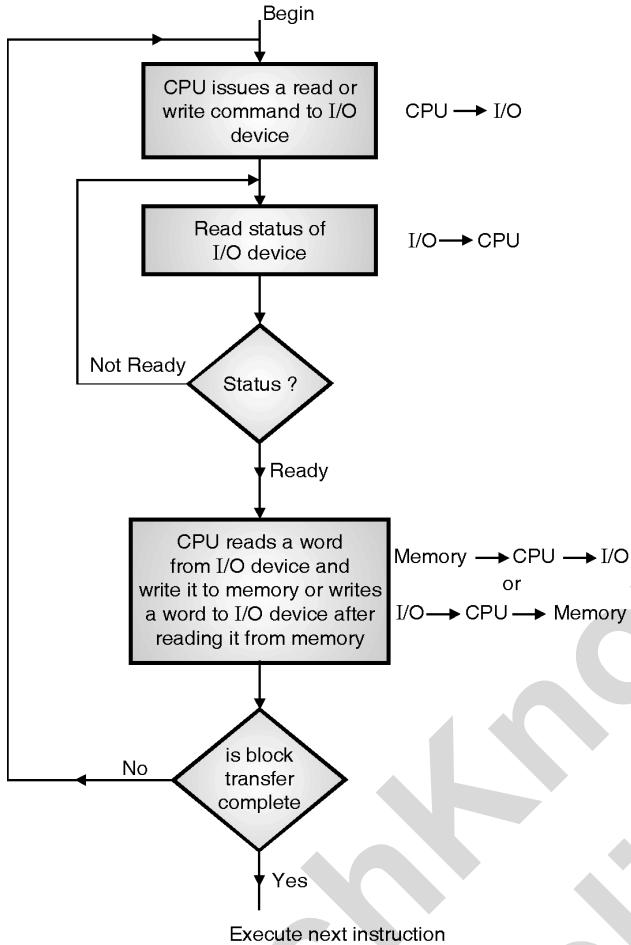


Fig. 11.13.1 : Transferring a block of data using programmed Input/output

11.13.2 Interrupt Driven I/O :

- Interrupt Driven I/O overcomes the disadvantage of programmed I/O i.e. the CPU waiting for I/O device.
- This disadvantage is overcome by CPU not repeatedly checking for the device being ready or not instead the I/O module interrupts when ready.
- The sequence of operations for interrupt Driven I/O is as below :
 - CPU issues the read command to I/O device.
 - I/O module gets data from peripheral while CPU does other work.
 - Once the I/O module completes the data transfer from I/O device, it interrupts CPU.

- On getting the interrupt, CPU requests data from the I/O module.
- I/O module transfers the data to CPU.

- After issuing the read command the CPU performs its work, but checks for the interrupt after every instruction cycle as seen earlier in this chapter.
- When CPU gets an interrupt, it performs the following operation in sequence
 - Save context i.e. the contents of the registers on the stack
 - Processes interrupt by executing the corresponding ISR
 - Restore the register context from the stack.
- IC 8259 has 8 interrupt lines and is used as a I/O module when Interrupt driven I/O is used.
- The interrupt driven Input/output mechanism for transferring a block of data is shown in Fig. 11.13.2.

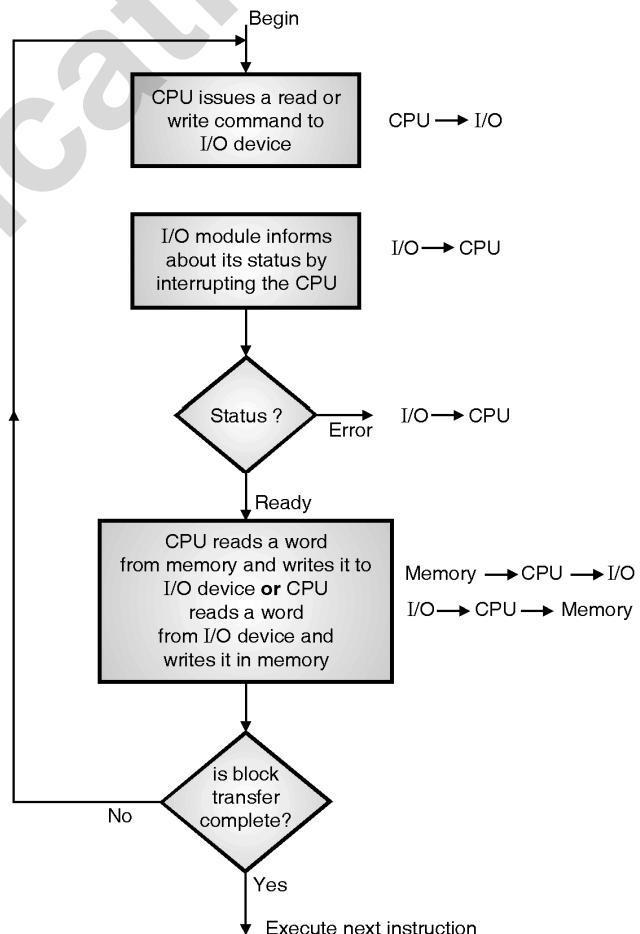


Fig. 11.13.2 : Transferring a block of data using interrupt driven Input/output

Transferring a word of data

- CPU issues a 'READ' command to Input/output device and then switches to some other program. CPU may be working on different programs.
- Once the Input/output device is ready with the data in its data register, Input/output device signals an interrupt to the CPU.
- When the interrupt from Input/output device occurs, it suspends execution of the current program, reads data from the port and then resumes execution of the suspended program.

11.13.3 DMA :

- DMA stands for Direct Memory Access. The I/O module can directly access (read or write) the memory using this method.
- Interrupt driven and programmed I/O require active operation of the CPU, hence transfer rate is limited and CPU is also busy doing the transfer operation. DMA is the solution for these problems.
- DMA controller takes over the control of the bus from CPU for I/O transfer.
- The internal block diagram of a DMA controller of the I/O module for DMA method of I/O interfacing is shown in the Fig. 11.13.3.

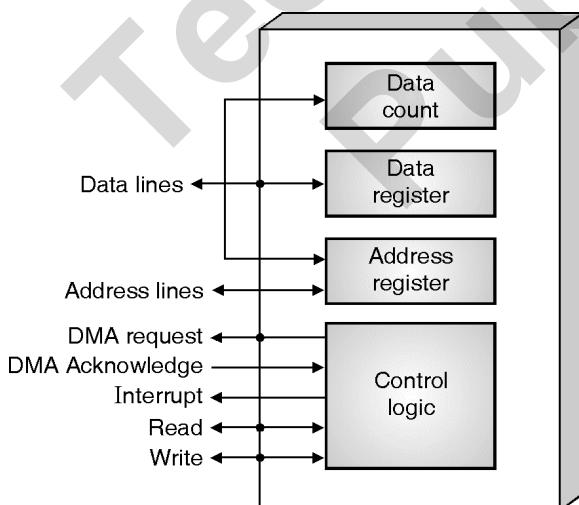


Fig. 11.13.3 : Internal block diagram of DMA controller

- In Fig. 11.13.3, you will notice that there are various registers like data count, data register and address register.

- The address register is used to hold the address of the memory location from which the data is to be transferred. There may be multiple address registers to hold multiple addresses.
- The address may be incremented or decremented after every transfer based on the mode of operation.
- The data count register is used to keep a track of the number of bytes to be transferred. The counter register is decremented after every transfer.
- The data register is used in a special case i.e. when the transfer of a block is to be done from one memory location to another memory location.
- Also you will note in the Fig. 11.13.3 the read and write signals are bidirectional.
- The DMA controller is initially programmed by the CPU, for the count of bytes to be transferred, address of the memory block for the data to be transferred etc.
- During this programming of the DMAC (DMA controller), the read and write lines work as inputs for the DMAC.
- This is because the CPU has to tell the DMAC whether it is reading or writing from the DMAC.
- Once the DMAC takes the control of the system bus i.e. transfers the data between the memory and I/O device, these read and write signals work as output signals.
- They are used to tell the memory that the DMAC wants to read or write from the memory according to the operation being data transfer from memory to I/O or from I/O to memory.
- The speciality of DMA is that the CPU carries on with other work while the DMA controller deals with transfer of data. DMA controller sends a signal when finished.

11.13.4 DMA Transfer Modes :

- There are various modes of operation used to transfer the data between the memory and I/O device by the DMA controller.
- The four major methods used are discussed below :

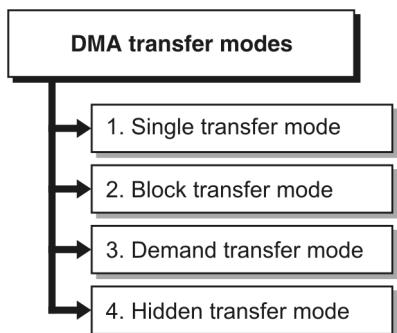


Fig. 11.13.4 : DMA transfer modes

1. Single transfer mode :

- In single transfer mode, the device is programmed to make one byte transfer only after getting the control of the system bus.
- After transferring one byte the control of the bus will be returned back to the CPU.
- The word count will be decremented and the address decremented or incremented following each transfer.
- The disadvantage in this method is that the I/O device has to wait for a long time after every transfer for the extra request grant signals.
- The advantage is that the CPU has not to remain out of the system or not having the access of system bus for longer time, instead only for one transfer.

2. Block transfer mode :

- In block transfer mode, the device is activated by DREQ (DMA Request) or software request and continues making transfers during the service until a Terminal Count (i.e. the counter becomes zero), or an external End of Process (EOP) is encountered.
- The disadvantage is that the CPU has to remain out of the system or not having the access of system bus for longer time, until all the bytes in the block are transferred.
- The problem further increases in case if the I/O device is slower and the system is waiting for the I/O device to complete its operation, thus the CPU has to wait for very long period in this case.
- The advantage is that the I/O device gets the transfer of data at a very faster speed.

3. Demand transfer mode :

- In demand transfer mode, the device continues making transfers until a Terminal Count or external EOP is encountered, or until DREQ goes inactive.
- Thus, transfer may continue until the I/O device has exhausted its data handling capacity.
- Thus this method is said to be a trade off between the earlier two methods. If the I/O device is fast enough it will keep on getting data and need not wait for extra time for the request grant signals as in the single transfer method.
- Also the CPU has not to wait for longer time in case if the I/O device is slower, because if the I/O device is slower the transfer terminate.

4. Hidden transfer mode :

- In hidden transfer mode, the DMA controller takes over the charge on the system bus and transfers data when processor does not needs system bus.
- The processor does not even realize of this transfer being taken place.
- The processor does not needs the system bus when it is performing some execution of an instruction in the ALU or certain instructions that do not need the system bus access at all. It happens mostly between the machine cycles.
- Hence these transfers are hidden from the processor.

Review Questions

- Q. 1 What are different memory parameters ?
- Q. 2 Give the classifications of primary and secondary memories.
- Q. 3 Write a short note on memory hierarchy.
- Q. 4 Write a short note on memory allocation.
- Q. 5 What is the principle of locality of reference ?
- Q. 6 What are the two different types of cache architecture? Explain look-aside cache design.
- Q. 7 What are the advantages and disadvantages of look-through cache design ?
- Q. 8 Write a short note on cache mapping techniques.
- Q. 9 What are the parameters considered for performance analysis ?



- | | | | |
|-------|---|-------|--|
| Q. 10 | What are the conditions for consistence occurrence of problem ? | Q. 16 | With neat block diagram explain I/O module. |
| Q. 11 | Write a short note on pentium processor cache unit. | Q. 17 | Write a short note on direct memory access. |
| Q. 12 | Write a short note on write hits and write-once policy. | Q. 18 | What are the different transfer modes in DMA ? Explain. |
| Q. 13 | With neat state diagram explain MESI model. | Q. 19 | Explain various types of ROM : Magnetic as well as optical. |
| Q. 14 | Differentiate between parallel and serial interface. | Q. 20 | Interface 7 KB EPROM and 6 kB RAM to a processor with 16-bit address and 7-bit data bus. |
| Q. 15 | What are the different types of communication system ? Explain. | | |

TechKnowledge
Publications